Coordinating Filtering Strategies in Cooperative Agent Communities

vorgelegt von
Master of Science
Dragan Milosevic

von der Fakultät IV – Elektrotechnik und Informatik
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften
– Dr.-Ing. –

genehmigte Dissertation

Promotionsausschuss:

Vorsitzender: Prof. Dr. Hartmut Ehrig
Berichter: Prof. Dr. Sahin Albayrak
Berichter: Prof. Dr. Klaus Obermayer

Tag der wissenschaftlichen Aussprache: 9. Juni 2006

Berlin 2006

D 83

Abstract

A great easiness to produce and publish information has influenced that users typically have to search numerous distributed sources while satisfying their needs. Besides, they exhibit great difficulties to formally represent their preferences, and thus many manually created filtering requests have weak chances to initiate a delivery of needed results. As a delay for providing the sought data becomes a more and more critical factor, underlying engines have also to carefully address a trade-off between result relevance and response time. The end users finally expect as good results as possible, and consequently a system should be able to always deliver them in spite of more or less severe internal failures.

Ignoring a necessity to address information retrieval in both distributed and dynamic enough manner is a major drawback for many existing search engines that try to survive an ongoing information explosion. These systems also process queries in a direct manner, and therefore the expertise from many excellent, and already fulfilled requests, remains unutilised. The resource availability is also often ignored, as developed filtering solutions are usually tested in protected conditions, where nothing unexpected can happen.

The essence of a proposed solution for performing distributed filtering is in installing communities around different databases, as well as setting cooperation mechanisms that are able to find the right sources for a given request. The deployed exploration techniques, which first identify the usable old requests, and then utilise them to adapt the actual one, guarantee that even wrongly formulated information needs will intelligently explore the potentially relevant areas of interest. The resource availability is taken into account by deploying different filtering agents, each knowing their own searching heuristics, inside communities, as well as coordinating them to alleviate a selection of those for which the needed resources are currently highly loaded. A coordination of multiple strategies can be also naturally used to increase robustness and provide self-healing functionalities, where an alternative agent is selected in a case of a failure of an initially selected one.

The cooperation, exploration and coordination schemes are realised in JIAC IV, being a framework for developing multi-agent systems. Every mechanism is developed as a set of components, which might be easily replaced with their improved versions, and which thus increase a flexibility of a given implementation. An applicability of these approaches is practically illustrated in a personal information assistant (PIA), whose filtering system critically depends on cooperation, exploration and coordination mechanisms.

Experimental results show that the given cooperation scheme manages to successfully locate the right communities for a given request in spite of the dynamic changes of their underlying content. The exploration approach also has superior performances, regarding a number of off-topic results, when being compared with pure collaborative filtering and mutation & crossover techniques. The proposed coordination approach finally scales well, and it can be applied even inside communities with thousands of agents. By estimating a load of resources before deciding which strategy will do filtering, it effectively eliminates long lasting jobs with duration over 1000 seconds, and thus reduces the average response time. It can finally improve itself during runtime by successfully using every single job, being shorter than 10 seconds.

Table of Contents

Chapter 1
Introduction

Even very simple online services nowadays silently assume the underlying application of sophisticated filtering strategies in surviving the course of finding only relevant data in a great abundance of the electronically available information. The obvious examples are services for supporting activities such as booking a holiday, figuring out which streets are currently with a traffic jam, getting a weather forecast for the wanted place, finding the most appropriate house to rent, locating an open supermarket from a favourite chain in a neighbourhood, buying a book in an online book store, searching for the articles from the particular area of interest, and so on. Not only that all these services have to deliver as good results as possible, but what is even more important, they have to provide them in an extremely short response time. Users are impatient beings, and leaving them to wait too long might be very dangerous, especially in the modern online environments, where concurrent applications are only few clicks away [224].

The great seriousness of the mentioned application domains has resulted that there are already many more or less specific filtering strategies [127], which are generally quite successful in providing the reasonably good results after a short delay. Two well-known examples are indexing techniques for document retrieval [59] and collaborative filtering for finding similar users [131]. The former, being nowadays deployed in search engines [58][223], is capable of finding within only few seconds all documents which have the given keywords, even from the collection with the millions of articles. The latter, being successfully used in online shopping portals for increasing the selling rate [159][221], is able to quickly recommend additional interesting products by finding what users with similar buying patterns have liked. These two and many other strategies are obviously tailored to successfully work in completely different conditions [221], and unfortunately, a real challenge becomes finding out which one can provide, for a particular information request in a given runtime situation, the best ratio between a relevance of results and the response time.

According to the author's point of view, a variety of strategies is naturally influenced since every single application domain has really many properties or constrains, which can be used in numerous ways in addressing this unavoidable trade-off between the relevance of results and the time needed for finding them. For example, booking a holiday can be seen as one very complex activity, having constrains such as location, price range, food

preferences, accommodation facilities, entertainment requirements, and so on [83][215][216]. In order to be able to quickly deliver holiday recommendations for impatient users, a particular strategy can first find a few candidate hotels based on a specified location and then further carefully examine them concerning other given requirements. This and many other sophisticated searching heuristics [181], which have made a foundation of each and every filtering strategy, can hopefully successfully both combine and efficiently exploit all available constraints to discover as good recommendations as possible in the permitted response time.

By reason of always being deployed on a real hardware, even a filtering strategy with a brilliant heuristic will definitely fail to quickly deliver the requested results when the needed system resources are not available as it has been expected. It is worth nothing the sophisticated processing that the used algorithm performs, in a case where the essential data cannot be efficiently loaded from the underlying collection, for example because of the overloaded database server. At the same time, the availability of system resources in a real environment is continuously changing, which certainly makes each and every system that is solely based only on a single filtering strategy to be very vulnerable. Much better robustness can be achieved by having a set of filtering strategies with different resource requirements, and by selecting the strategy for which the favourable resource situation exists.

In spite of being very careful while analysing the applicability of every strategy in the given situation, the unexpected behaviour and unavoidable mistakes are always possible. There are many novel and mysterious strategies about which not enough past experience is available. They certainly have to be tried by getting sometimes the requests to process, being a chance for them to prove their capabilities. The excellent results can be hopefully found, but it might happen that they are not able to deliver expected results, and therefore a recovery procedure is necessary. Although a useful experience about a chosen strategy has been acquired irrespective to the occurred failure, end users are much more important, and consequently they cannot be left without the requested results. A system should be obviously able to do self-healing and to efficiently compensate for the lost time, probably by selecting a well-known filtering strategy, being very fast and thus especially appealing in the time critical situations after failures.

The unavoidable implication of services for planning holidays, browsing bookstores or finding streets, also assumes that the underlying information is typically scattered in the vast number of distributed sources. While the one source contains the needed data about the currently top holiday offers, another one is the database with summaries about books that are available for sale. Since there are thousands of such distributed databases whose underlying content has besides a very chaotic global order, their individual inspection is at least impractical. Users have no time to look all around, and their attention has become a precious resource in modern and fast living world [276]. Without the autonomous tools for identifying where the potentially relevant documents are located, the information rich sources are almost useless.

These numerous distributed sources are also always more or less dynamic, where both the new information becomes available, as well as the old has changed or even has turned out to be outdated. It might happen that an online portal for giving the information about houses, which are available for renting, is no more maintaining the data about favourite

city areas. An unlucky user is therefore forced to discover the alternative sources either by querying search engines that hopefully know everything or by explaining its problem and asking friends for a useful advice. The big weaknesses of such reasoning are at least twofold. On the one hand, there is unfortunately an open doubt that the centralised search engines, such as Google [58], AltaVista, Yahoo, and numerous others [223], will not be able to adequately respond to the information explosion in the future. Even though they provide much more capabilities today than ever before, the information that is potentially available from a World Wide Web continues to grow exponentially [187]. It is therefore logical to expect that a relative ratio between indexed and un-indexed web pages is going to slowly decrease in the future, and consequently the search engines will be less and less competent. On the other hand, a wiseacre, who surely knows from where the necessary information can be currently retrieved, is usually difficult to be found. Although there is certainly such a person, every user knows only a tiny fraction of others, and consequently it is hard to believe that exactly one of its friends is capable to give a desperately needed advice. The new technologies for effectively managing more or less dynamic distributed sources are consequently becoming critical means for first locating, then retrieving, and finally processing the sought data [47]. The author's point of view is that the distributed knowledge management, supported by multi-agent communities that mutually cooperate, can be exactly one instance of the necessary technology.

The existence of communities is also motivated by making applicable many filtering strategies [30][50][82][94][196][213][284] and data mining algorithms [105][127] that are always more or less scalable. The design and structure of many strategies may not be appropriate at all for a very large when for instance the run times behave exponentially to the size of the underlying collection. Two nice examples can be simultaneous clustering with dynamic keyword weighting [105] and self organising maps [148]. The former gives the remarkably good results, but unfortunately only on the collection with two thousand documents. The latter is applied on seven million short abstracts, but it took 6 weeks on a six processor computer to train such a SOM. For these two and many other strategies, it is thus crucial to have the collections with desired properties, where they can give excellent results. While one strategy might have great difficulties with highly dynamic underlying collections due to the very expensive pre-processing activities, another one typically has problems when available documents are huge. Under an assumption that there are already numerous collections with the highly different properties, the installation of appropriate cooperative communities around them should provide the necessary advanced means for retrieving information in a distributed manner.

Although the multiple communities with intelligent agents have a necessary potential to handle the scattered information, they still have to communicate with users. Without clearly understanding what kind of information is searched for, the chances that the right recommendations will be offered are very weak. The precise specification of information needs is therefore imperative, but an imperfect nature of humans unfortunately introduces significant difficulties while creating compact and understandable queries. The problem is even more severe as not only that the users have to explicitly articulate their thoughts about their current interests, but also they should have a solid familiarity with sometimes difficult query syntax, as well as the generalised knowledge about which results might be potentially retrieved. On the one hand, a direct usage of online portals requires that a user is familiar with many advanced options, being usually essential for getting the best results.

On the other hand, to create a good query at least a global idea about the available data is needed. It makes for example no sense to search for authors' family details in a database with books, where only bulk authors' names are stored. The creation of a query, having reasonable chances to deliver expected results, is obviously a hard task, and consequently many of them will be wrongly formed. The straightforward treatment of these bad queries, being most often performed by the available search engines, is the main reason of having an extremely high percentage of queries without results [260]. The many provided results are also not related to the information needs that users had in mind while posting a query. Users at the same time do not typically believe that their wrong queries are main reasons for not getting the expected results, and consequently they will lose their confidence in a particular filtering engine, and even start to utilise another one that will better satisfy their information needs.

Since the users' habits regarding a creation of the queries cannot be easily influenced, the intelligent processing techniques on the side of a filtering engine seem to be necessary. One such technique is a so-called intelligent query answering [127], where the underlying intent of a query is analysed in order not to provide only exactly what is being asked, but also to explore the potentially relevant information areas. The user, who is looking for the nice holiday destination, might also get the data about renting a car in spite of not being explicitly asked, mostly by reason of being the commonly requested extra information for a particular holiday offer. The provided results will be much more diverse, having better chances to satisfy not only the imposed information needs, but also the needs that were hidden from a user in the time of creating a query.

Even though the provision of the extra information seems to be very useful, too much additional data, being not explicitly requested, might have a negative effect. A user can find to be overloaded with something, being maybe irrelevant from its point of view, and therefore the level of exploration should be carefully controlled. Not thinking about the extent of exploration, being typical for very frequently used collaborative filtering, tends unfortunately to deliver results that do nothing with actual information needs. Although recommending the items that are liked by similar users sounds logical, these items will have a low usability when being far away from specified information interests. Offering for example the weather information about asked cities can help only when the explicitly asked sightseeing recommendations are also got. The deployed exploration mechanisms should be thus able to control the automatic changes of the specified information needs, in order to be able to alleviate situations where users are asking themselves which results really correspond to given preferences.

A very promising technique for performing such a controlled exploration happens to be automatic transformation of an actual query, being done under the guidance of already processed past requests. Many experts have already posted good queries while having in mind almost the same information interests as a novice user. It will be wonderful to first identify these sophisticated queries, and afterwards utilise the expertise from them to help an inexperience user get surprisingly good results. A wrongly formed query for example about tools for designing the multi-agent systems might lack many important keywords, which might be hopefully found from other queries that have been created by the domain experts. A positive effect of a fulfilled exploration is the provision of results for example

about many very helpful editors and working environments, which should have remained undiscovered without the applied automatic query transformations.

The definite advantage of utilising query adaptation techniques to achieve exploration relates to getting the full transparency regarding the deployed filtering strategies. A query is simply modified before being delegated for example to a particular indexer, which does not know anything about performed changes. A motivation from the begging to combine different filtering strategies in order to more efficiently exploit available resources and to increase the overall robustness might be also realised by seeing searching algorithms as black boxes. Since cooperation between multiple agent communities as well does nothing directly with particular filtering techniques, the design of any concrete searching strategy will be out of the scope of this thesis, being structured as follows.

Chapter 2 is the longest part of this thesis, giving a state of the art regarding the other systems that address the information filtering and retrieval problems. The thirty presented systems evidently demonstrate on the first place that there is no best technique for solving all filtering challenges. Chosen systems might be globally seen from two points of view. On the one hand, many of them are aware of a necessity to combine multiple strategies to achieve better performances. Their presentation is thus mainly focused on identifying the potential weaknesses of the realised combination, and on pointing out on the promising enhancements. On the other hand, the potential of combining different strategies remains undiscovered for some systems in spite of usually testing really many algorithms, as well as almost always remarking that different approaches are the best solutions in the various situations. These systems are typically presented by stressing the necessity to establish a combination between available strategies.

The cooperation, exploration and coordination great challenges are deeply discussed in Chapter 3, mostly by giving the appropriate scenarios, which facilitate an identification of even very small difficulties that are worth attention. The cooperation scenario is therefore pointing out first on the database selection problem, and then on the information fusion. The cornerstone challenge for exploration is the identification of usable old requests, and afterwards their application for adapting the actual one. The coordination scenario finally gives problems while tending to successfully learn to combine filtering strategies in order to optimally exploit the available resources, give fair chances to everybody, and recover after failures.

The proposed approaches for addressing mentioned challenges are deeply analysed in Chapters 4, 5 and 6. The first of them gives cooperation mechanisms, being the initially responsible for the received request. It is shown how the most promising communities are first found in order to be used for responding to the actual request. A proposed approach for finding communities is also fully aware of a dynamic nature of underlying collections, being achieved by a compact solution for being informed about changes. The second part of cooperation approach handles information fusion by introducing an efficient technique for re-ranking results, being found by different communities.

Chapter 5 gives the exploration approach, which is clearly separated into the activities of first finding usable old requests, then mining them, and finally adapting the actual job based on the mined data. The big difficulties of efficiently comparing highly dimensional requests are specially addressed by making the synergy between Euclidean distance and

Jaccard index. The major part of the exploration mechanisms is contained in performing the controlled adaptation of the actual request by both adding attributes that are missing, as well as modifying the already present ones in order to better represent the underlying information interests.

The most important coordination schemes are presented in Chapter 6. To make their presentation as clear as possible, the different aspects are gradually introduced. There are consequently resource aware, job aware, and self-healing coordination mechanisms that separately discuss about the optimal usage of available resources, the analyses of the job properties while selecting strategies, and a recovery after failures, respectively. Common to all coordination techniques is both the application of the proportional selection while deciding which agent will find results, as well as the adaptation of knowledge about the responsible strategies. These two activities are capable to intelligently handle a trade-off between exploiting the well known strategies and exploring the hidden capabilities of the novel ones.

The implementation details are, together with perspectives for further improvements, discussed in Chapter 7. The cooperation, exploration and coordination mechanisms are realised on a component basis mostly to provide the great flexibility while experimenting with different techniques. The presentation of every component, being currently available or planned for the future, always contains the ideas about the promising improvements of the supported functionalities.

Chapter 8 finally gives the performed evaluation of the realised approaches. The tests are made both in the simulated environment, where for example the small collection of requests is first manually generated and then used for comparing different approaches, as well as in the surrounding with real users, whose valuable relevance feedback points out on both applicability and usefulness of every approach.

The thesis is finished with Chapter 9, which briefly summarises its main contributions, regarding the proposed cooperation, exploration and coordination mechanisms.

Chapter 2
Related Work

The ability of efficiently locating the most relevant information at the right time [243] becomes an unavoidable ultimatum in a competitive information-oriented society [84]. In the direction of facilitating the access to the right information, many intelligent assistants have been invented to help users cope with the immense amount of data available on-line. Although they incorporate many sophisticated information filtering and machine learning techniques [127], finding the given document of interest can still amount to the Herculean task [59]. The underlying difficulties are certainly related to the amount of collected and accumulated data that keeps growing exponentially [187] across the wide variety of fields [98]. Internet and information overload are therefore still assumed to be synonyms [243] in spite of a dramatical increase of an efficiency of accessing the information [282], being a precious raw material of a digital age [82].

The following critical presentation of many existing systems aims at uncovering their inbuilt strengths and weaknesses, and thus motivates the realisation of the novel filtering solution that will try to provide better performances. The great difficulty for investigating other solutions, which tend to be a holy grail for the information filtering problems [235], is unfortunately concered with not having enough scientific publications describing them, being especially critical for the highly popular modern commercial systems. Probably the best example is currently the most widely used search engine Google, being presented in only one known, seven years old, scientific paper [58], which only describes the initial academic attempts towards exploiting the hyperlink structure of a Web. Such commercial systems are typically presented only through advertisement materials, and consequently they cannot contribute when the deployed algorithms are seriously taken into account.

One of the guidelines for deciding which systems are worth of being analysed relates to their scientific popularity, measured as the number of citations to them. The citations are found by CiteSeer (http://citeseer.ist.psu.edu/), a system being able to determine how many other articles are putting a given scientific paper as a reference. Figure 2.1 presents thirty different systems, which are sorted based on a number of citations, and form groups of highly, medium and low cited articles. Since the older systems have better chances to be more cited, the time interval, in which papers about the particular system have been published, is also given. On the one hand, the first publication about a GroupLens system [213] for example appeared in 1994. This paper [213] has afterwards greatly contributed in making a GroupLens to be the system with 460 citations, being the greatest number of

citation on Figure 2.1. On the other hand, a single available publication about a DisCover [154] for example dated from 2004, being probably a main reason for putting this system almost at the end of a list.

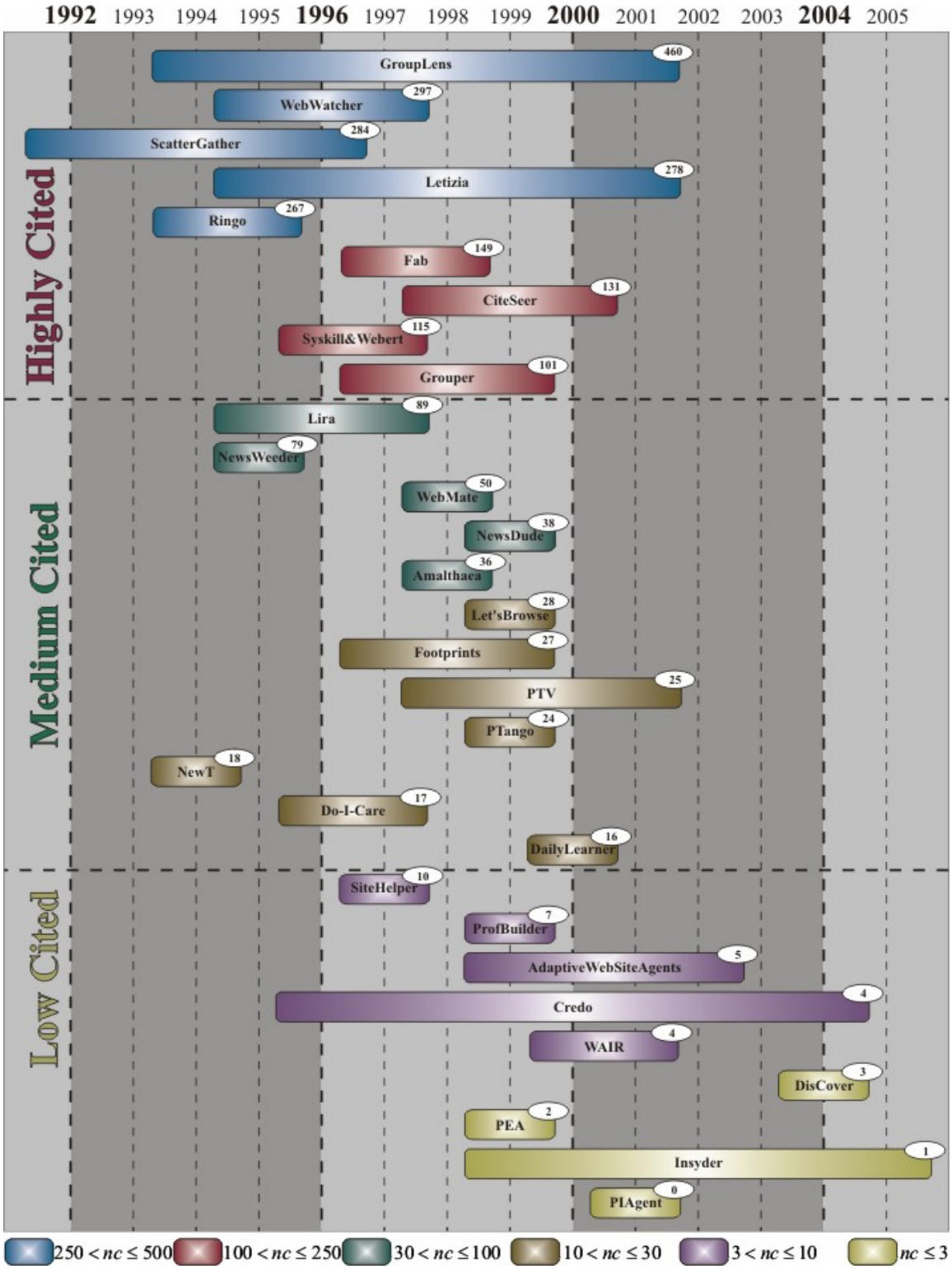

Figure 2.1: The citation and timeline distribution of thirty analysed information retrieval systems, where the utilised acronym nc should be associated with the number of citations

The combination between the timeline and a number of citations on Figure 2.1 can be also very practical in figuring out not only when the particular ideas were born, but also in which extent they were accepted by a scientific community. To facilitate such analysis Figure 2.2 classifies the same systems based on applied techniques, and Figure 2.3 gives the information about a main task that is fulfilled. By reason of assuming that the number of citations is important, a colouring of systems on Figure 2.2 and Figure 2.3 is preserved to illustrate their popularity. Solely from Figure 2.2 for example can be concluded that systems that utilise the collaborative filtering are quite popular, mostly due to GroupLens, Ringo and Fab, whose colouring points out that they belong to the group of highly cited. Figure 2.1 and Figure 2.2 together can further uncover that these collaborative filtering systems appear relatively early (GroupLens and Ringo dated in 1994) and the last known publication about them happends to be from 2001 (GroupLens and PTV). A combination between Figure 2.1, Figure 2.2 and Figure 2.3 might be for example exploited when the application domain of browsing filtering results is taken into concern. From Figure 2.1 and Figure 2.3 can be for example seen that the most popular system is a ScatterGather, whose known publications dated from 1992 to 1996. Figure 2.2 also shows that with an exception of CREDO, every other system, which facilitates a browsing through filtering results, utilises clustering techniques. CREDO is at the same time not so popular in spite of having many publications from period 1996-2004, being shown on Figure 2.1, which further uncovers that younger Grouper, while serving the same purpose as CREDO does, has also much more citations. By knowing exactly what is behind these two systems, the last observation might lead to the conclusion that suffix tree clustering used by Grouper is probably more reasonable solution for browsing the filtering results than formal concept analysis that CREDO exploits.

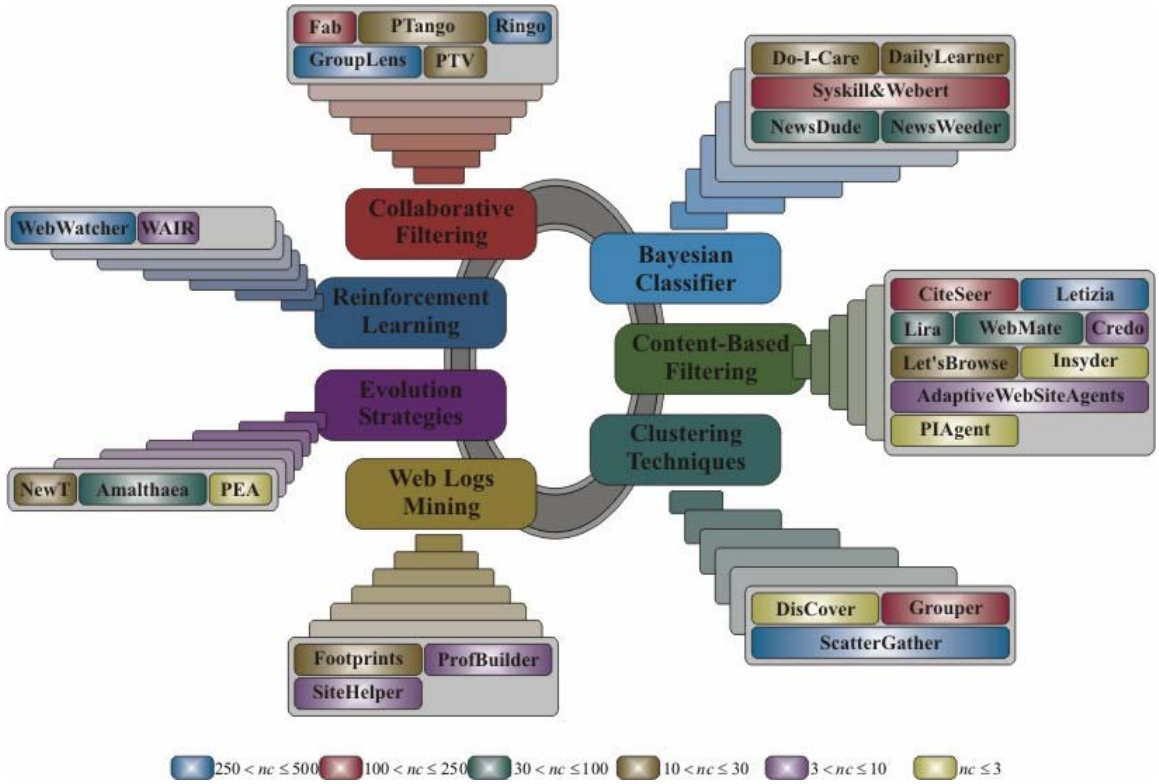

Figure 2.2: The classification of analysed systems based on the mainly deployed strategy

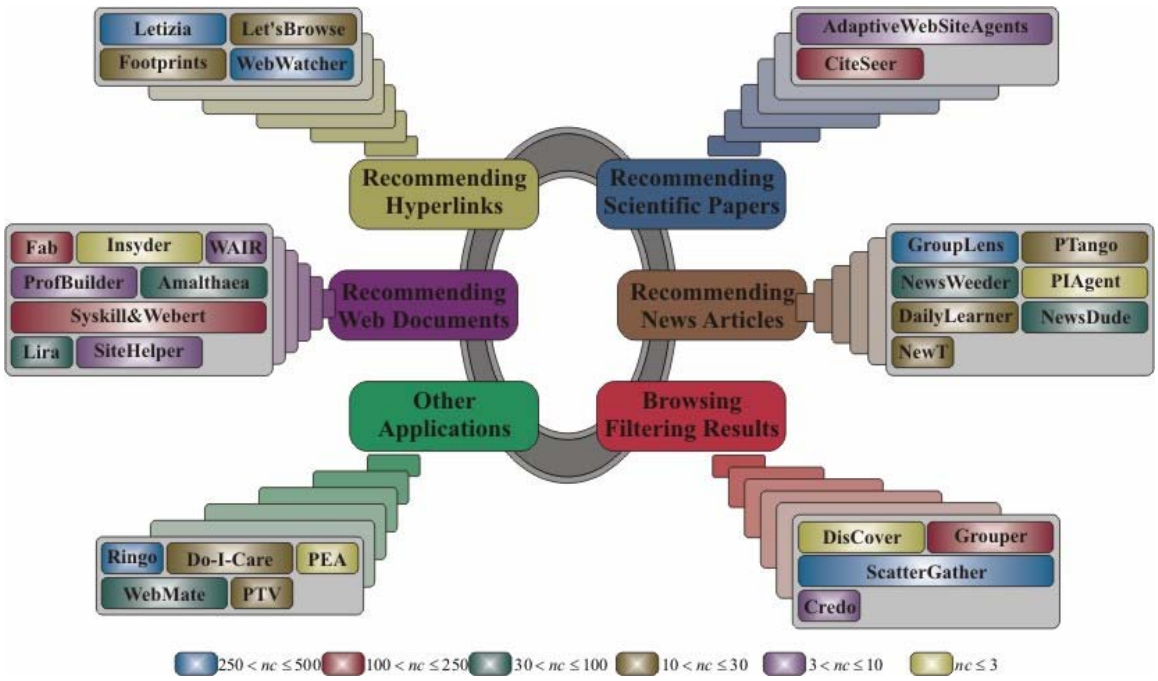

Figure 2.3: The classification of analysed systems based on the intented application purpose

As far as a group of other applications on Figure 2.3 is concered, Ringo is specialised for recommending music albums, Do-I-Care monitors given web sites in order to send notifications when the relevant chances happen, PEA plays the role of a personal email assistant, WebMate is the expert for a query refinement, and PTV generates personal TV program guides. By additionally consulting Figure 2.2, it can be for example noted that only two systems Ringo and PTV, which do not work with pure text documents, actually utilise collaborative filtering, being known as excellent in supporting any type of objects.

The following sub-sections will separately present the main characteristics of every system, being introduced on Figure 2.1. Since the greatest attention will be given on the underlying algorithms, Figure 2.2 will be utilised to guide the structuring of a rest of this section. The motivation behind analysing deployed algorithms is mainly found in trying to identify at the level of every single system both which strategies are used and how they are eventually combined. The possible improvements of the presented systems are going to be carefully analysed, by sometimes making their cross-comparisons, saying that an element from one solution can find its application and improve yet another system.

2.1. Collaborative Filtering

The emerging understanding, saying that good recommendation accuracy alone does not always guarantee effective and satisfying users' experiences [213], becomes a driving force for many filtering strategies which are capable of offering not only accurate but also novel and serendipitous results. One of such strategies is collaborative or social filtering, which aims to recommend the items that are liked by similar users, and which is therefore usually very successful in finding surprisingly interesting results that might not have been otherwise discovered. As supported novelty and serendipity are opposite to high accuracy,

collaborative filtering is commonly combined with other strategies, such as content-based one, which can provide better guaranties towards finding accurate results. The remainder of this sub-section is going to show that with an exception of Ringo, which recommends music albums, every other system somehow improve the quality of results by making the synergy among content-based and collaborative filtering. The Usenet news recommender GroupLens is integrating content-based filtering as filter-bots that augment real users and therefore help overcome both sparsity and early rater problems. Fab aims to recommend web documents by exploiting content-based analysis to first build accurate users' profiles that are afterwards used for finding similar users by collaborative filtering. News articles are selected in P-Tango through benefiting from both filtering strategies by establishing a weighted sum based combination among them. PTV utilises collaborative filtering, being able to easily recommend any type of objects, together with content-based analysis while generating personalised TV program guides.

2.1.1. GroupLens

To help people focus their valuable attention solely on interesting content in the huge stream of available Usenet news articles, a GroupLens system [54][117][132][150][184][213][214][222] uses the collaborative filtering technique, assuming that users having the similar past subjective evaluations are also likely to agree again in future. The selection of articles based on users' judgments has an important advantage, allowing that other dimensions, such as quality, authoritativeness, and respectfulness [213], are used in spite of being hardly analysable by any machine. The inbuilt property of not considering the content of items makes collaborative filtering to be also universally applicable in any domain. By using only the reactions and opinions of users, who have already seen a piece of information, to make predictions for others, a lacking of serendipity, being typical for a content-based filtering whose recommendations have to be always similar to already seen items, is practically overcome [54]. The problems, which are unfortunately brought together with a collaborative filtering, are mainly known as early-rater and sparsity ones. Since the GroupLens system is fully aware of all these problems, the remainder of this presentation will primarily discuss possible solutions with filter-bots [117][222], being nothing else than more or less sophisticated content-based filters. A short notice, about boosting the usability of collaborative filtering by giving the sought explanations, ends this subsection.

Not only that the typical users read merely a tiny fraction of Usenet news articles, but also they prefer not to have to do any extra work to enter needed ratings [150]. There are consequently on average only 0.01% of all available ratings that are possible [184], and rating matrix is sparse. The unavoidable consequence of this sparseness is in an inability to find enough like-minded users, who have already rated items for which predictions are needed. As a result, collaborative filtering system will at best generate weak predictions. The first trial to fight against the sparsity is in separately handling each newsgroup [213], being also preferred by reason of improving the scalability respecting the volume of data. Such an idea of simply using separate collaborative filtering engines for every newsgroup additionally improves performances by disabling recommendations across various topics, being logical as for example agreement in humour does not necessary imply agreement in recipes [150]. These improvements, being certainly necessary for making GroupLens be more scalable, are unfortunately not strong enough, mostly because the ability to generate

solid recommendations still mainly depends on the user readiness to rate articles. As there are many users, who are solely interested in a free ride by consuming evaluation provided by others [214], sparsity problems are also treated through the integration of alternative filtering strategies, which are encapsulated into different filter-bots.

The most important property of filter-bots is their readiness to provide ratings without expecting any recommendation in return. While internally filter-bots use any information filtering technique that somehow generates ratings for articles, a collaborative engine sees them as users, who generously rate everything. The initial filter-bots utilise the extremely simple content-based algorithms, which examine articles regarding their spell correctness, their length, or the amount of repeated text from old messages [222]. The spell checking algorithm simply first computes the percentage of correctly spelled words, and afterwards converts that percentage into rating scale by trying to closely approximate the distribution of human ratings for that newsgroup. The length of an article is transformed by using the hypothesis that readers value brevity, and thus shorter articles receive higher ratings. As many users dislike long messages with little new content, more repeated old texts assume lower rating. The noteworthy property of integrating filter-bots into collaborative filtering engine is that they cannot spoil anything since they will affect only users with whom they correlate well, and for whom they will be good neighbours. Other users that have ratings, which do not agree with filter-bots' ratings, will even not notice their existence because they will not be chosen to be consulted while making predictions. Such a safe integration of filter-bots into collaborative engine together with very promising experimental results, which have been obtained with described simple content-based algorithms [222], have motivated the realisation of more sophisticated filter-bots with learning abilities [117].

The one of learning filter-bots exploits well-known term frequency inverse document frequency scheme to generate a profile based on ratings that are available for a particular user. The procedure is first to extract the most important words from a rated article, and afterwards to integrate them into a profile depending on the provided rating. The obtained profile is subsequently used to produce ratings for any other article simply by computing a cosine measure, which is finally translated into a rating scale. The yet another learning approach induces decision rules based on rated articles that are used as training examples, and where two hundreds words are chosen as the most representative ones. The obtained rules are afterwards used to classify any unrated article, and to generate its rating. These two examples only illustrate that practically any machine learning technique can be used to provide the foundation for filter-bots. The collaborative engines, which typically pre-compute correlations between users on a daily basis to speed up rating prediction [150], unfortunately have difficulties to support efficiently such filter-bots that learn and change their ratings after the new knowledge is acquired. While such sophisticated content-based filters can be easily integrated into a theoretical collaborative engine that searches always on the flight for the best neighbours, the practical constrains of having always quickly to find recommendations have restricted such a combination of various filtering strategies. As filter-bots have to work simultaneously, the expensive computations, which they have to perform, can severely degrade working performances, and force inpatient users to wait even longer for their recommendations. A collaborative combination, where each strategy has to work all the time, cannot be consequently taken into a serious consideration when the real systems are designed.

The added value of the GroupLens system is also in both noticing a necessity, as well as providing solutions, to explain why particular articles are recommended [132]. As the lacking of ratings is very often suppressed by utilising implicit indicators, the first step in explaining is to show to the user its profile, which is used for guiding a search, and which might be quite surprising. A more important explanation is concerned with the goodness of the found like-minded users, who critically define predictions. The recommendations will be weak always when either not so many neighbours are found or their ratings for the article in question deviate a lot. Evaluations with different explanation techniques, which tend to illustrate how good and close are neighbours, have shown that the histogram with grouping performs the best [132]. That explanation only presents how many neighbours have given bad, neutral, and good ratings for a particular article. Additionally presenting how similar found neighbours are to the actual user has not been so successful, probably because a graphical presentation becomes much more difficult and understandable only by experts. Every recommendation engine should help overcome information overload problem, and should not introduce another one by providing huge explanations [132].

Some critics can be sent to GroupLens regarding the exploitation of only reading time as implicit indicator, and only planning to benefit from the other actions, such as printing, saving, forwarding, and posting a follow-up message [150]. These implicit indicators can most successfully work only in their combination, as being already shown in numerous systems, where Letizia [170] is maybe the most illustrative one. When the combination of different filtering strategies is concerned, the integration of various filter-bots has proven that collaborative filtering desperately needs help of other content-based techniques. This strongly encourage the investigation regarding how effective synergies between various strategies can be designed, especially by knowing that GroupLens authors have already published with their colleagues more than 50 publications dominantly about the different applications of collaborative filtering.

2.1.2. Fab

An automatic recommendation service for information retrieval, which is able to over time adapt to its users, who consequently receive increasingly personalised documents, is realised as a multi-agent system that is named Fab [28][29][30]. By maintaining both collection and selection agents, Fab system is one perfect test-bed for trying out different strategies, which either collect documents from the Web that belong to a certain topic, or select some of the collected documents that are suitable for a particular user. The creation of profiles through the content-based analysis, which are afterwards directly compared to find similar users for collaborative recommendations, represents the unique synergy of these two frequently combined filtering techniques. The remainder will mainly discuss about adaptation strategies that are inbuilt into collection and selection agents, as well as about the unique properties of the realised, profile based, combination between content-based and collaborative filtering.

The immense abundance of documents, being available on the Web, has motivated the creation of various collection agents, which employ different heuristics to retrieve pages that will be relevant to most users. Every collection agent has its profile, which consists of important words together with the associated weights, and which is used in guiding the agent in finding relevant information. How exactly a profile is utilised, actually depends

on the type of a collection agent, where search and index ones are most important. Search agents assume that pages have hyperlinks to similar pages, and consequently a best-first search might uncover pertinent information, matching encapsulated profile. Index agents ask various search engines by submitting a disjunctive query, having small number of top words from its profile. Even though both types of collection agents are serving the same purpose of retrieving Web pages, the index agents used to require much less resource, as already got results from search engines are utilised. These different resource requirements of search and index agents are unfortunately not utilised to improve the scalability of the system, and should be taken into account while deciding which agents are better, based not only on the user satisfaction, but also on the actual runtime load.

An adaptation of collection agents seems to be well-known Baldwin effect [33], where each agent evolves, between two successive generations, to speed up evolution process. On the one hand, this short-term learning that speeds up evolution utilises user ratings, which are always forwarded only to a collection agent that has originally retrieved a rated page. Since users, who give negative ratings for pages retrieved by a particular collection agent, are less likely to be shown such pages in the future, and consequently to influence that agent, such an adaptation also encourages the specialisation of every collection agent towards a particular topic. On the other hand, the population of collection agents evolves by periodically killing the worst and duplicating the best agents, where the fitness of an agent is simply defined as the average value of received ratings. These operations have obviously the same intention as well-known mutation and crossover, which also ensure the diversity of a population by applying the principle, where only the fittest will survive. Although both mutation and crossover are usually deployed, new killing and duplication operations are not compared with them to prove that are reasonable to be used instead.

Even though the real Fab system has only used these killing and duplication operations, the splitting of profiles, by applying the graph partitioning techniques, has also suggested as a possibility for utilising the best collection agents [29]. The problem is concerned with somehow splitting a profile, being collection of words, on parts that will contain words, which usually co-occur together in documents. On the one hand, the idea about which words, and in which extent, co-occur together is obtained, by building a so-called co-occurrence matrix based on rated documents, whose elements will practically show how many times every pair of words has been found together. On the other hand, this co-occurrence matrix is equivalent to an adjacency matrix for an undirected weighted graph, and consequently various graph partitioning techniques can be utilised to find the optimal partitions, which have the minimal number of connecting edges. Since the weights, being assigned to edges, illustrate how many times corresponding two words have been found together, created partitions actually correspond to optimal sub-profiles with related words. Although this profile splitting by graph partitioning seems appealing, the experimental evaluation, being unfortunately not given in any known publication about Fab system, is necessary to provide evidences about its usefulness and effectiveness.

Every user has its own selection agent that is responsible for choosing from collected Web pages which ten best match the encapsulated profile, representing current user needs. Since pages are represented as weighted vectors, a comparison with profile is established by computing the usual cosine similarity. Besides stop word elimination, stemming, and term frequency inverse document frequency weighting scheme, a pre-processing includes

also normalisation, which transforms all vectors to a unit length, and thus eliminates the influence of the different lengths of documents. The noteworthy properties are also both the estimation of inverse document frequencies based on around five thousands randomly picked Web pages, as well as avoiding over-fitting by using only one hundred highest-weighted words from any document [28]. Although a realised multi-agent system enables the usage of different types of selection agents, currently only the term frequency inverse document frequency weighting is used together with the cosine similarity measure. The natural extension is therefore not only to integrate different distance functions, but also to try various, more sophisticated, weighting schemes, such as the one based on information gain [205]. Since the quality of results that user is getting critically depends on the built profile, the integration of these various strategies, which will make possible the trade-off between quality and resource requirement, can be a reasonable trial to improve a system.

The user profile, being inside every selection agent, is adapted based on the obtained ratings to better represent current information interests. In order to model that the users' interests change over time, irrespective of the number of recommendations requested per day, each night every weight in a profile is multiplied by 0.97, thus after 23 days, weights that are untouched by ratings would have halved. The purpose of these profiles, stored in selection agents, is also to provide the foundation for finding similar users that are needed to enable collaborative recommendations. Instead of comparing users based on ratings, as pure collaborative filtering does, similarity between profiles is computed by utilising the same cosine metric. While the pure collaborative filtering, based on ratings, can therefore find as similar users only those that have rated exactly the same pages, this novel scheme can make collaborative recommendations as long as similar pages have been rated. This practically means that users are receiving both content-based results based on the direct comparison between their profiles and collected documents, as well as Web pages that are rated highly by users with similar profiles through collaborative filtering. The only possible difficulty of this novel collaborative filtering approach can be concerned with its high dependence on the accuracy of the formed profiles, which ensure that similar users indeed have related interests. This observation can therefore stimulate the integration of various, already mentioned, techniques for word weighting and building accurate profiles.

The noteworthy role, being maintained by selection agents, is concerned with ensuring that neither from the same site nor identical pages will be presented to the user. To escape the presentation of same pages in different sessions, the identical pages are searched for all pages, being seen by the user during the last month. Since many comparisons should be potentially made, one efficient consolidation strategy, based on the short signatures of documents, is thus introduced [29]. These signatures contain only ten most important words with the largest weights from the underlying document, and the used assumption is that only the slight modification in the content will certainly change the corresponding signature. Even though this consolidation solution enables an efficient trade-off between reliability and signature length, no quantitative evidence is given concerning its usability.

The greatest problem, which has seriously limited the scalability of the Fab system, is the existence of so-called central router, which stores pages retrieved by collection agents, and also maintains the inverted list of profiles, kept by selection agents. This inverted list facilitates that the collected documents can be rapidly dispatched to only these selection agents that potentially may show interest. Such a central router is obviously a single point

in a system, and every single collection and selection agent crucially depends on it. While the introduced multi-agent architecture facilitates an easy installation as many collection and selection agents as necessary, they all have to communicate heavily with this central router, which both stores collected pages and propagates ratings. Although such a central router simplifies the communication between collection and selection agents, it severely limits both the scalability and robustness of a system, and the workaround has to be found. Maybe one of the possible solutions can be inspired by Amalthaea [188], whose selection and collection agents communicate directly by making contracts among themselves.

The added values of the Fab system are twofold. On the one hand, multi-agent system is designed, which shows that some activities, such as collecting Web pages and keeping track about user interests, can be naturally mapped to the different types of agents. On the other hand, Fab is a hybrid content-based and collaborative recommendation system that eliminates many shortcomings of pure versions of either technique. Many remarkable solutions, such as consolidation based on short document signatures, and profile splitting through graph partitioning, are additionally introduced. Never the less, this presentation has managed to identify its more or less severe drawbacks, which are mostly concerned with the unutilised potentials of multi-agent technology that enables an easy integration of many various filtering strategies, whose combinations can bring even better results.

2.1.3. P-Tango

To intelligently deliver a personalised newspaper, which contains only the articles of the highest interest that are individually selected everyday for each user, the P-Tango [76] system proposes a framework for combining different filtering strategies. Although the currently combined strategies are only content-based and collaborative ones, a proposed framework is significant, by reason of being easily extendible to any filtering methods. After mainly presenting the strengths and weaknesses of content-based and collaborative filtering strategies, the following paragraphs will be focused on the realised combination of them, which manages to utilise only advantages and omit disadvantages of each one.

A content-based solution in P-Tango is worth of attention by reason of defining, inside every user profile, multiple sections that correspond to various information needs. Every section is personalised to the preferences of each user by having both explicit and implicit keywords. While the explicit keywords are got directly from a user, the implicit ones are obtained by pre-processing the highly rated articles. Even though the used pre-processing of articles performs stop word elimination and stemming, implicit keywords are selected solely based on term frequencies, and essential inverse document frequencies are ignored. Since the selected implicit keywords afterwards critically define the accuracy of content-based filtering, the replacement of the used simple weighing scheme, being based on term frequencies, with more sophisticated solutions, such as the one from a Syskill & Webert [205] system that computes information gain values, seems as a natural improvement.

The content-based filtering can be also criticised by reason of computing the similarity between profile and articles by simply determining the number of words in common. The utilised distance function consequently completely ignores importance values of different words, and its replacement with usually used cosine metric, being typically used in the combination with term frequency inverse document frequency scheme, should be the next step in improving the content-based filtering. In spite of mentioned drawbacks, regarding

both term weighting and distance computation, the realised content-based solution points out that the final relevance score may be first computed on three different ways, which can be afterwards also effectively combined. The first two of them utilise either explicit or implicit keywords, whereas the third one computes the relevance by checking in which section the particular article belongs. The relevance scores, which are obtained by these different content-based strategies, are simply combined in the single score as a weighted sum, having unfortunately fixed weights, each being one third. This weighting scheme is practically the same as the one, deployed in the CiteSeer [162] system for combining the Levenshtein distance on headers, content and citation based similarities. The appealing improvements are hence naturally concerned with utilising the available user ratings for finding the optimal weights, which will boost the performance of this combined content-based strategy.

Although being based on three different strategies, this universally applicable content-based approach happens to be fairly inferior relative to humans when making meaningful decisions about the information content. The P-Tango therefore utilise the collaborative filtering, which combines informative opinions of humans to make both personalised and accurate predictions [76]. The similarities between users are first computed by using the typically applied Pearson correlation coefficient, and then utilised to deduce the relevance for the novel articles. The collaborative filtering is consequently ineffective alone when users have not rated a given novel article, for new users who do not have enough ratings, or for users who do not generally benefit from the opinions of others. The combination is going to be successful only when tending to exploit strengths of content-based filtering, which include early predictions that cover all articles and users, while gaining benefits of accurate collaborative filtering predictions, as the number of users and ratings increases.

The available strategies are combined in order to form a final set of results, which will optimally mix content-based and collaborative recommendations for each user. While the combination is again realised as the weighted sum as in the case of mixing pure content-based strategies, the weights are not anymore fixed. They are adjusted separately for each user in order to minimise the absolute past error between predicted and real ratings for content-based and collaborative recommendations. The used learning rule will generally reward any strategy, which delivers recommendations that users like, and consequently such a strategy will have greater influence when future result sets are formed. Since the optimal weights, which result in the most accurate predictions, are obtained based on user ratings received for particular articles, such weights are personalised both on per-user and per-article level.

The adaptation unfortunately highly depends on the number of received ratings, which are solely responsible for finding optimal weights. In the case where much more than two strategies are combined, any good strategy, which unfairly has low assigned weight, will probably have no influence when result sets are formed. Since users cannot rate articles that are not recommended, such a strategy will consequently be without any chances to directly improve its qualifications. The possible workaround can be in extending the used adaptation rule, by simply asking the selected strategies to pay something like a fee for being able to send back their results to the user. The basic motivation for the explained modification is certainly concerned with the speeding up of the learning process, which the similar fee-payment technique has achieved in evolution-based systems.

There are many opportunities for improving the introduced framework for combining filtering strategies, such as not solely using only explicit user ratings, being unfortunately got only for 0.5% of delivered articles [76], but also taking care about implicit indicators of interest. Even though many things can be additionally done, the P-Tango system has already demonstrated that the realised simple weighted combination of content-based and collaborative filtering has managed to provide better results that any single used strategy. The extensibility and scalability of the realised combination scheme are obviously issues, which have not got enough attention in P-Tango system, and which have to be seriously taken into consideration while analysing the plug-in of other filtering strategies.

2.1.4. Ringo

Since content-based filtering techniques find recommendations based on correlations between the content of objects and users' preferences, numerous complications might be observed while tending to apply them in domains, such as the musical one, where content cannot be so easily parsed by machines, users have the aspirations towards serendipitous findings, and elements of styles play the important role. These and similar requirements, which are hardly realisable by applying the pure content-based strategies, are probably a cornerstone motivation for using collaborative or social filtering techniques in developing Ringo system [232][233] that makes the personal recommendations for music albums and artists. While four different collaborative filtering techniques are designed by changing a distance function, which is utilised for computing similarities between users, even more strategies can be obtained by varying underlying configuration parameters. Although the Ringo system points out that the optimal combination of these collaborative strategies can probably yield the best performances, this potential unfortunately remains unutilised, and will be discussed in the following paragraphs, after briefly introducing these techniques.

A basic collaborative idea is to maintain for every user a profile with provided ratings, which will be afterwards used to determine which users have similar tastes. Since profiles can be compared on multiple ways, four collaborative filtering techniques are defined. The first one compares profiles by using the mean squared difference, which is computed for every two artists that two users have both rated. The second and third strategy utilise either standard or constrained Pearson correlation coefficient. The standard coefficient will increase a correlation by some amount always when two users have rated a particular artist with a score that is above the average score that they separately give. By increasing a correlation always when two users have rated the same artist with a score that is above four, being the absolute average rating score in Ringo, the constrained Pearson coefficient is obtained. The forth strategy generates recommendations by utilising the correlations between artists, which are established based on given various ratings by different users. Even though the utilised distance function is actually the constrained Pearson correlation coefficient, this strategy can compare artists on any of the previously mentioned ways.

Each and every introduced collaborative filtering strategy further heavily depends on many tuning parameters, which have the critical influence on quality of recommendations. On the one hand, similar users can be selected as those, whose similarity with the actual user is above a certain threshold. Since the reported experimental results [233] shows that wrongly set threshold will result that either many weakly or no similar users can be found, many highly different strategies can be defined simply by varying this parameter. On the

other hand, every strategy tries to determine the level of a confidence into each prediction, being important since recommendations, which Ringo generates, are sometimes used as guidelines for making purchases. Only the some of possible techniques for measuring the reliability of a particular prediction are such as insisting on a commonality between users that have to have at least minimal number of ratings in common, checking the number of found similar users, computing the variance of ratings that the chosen similar users have given to the artist in question, and so on. These algorithms can be freely combined with any of the introduced distance functions, and consequently can successfully produce even greater diversity in the collection of available collaborative filtering techniques.

The motivation to combine these different collaborative filtering strategies mostly lies in the experimental results, which are reported in [232][233]. Even though in most of the cases the constrained Pearson correlation has shown the best behaviour, sometimes other solutions can be more accurate, but together with offering lower coverage. The resource requirements are also extremely different, where the strategy, which uses pre-computed correlations between artists, happens to be the cheapest one. Since the real-time usage has always to assume continuously changing load of needed system resources, the availability of strategies, having different execution costs, can improve performances. The resource aware combination of strategies can either select the cheap strategy, which utilise the pre-computed correlations, or exploit more expensive collaborative techniques, depending on the estimated load of system resources. Since no two users are the fully the same, system that combines different collaborative filtering techniques can learn, based on available ratings, which strategy works the best not only in a given runtime situation, but also for which users. Maybe results reported in Adaptive Web Site Agents [202], where each user prefers a particular strategy for recommending scientific papers, are also valid when various collaborative filtering techniques are applied in a music domain.

By reason of being almost the first real application, which uses collaborative or social filtering, many things, such as the synergy with content-based techniques, are ignored. In spite of that Ringo system has managed to achieve high popularity that is measured in thousands of active users. Although the functionality, such as receiving the personalised reviews that users with similar tastes have written, has without any doubt contributed to the successfulness of Ringo system, its added value is in introducing a novel technology, which happens to be able to autonomously and automatically consider thousands of other users for recommending only the best of available items. The possible combination with other strategies that do not require critical mass of users to be successful can only further increase the usability of collaborative filtering.

2.1.5. PTV

As it becomes almost impossible for people to cope with hundreds of TV channels and thousands of daily TV programs, one personalised television listing system, named PTV [78][238][239][240][241][242] is developed. Due to its nice ability to automatically learn about both implicit and explicit preferences of individual users, PTV is able to generate a personalised daily TV guide, which is specially compiled to suit the particular viewing needs, and which could be finally delivered on multiple end devices [78][240]. Since the quality of generated TV guides critically depends on the ability of deployed both content-based and collaborative filtering techniques to find really relevant TV programs, the

remainder of this discussion will focus on the critical presentation of deployed strategies and their combinations.

A content-based filtering aims to recommend TV programs, which are similar to those that are liked in the past, and which are found through the comparison to the user profile that encapsulates different viewing preferences, such as preferred viewing time, available channels, favourite genres, and so on. Since TV programs, being automatically collected by schedule agents, each mining the particular online resource [241], are represented by features that do not appear in users' profiles, these two different representations have to be transformed into the same space to enable their comparison. TV programs and profiles are afterwards compared as a weighted sum of similarities among corresponding features. Such a content-based filtering unfortunately requires great knowledge-engineering efforts to convert profiles and TV programs into the same world of features, as well as to define necessary similarity metrics, which are for example able to compare different genres. The one of possible improvements of the utilised measure for obtaining the overall similarity is naturally concerned with adapting the weights for every single user rather than globally setting them. It is reasonable to assume that each feature has a different importance for every user, and a system should for example learn that the right genre is more significant for a particular person than the viewing time. These weights can be adapted based on the received user feedback values, where weights of features, being more responsible for a final similarity score, are adjusted in a greater extent. Useful ideas can be also borrowed from the adaptation algorithm that Adaptive Web Site Agent [202] exploits to learn about which filtering strategy is preferred by a given user.

While the success of content-based filtering relies on the ability to accurately represent both TV programs and users' preferences through a suitable set of features, collaborative techniques exploit the correlation between users in terms of their ratings that are assigned to various TV programs. Since collaborative filtering treats every TV program as a black box, engineering efforts for obtaining accurate content representations do not exist. The recommended TV programs are those that are preferred by the like-minded users, and not those that are similar to the actual profile, and therefore a diversity problem is overcome. The provided rating vectors, corresponding to different users, are compared by using the simple graded difference metric, being actually Manhattan distance [127], which assigns default value always when a rating is missing. Computing similarities solely based on the ratings has a drawback that only users, who have rated exactly the same TV programs can be potentially found as being similar [30]. As each user is also described through the set of feature values, which are used for content-based filtering, the additional collaborative technique can find similar users based on this content part of a profile, as being suggested in Fab system [30]. The obtained added value can be especially useful in the TV domain, since users do not need anymore to rate exactly the same programs in order to be found as neighbours. The interests in the same genres of comedies can be for example sufficient for couple of users to be assessed as being the similar ones.

The last part of a puzzle is concerned with the realised combination between content-based and collaborative filtering techniques, known to complement each other perfectly [238]. Since collaborative filtering can recommend only items that are already rated, both new and special single-episode TV programs, such as concerts, have to be solely handled by the content-based approach [240]. The importance of the right combination between

these strategies is probably the consequence that different publications about PTV system present various synergies. As the collaborative filtering requires the substantial number of users to be able to generate usable TV program guides, the initial releases have combined these two strategies similarly as NewsDude [40] and DailyLearner [42] combine nearest neighbour algorithm and naïve Bayesian classifier, while modelling short-term and long-term user interests. Simply the collaborative filtering is always asked first, and only in the case where the bad similar neighbourhood is created, the content-based technique will be responsible for recommending TV programs [239]. By reason of having more than 20 thousands registered users [240] by spring 2001, collaborative recommendations are no more problematic, and consequently the latest PTV releases more ambitiously mix these strategies [240]. Some of recommended TV programs are content-based, and others are collaborative, being a solution for taking into account unrated programs and ensuring the diversity of results.

Even though the successfulness of PTV greatly depends on the way how the content-based and collaborative recommendations are mixed, no known publication about PTV is uncovering a mystery behind the realised synergy, most probably because of commercial reasons. Both popularity and usability of PTV system are at the same time only proving that one applicable deployment typically requires synergies between ranges of different strategies in order to be able to cope with very real problems. A hybrid PTV approach has managed to effectively produce personalised, very real, TV guides, containing programs that a target user has enjoyed in the past, and programs that other similar users have liked.

2.2. Reinforcement Learning

Learning from numerous trial and error interactions with an environment, in order to perform the best possible actions by receiving delayed rewards, makes the foundation of an unsupervised technique, known as reinforcement learning [248]. As it requires neither the exemplary supervisions nor complete models of an environment, two systems, named WebWatcher and WAIR, have found the necessary motivation for trying to utilise the reinforcement learning on solving difficult information filtering tasks. On the one hand, WebWatcher applies reinforcement learning for figuring out which hyperlinks are most relevant for the actual information needs of the current visitor of a web site. On the other hand, WAIR aims in utilising reinforcement learning to optimally balance exploration and exploitation while deciding which search engine results are worth to be returned back to the inpatient user. Such applications of reinforcement learning unfortunately have their shortcomings, which can be usually overcome by the integration of alternative strategies, as it will be shown in the following subsections.

2.2.1. WebWatcher

Users may find it difficult both to create the appropriate queries and to locate the information of interest in the case of having no specific knowledge about the content of the underlying document collection. On the one hand, systems, like Scatter/Gather [80], aim to deploy efficient clustering algorithms, which will dynamically produce the table of contents, needed to facilitate users' browsing activities. The cornerstone idea is to help a user first obtain the overview concerning the available content, and afterwards accurately

somehow express its information needs. On the other hand, WebWatcher [27][253][254] acts as a tour guide that provides the assistance, which is similar to the guidance of a human in a real museum. It accompanies users from page to page, suggests appropriate hyperlinks, and learns from the obtained experience to improve its advice giving skills. The used WebWatcher learning algorithms, together with the combination of them which brings superior performances, are going to be the main topic in the following discussions.

The simplest approach for easily recommending the hyperlinks to a user is to count the frequency with which each link on a page has been followed in the past. This approach is constantly suggesting the links with the highest frequency, or in other words the most popular hyperlinks are always selected as recommendations. The underlying logic is the same as in the Footprints [265] system. It creates a directed graph, whose nodes are pages, whose edges represent the hyperlinks, and where the weights aggregate how many times the corresponding link has been followed in the past. Since this simple recommendation strategy only takes into account how many times every link has been selected by any user, its obvious drawback lies however in the complete ignorance of actual user preferences. No matter what a user is searching for, this strategy will provide the guidance towards the most frequently visited pages, which unfortunately cannot satisfy any information need.

The straightforward way to take into consideration the actual users' information needs, being represented as the collection of relevant words, is to match these words with the underlying anchor text, which is used to describe the hyperlink. The matching procedure is based first on term frequency inverse document frequency scheme, which transforms both anchor texts and users' preferences into feature vectors where more descriptive words have a higher importance, and then on computing a cosine distance [127] between the formed vectors. Even though this approach takes care of the user preferences while deciding which links are promising to be followed, the links can be better represented when being modelled not only through the associated anchor texts, but also through the actual sentences and the corresponding headings where the hyperlink is located [27]. The unavoidable drawback of this strategy is in its static nature where everything depends on the way how the corresponding page is created and the system cannot learn from the experience with previous users.

The strategy for recommending noteworthy links, which is able to learn from the past experience, represents every hyperlink not only through the text that is associated to the link by the creator of the page, but also through the preferences of users, who have followed a particular link and at the same time have satisfied their information needs. By augmenting the description of a hyperlink with the keywords, which are typed at the begging of the tour, whenever the user follows that hyperlink on a successful search for the relevant information, this strategy is able for example to learn that a term such as "machine learning" matches a hyperlink such as "neural networks", although these phrases share no words in common. Even though the last property sounds as the great advantage of this strategy over the others ones, the correct associations among words can be learnt only in the case where users are persistent in their browsing activities towards the needed information. In reality users unfortunately tend to have a fairly short attention, and for example in spite of stating their initial interests in some technical subject such as "algorithmic complexity", once they notice that there is an online cake machine, their browsing path goes in a completely new direction [253].

The last proposed method for selecting links aims to exploit the hypertext structure that exist among different pages, with the underlying reasoning that in many cases only a sequence of pages and the knowledge about how they relate to each other can best lead to a users' satisfaction. A well-known reinforcement learning technique is used for finding the optimal browsing paths, which will maximise the sum of received rewards, being proportional to the degree to which a particular Web page fits the users' interests. As the suitability of a given page for a single word, the deployed learning algorithm will actually utilise the corresponding term frequency inverse document frequency value of that word. The outcome of a reinforcement learning algorithm, being deployed for each single word and every page, is the so-called Q value that is assigned to every link on any known page. The selection strategy for user interests, which are expressed as a single preference word, is to track the hyperlinks having the largest assigned Q values for that word. In the case where users' interests have multiple words, the resulting Q value, whose highest values should be finally followed by a user, is simply computed as the sum of Q values, which correspond to the given users' preference words.

Even though the reinforcement learning sounds very appealing for finding the optimal browsing paths, it is efficient enough only in the case of the relatively static collection of pages. In the case of highly dynamic changes in the hyperlink structure, the application of reinforcement learning becomes unfeasible, by reason of having to perform the very time consuming learning of optimal paths every time any change occurs. Additionally, there is the great diversity in the interests of users, whose preferences are often expressed with unlearnt words. This finally seriously limits the applicability of an approach, being based on reinforcement learning.

By reason of observing the strengths and weaknesses in every presented approach, WebWatcher proposes the application of the hybrid strategy, which should provide the superior performances. At the same way as the classificator accuracy can be increased by using multiple classificators and deploying so-called majority voting reasoning technique [127], WebWatcher counts how many times a particular hyperlink is suggested by every available method, and recommends the hyperlinks selected by most of methods. In the case where a particular link is faulty thought to be the important by one method, other methods will not make the same mistake, and the majority vote will not recommend that hyperlink. Experimental results have shown that this multi strategy approach outperforms every single method, which means that an appropriate combination of available strategies can even bring improvements in the quality domain.

The other noteworthy functionalities of the WebWatcher system are concerned with an ability to find similar pages only based on a hyperlink structure, and with a capability to notify a user via email in a case where a page of interest has been changed. An algorithm for finding the similar pages works under the assumption that two pages are of the similar interest if some third page points to them both [254]. The underlying problem is actually almost identical to the nearest neighbour search in the collaborative filtering, where two users are similar when they like the same items. A matrix where each and every page has corresponding row and column is created, and the value 1 in (i, j) means that page i has the link to the page j, and if such link does not exist the assigned value to (i, j) is 0. In order to find the similar pages to a given page, the pages with the similar columns should be found. In doing that the well-known collaborative filtering problems, such as sparsity,

grey ship, banana problem and so on, also take place, and they might seriously limit the application. In the case where for example there are not enough Web pages with links to a candidate page, the bad similar neighbourhood of pages will be created, and not similar enough pages can be found.

Ten years after the first version of the WebWatcher system was realised, it becomes obvious that WebWatcher did not mange to handle the huge expansion of World Wide Web. Its learning is based on the assumption that pages, being inside its competence, do not change, which is unfortunately far away from being truth in the nowadays highly dynamic Internet world. Always when a particular page changes, WebWatcher has to forget everything that it has learnt in previous tours. Some of the used link selection algorithms, such as reinforcement learning, have such complexity, which additionally restrict the possible application of the WebWatcher system only to small sub-domains, having not so many associated pages.

In spite of these drawbacks, which have probably disabled the successful commercial deployments, WebWatcher is important as the research system that combines different strategies. It has shown that the realisation of the same functionality, being in its case the selection of promising hyperlinks to be followed, through multiple strategies, can bring improvements in the overall achieved quality. Even though the way how these strategies can be combined is only marginally addressed in [253], it encourages a research towards the more scalable combination of available strategies, which will additionally take care of the available resource and user preferences.

2.2.2. WAIR

Seeking the state of a user profile, which best represents actual information interests and therefore maximise the expected value of the cumulative user relevance feedback, is formulated in the WAIR multi-agent system [60][278][279] as the reinforcement learning problem. The insufficiency of explicit user ratings is overcome by using the classification approach based on the neural network, which exploits the different implicit indicators of interests in order to estimate the real relevance feedback values. The remainder of these discussions will first analyse the potential problems of each these two approaches, and afterwards will point out to the main reasons of their successful synergy.

The reinforcement learning treats information filtering as an interactive process where got rewards are observed for the actions performed in the different states of a user profile. While rewards directly correspond to the relevance feedback, actions are decisions which documents should be offered as recommendations. The fundamental goal is to selectively retain those actions that are most effective, and which will maximise rewards, obtained in the long run from a user. The initial set of candidate documents, which will be taken into account to be sent back to a user, is obtained through a usage of different search engines. A user profile, being the actual collection of relevant words with their associated weights, is used to generate queries, which are sent to these search engines. To balance exploration and exploitation, words to be included in a query are chosen by using ε greedy selection, which either selects the highest weighted words or randomly picks any profile word with $1-\varepsilon$ or ε probabilities, respectively. Documents found by search engines are afterwards pre-processed, where both stop word elimination and stemming are used to generate word vectors. They are finally normalised to speed up subsequent comparisons to a user profile

by computing cosine distance measure. The obvious drawback is concerned with ignoring other documents, while deciding about the importance of a particular word, whose weight encapsulates only the local frequency information. Even though Internet is a dynamically changing environment [60], important document frequencies can be estimated through the creation of one static sub-collection, as being done in the Fab system [30]. The known term frequency inverse document frequency scheme, being known to give the best results in most situations, can be consequently made applicable for modelling.

To maximise the amount of rewards that will be received over time, the correct actions, which select relevant documents, should be performed in every state of a profile. On the one hand, the immediate reward depends on the quality of documents that are currently delivered. This reward can be maximised by exploiting what is currently known, and by selecting only the highest ranked documents. On the other hand, the relevance of initial documents, which are found by search engines, correlates directly with the accuracy of a profile in representing user information needs. Not accurate enough profile will result in bad queries that are posted to search engines, and consequently even the highest ranked documents will be of a pure quality. A system obviously has to explore user interests, and to recommend also the low ranked documents from which the most can be learnt. The exploitation-exploration policy is again ε greedy selection method, which gives chances to every single document to be recommended. Although performing random actions with the ε probability is usually dangerous in reinforcement learning, WAIR actually always selects documents that are already retrieved by search engines. They serve as a pre-filter that eliminates really unsafe recommendations, and therefore greatly reduce chances that a really bad document appears. The difficulty that is unfortunately discussed in no known WAIR publication is concerned with the efficiency of reinforcement learning, which has to successfully handle not only practically the infinite number of profile states, but also the unlimited number of actions. The additional reinforcement problems are related with the instability of rewards, as it will be discussed in the following paragraph.

As providing the explicit ratings is a tedious process, where users might be unwilling to participate, WAIR exploits the implicit indicators of interests, like reading time, book-marking, scrolling, and following hyperlinks. The measured factors, which characterise a user's behaviour on filtered documents, afterwards represent inputs for three-layer neural network that is trained to be able to estimate relevance. The training is initially performed separately for every user, who is asked to provide the explicit feedback. Consequently, it makes possible to discriminate well the personalised importance of the different, always available, implicit indicators. Even though this neural network based approach for always generating a relevance feedback manages to provide thousands of needed rewards, the known inconsistency of user explicit ratings can either completely disable or at least slow down the training of a neural network. The difficulty is finally found in the huge number of needed explicit ratings, which a typical user is unfortunately unwilling to provide.

The added value of a WAIR system is certainly in the obtained experimental evidence about the reliability of every implicit indicator. On the one hand, reading time and book-marking seem to be good indicators of interests, where the latter happens to reflect users' interests most strongly [60][279]. On the other hand, following hyperlinks and scrolling are not so usable for deducing the relevance of the read document. These observations are consistent with a conclusion from Letizia [170], saying that no single indicator is reliable

enough, and that their proper combination tends to outperform any separate solution. The WAIR system tries to successfully realise this combination by using the presented neural network approach, which at least sounds appealing from the combination point of view. Although the ability of reinforcement learning approach to adapt to changing information interests during the longer period of time is maybe problematic, using such an approach together with both combining multiple search engines and taking note of various implicit indicators, represents almost the unique solution for the information retrieval, and thus is worth of an attention.

2.3. Evolution Strategies

Adapting to the constant changes of users' interests together with exploring potentially relevant information areas is achieved in evolution strategies through the application of both recombination and selection operators. While mutation and crossover recombination operators are responsible to efficiently explore a search space, selection operator exploits already learnt knowledge, encapsulated into a population [36]. The known exploration – exploitation trade-off is thus simply controlled by changing a value of selection pressure, being a degree to which better individuals are favoured [183]. The successful application of evolution strategies in the area of information filtering is therefore quite logical, where Amalthaea, NewT, and PEA are only some of the numerous systems, which try to benefit while recommending web documents, news articles, and e-mail messages, respectively. Although all these systems typically represent members of a population as a collection of words with sometimes associated importance, the design of recombination and selection operators is something that mainly makes them different from each other. The following sub-sections will consequently try to uncover these subtle differences, which exist in the deployed evolution techniques inside Amalthaea, NewT, and PEA systems.

2.3.1. Amalthaea

Since information discovery and filtering are proven to be especially suitable domains for applying appealing multi-agent technology, a personalised system, named Amalthaea, which proactively tries to discover from various distributed sources the information that is relevant to a user, has been introduced in [188]. The multi-agent technology is applied by maintaining two different types of agents, being information filtering and information discovery ones. The ways how these agents are managing to learn the user's interests and habits, to maintain their competence by adapting to the changes in the user's information needs, and to finally explore new domains that may be of interest to the user, are going to be critically presented in the following paragraphs.

The high level architecture of the Amalthaea system is presented on Figure 2.4. The meaning of the illustrated operations is as follows. Operation 1 retrieves from the user its preferences, which are necessary for the creation of profiles that are encapsulated into the separate filtering agents. It is the only operation that occurs only at the begging when the initial agents are created, as well as when the user wants to manually configure additional filtering agents. Operation 2 represents the posting of information requests from filtering to discovery agents. As soon as information discovery agents have obtained the request, they will query the distributed information sources, being actually known search engines,

through the application of Operation 3. Operation 4 returns the results that are found by the search engines to information discovery agents, which further send the corresponding downloaded original documents to the filtering agents through Operation 5. The selected documents are in the form of a digest returned to a user by Operation 6, and a user finally is providing the feedback through Operation 7. The obtained relevance feedback, being necessary for the evolution of the system, is from filtering agents to responsible discovery agents forwarded by Operation 8.

The information filtering agent keeps a user's profile that is represented as a collection of words together with associated weights. Such a representation of a profile is necessary by reason of making possible its comparison with the term frequency inverse document frequency models of documents. While these document models are built by performing both stemming and stop word elimination, few noteworthy additional activities should be mentioned. The one is extraction of links, which are treated as a special type of keywords. The other is the multiplication of the computed weight of a word by the so-called header constant. It is therefore ensured that the position of a word in a given document has the influence on the computation of its weight. For example, it seems to be logical to assign the higher weights to words that appear in a title than to words from a body text.

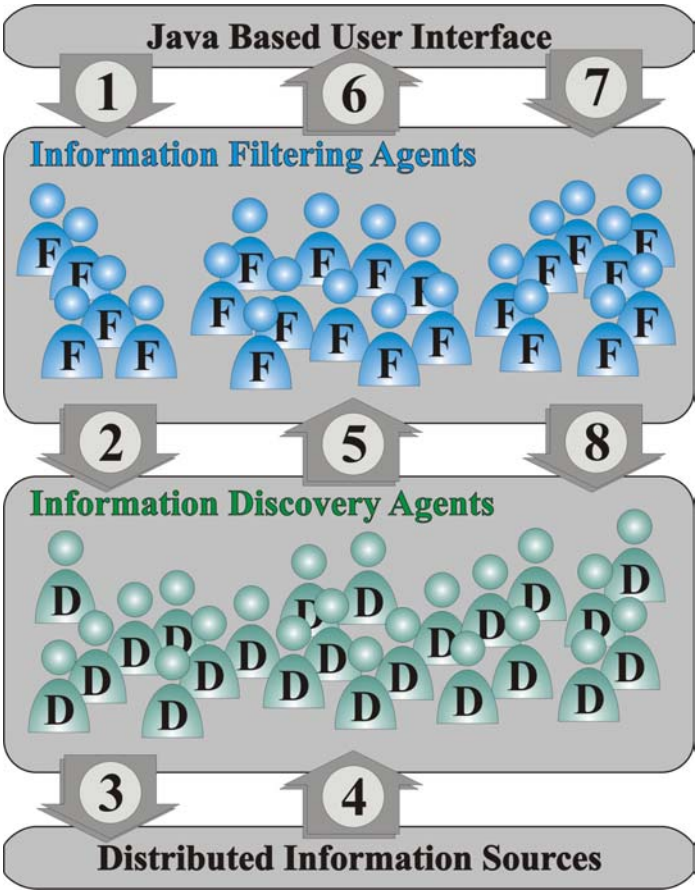

Figure 2.4: The architecture of Amalthaea system

By performing the comparison between its profile and candidate documents, filtering agent acts as a mask, which allows only to a document that is the closest to its profile to

pass through. The result of this comparison is also the so-called confidence that illustrates how the particular filtering agent is confident or sure that a user would like the suggested document. In practice, this confidence is nothing else than a known cosine distance [127], computed between a profile and the analysed document. A system decides, based both on the found confidence and the fitness of the responsible filtering agent, whether or not the particular document will be included into the digest that is sent to a user. In other words, the filtering agents, which are known as the non-competent ones by reason of not finding really relevant documents in the past, will have the low assigned fitness values. Therefore, they will have the weak chances that the documents, being found by them, will be at the end recommended to a user.

The adaptation to changes in the user's interest is achieved through the usage of the explicit user relevance feedback values, which express a real relevance of recommended documents. A system relates this feedback to the filtering agent that has proposed that document in order to either increase or decrease its fitness value. On the one side, in the case where both the used feedback is positive and the filtering agent is confident that the user would like the document, its fitness value will be increase proportionally to the stated confidence. On the other side, the agent will pay penalties through the reduction of its fitness value in a case where it has been confident but a negative feedback is received. To accelerate the destruction of the filtering agents whose documents are never sent to a user, the fitness values linearly decay. Since their fitness values will not be changed by reason of getting neither positive nor negative feedback, the introduced decay happens to be the only way of removing them. The diversity of the population of the filtering agents is finally stimulated by introducing the penalties for similar agents that propose the same documents.

The cornerstone mechanisms, being important not only for the adaptation but also for the exploration of a search space, are the well known evolution operators being known as crossover and mutation [181]. The evolution of agents is controlled by their individual fitness, as well as the overall fitness of the system. On the one side, the individual fitness defines the chances that a particular filtering agent will either produce an offspring or will be destroyed. While a larger individual fitness increases the chances of a particular agent to become a parent, a smaller fitness implies a greater probability that a given agent will be removed from a population. On the other side, the evolution rate, which defines how many children will be born, is defined by the overall fitness of a system. The simple rule is to increase the evolution rate, in order to speed up the adaptation to the new interests, always when an overall fitness is diminishing. Such a situation happens always when the user is providing many negative feedback values by reason of not being satisfied with the relevance of results.

A user can tailor the Amalthaea system on several ways. It is first possible to manually accelerate the evolution of a system always when having the feeling that the information interests have been changed a lot. Furthermore, a number of both filtering and discovery agents is under the direct control of a user. Finally, a user can introduce its own filtering agents by either providing the passage of the relevant text or giving the URL of a liked document. The provided document will be afterwards pre-processed in order to generate its weighted representation, which will be finally used as the initial profile of the created filtering agent. More importantly, these manually created agents are also called long-term

agents because they cannot be destroyed easily even in a case where they do not perform as well as other agents.

It is essential to explain how the crossover and mutation mechanisms are applied on the filtering agents that are represented as the weighted vectors of words. As a crossover the well known two point crossover operator [181] is used. It first randomly selects two points in the keyword vectors and afterwards exchanges the words of the two parents that lie between these points. Consequently, two new weigthed vectors and two new agents are created. A mutation is attained by both randomly modifying the weights of mutated keywords, and adding randomly selected keywords from another agent. The underlying hope is that, such a crossover and mutation operations will manage to successfully refresh and specialise the agent population. The performed experiments [188] have unfortunately shown that hundreds of generations are needed for a creation of a solid agent population. This is probably the greatest drawback of the Amalthaea system, since it is assumed that users have to provide feedback for the creation of each generation. It seems quite logical that many users will give up of rating tons of articles before starting obtaining acceptable recommendations.

The advantage of the Amalthaea system most probably lies in relying on the search engines, as well as in utilising the results of their queries as the good starting points. The modern search engines are indexing billions of Web pages, and the idea of first querying them to find initial document set, which will be then further analysed by downloading the original documents, seems to be appealing. In the Amalthaea system, the search engines are used through the so-called information discovery agents that are assigned to them. It is specific to each and every discovery agent not only which underlying search engine is used, but also how many keywords are sent when querying and how these keywords are combined. To additionally support a monitoring functionality, separate discovery agents are created for Web pages that should be monitored. They will inform the filtering agents with whom they collaborate when the new content is available on the monitored page. The informed filtering agents will afterwards check the new content, and decide whether or not it is really relevant to a user.

The mentioned cold start problem, which unfortunately requires too many user ratings, might be partially overcome by initially trying to create the population of filtering agents with profiles that are relevant to the user's interests. Instead of creating random profiles, a user is simply asked to give the list of URLs that correspond to the relevant documents, which are then pre-processed in order to create the meaningful initial filtering agents. The yet another way is to automatically obtain the list of liked URLs by examining the user's favourite list and its browsing history. In spite of these efforts, the Amalthaea system will still need many expensive generations for figuring out which agents are really good. This finally puts the doubt into the effective usage of evolution strategies for the information filtering tasks that are time critical.

From the filtering point of view, even though Amalthaea has hundreds of different filtering agents, all these agents use the same filtering strategy, which is based on the simple cosine similarity measure. The only difference among these filtering agents is that each one encapsulates different user profile. The possible step further in improving this multi-agent population can be in first introducing different strategies as the extra filtering agents. More importantly, these strategies might be combined by both taking care of the

separately determined confidence for a given result, as well as the fitness of a responsible agent.

By reason of depending on external search engines, from where the results are first retrieved in order to point to the original documents that are then downloaded, Amalthaea might have the serious problems with the response time. The used query driven approach [127] obviously has influenced a decision that the results are presented only in the form of digests, without even thinking about the usage in the real time. The fast response time can be achieved by the application of the update driven scheme [127], where the updates are always locally stored, and where the requested documents are quickly available. This interesting update driven improvement has been unfortunately out of the scope of the Amalthaea system.

The application of evolution strategies finally imposes the existence of not so small number of agents. Because all these agents have to belong only to one user, Amalthaea can have the serious scalability problems to present a successful deployment. A possible workaround can be found in the separation of information filtering and discovery agents, in the way that only filtering agents inhabit users' computers. They will be configured to send their request to the discovery agents that run on online server [188]. In spite of the more or less severe drawbacks, being mentioned in last paragraphs, it is fair to conclude that, by creating several processing levels between the actual information sources and a user, Amalthaea gives the excellent example of increasing flexibility in utilising the other novel filtering strategies.

2.3.2. NewT

The ability to both specialise to user interests, adapt to preference changes and explore newer information domains makes the foundation of NewT [235], being one personalised multi-agent filtering system for news articles. As user information interests are modelled as the population of competing profiles, the used learning mechanisms are both relevance feedback, as well as crossover and mutation genetic operators. While these recombination genetic operators are mainly responsible for adaptation and exploration issues by creating more fitted future populations, in the meantime, every user profile also learns through the application of relevance feedback techniques. Taken together these learning mechanisms make the so-called Baldwin effect, saying that a population evolves towards a fitter form much faster, whenever its members are allowed to learn during their lifetime. A focus in the following paragraphs will be therefore set on this combination, which leads towards a multi-agent system that is specialised, adaptive and exploratory, and that is a holy grail of personalised information filtering [235].

User information interests are represented by associated multiple profiles, each having various fields, such as newsgroups, location, authors, and preference keywords. The same metadata are extracted from semi-structured news articles, which contain more than just a text [235]. While newsgroup, location and author metadata are got from the article header, well-known term frequency inverse document frequency scheme is used for selecting the most significant words from each pre-processed document. As every field from a profile has additionally its unique importance for a user, the final relevance is therefore obtained as the weighted sum of single cosine distance values, computed separately for each field. The found relevance together with the fitness, which corresponds to a responsible profile,

finally determines whether a given document is going to be offered to a user or not. More fitted profiles have a permission to give more documents, whereas the underlying fitness has also influence on the found score, and thus changes the ordering of recommendations. The scores for documents, found by not very fitted profiles, will therefore pay penalties, which will consequently place them not on the top of the result list. Since fitness directly defines the number of results that will be offered, the profiles with low fitness will maybe not even get a chance to present documents, and consequently will have weak chances to get any feedback, and improve themselves. This drawback can be overcome by installing proportional selection, saying simply that the fitness of a profile defines its probability to offer articles. While more fitted profiles will be selected more often, such a selection will also give chances even to the worst one to sometimes provide results.

The straightforward way for a user to provide the feedback is to rate the recommended article as either relevant or not. Such a user feedback will adapt all fields of a profile that is responsible for finding a given article, as well as profile fitness will be moved to better represent its successfulness. There is an interesting opportunity to provide also a selective feedback, saying for example that the user likes the responsible author, which will adapt only the corresponding field of a profile. To make less severe the problem where profiles that do not offer articles cannot learn by reason of not receiving any feedback, users have the opportunity to provide examples of liked documents, being recommended by nobody. These examples will either adapt a profile that retrieves articles from the same newsgroup, or new one will be created whenever there is no such profile. The learning by examples obviously happens to be too weak in porting a feedback towards low fitted profiles, even under the assumption that users are very motivated to provide many liked articles. The additional drawback of NewT is also in the only long-term planning to utilise the implicit feedback techniques, which are quite common especially for the modern systems, whose learning heavily depends on the amount of provided ratings.

Although mutation and crossover operators are claimed to be primarily responsible for the ability of a system to both adapt to the changing user interests and explore potentially interesting areas, their application is at least quite restricted. On the one hand, mutation is contained in the application of small random changes that should support the exploration activities. These changes are unfortunately limited only on newsgroup field, whose value is changed by randomly replacing one newsgroup with one of its nearest neighbours. As the nearest neighbours are pre-computed by finding cosine distance values between most representative words of every available newsgroup, such a mutation will only result in the serendipitous discovery with the same words in similar newsgroups. On the other hand, a used two point crossover operates by exchanging fields between selected profiles. This crossover also does not internally change fields, which disable an exploration of the novel areas of interests, being easily achievable by collaborative filtering techniques.

In spite of the mentioned drawbacks, NewT offers a solid framework for personalised information filtering, having genetic mechanisms in its basis. In a current implementation, the only used filtering strategy is a content-based technique, which simply matches words from documents and a profile. A very natural way of extending the system in a direction of combining multiple strategies is contained in independently using for every profile a separate filtering technique. The assigned fitness will consequently represent not only the accuracy of a profile, but also the ability of the utilised filtering strategy to find relevant

documents. The integration of multiple filtering strategies will therefore together with the already noted trials to automatically adjust the evolution [235], which can be for example speeded up in the case where the user is providing many bad ratings, represent the most promising improvements of NewT.

2.3.3. PEA

Personal Email Assistant (PEA) [269] filters incoming mails and ranks them according to their relevance in order to help nowadays users, who easily end up with the large part of their working days being spent with reading the numerious emails. PEA maintains the personal user model that may consist of several profiles and uses evolutionary algorithms to move them constantly closer to the current information needs. By doing that PEA aims at assisting users in dealing more effectively with the daily load of their emails so that the valuable working time is saved for more productive and creative tasks. The remainder of this sub-section will be focused on presenting the approach that PEA uses for overcoming the problem of modelling the users, who have usually great difficulties in specifying their interests that additionally can be easily and unpredictably changed in the course of time.

The users' inability, to precisely describe their information needs by directly giving the exact keywords and phrases, is addressed through the automatic extraction of relevant words from the representative pool of documents. These documents are selected by a user as the ones that best describe the particular domain of its interest. By providing the right pools of documents for each of the interest domains, a user will be modelled by separate profiles, which are mapped to its various information needs. The construction of the high-quality profiles is ensured by requesting that the size of their pools should be larger than a certain threshold, being currently fifty documents [269].

In order to find the most descriptive words for a particular pool, after the documents are transformed into a sequential word list, the well known stop word elimination and stemming activities are performed. From the pre-processing point of view, the added value of the PEA system is in handling two common types of spelling errors, being the insertion and deletion of one wrong character. These spelling errors are especially very common in unofficial documents such as emails, and they are tolerated by performing a string comparison that allows one missing or one additional character in the two strings. The weights of the words are computed by applying the term frequency inverse document frequency scheme, where the term frequencies are computed for a given pool as a whole, and inverse document frequencies are determined respective the all other pools.

As soon as the degree of selectivity or a weight is determined for different words, the rank list of words can be formed. More importantly, a user can freely edit it by modifying the ranking, deleting words, adding additional words, defining synonyms, and specifying translations in order to support the cross-language filtering. These corrections, which are done by a user, are improving the representation of an interest profile, and thus ensuring the better behaviour of a system. The final output of the specialisation to users' interests can be seen as the list of chosen words, being ranked based on their relative importance in describing the corresponding information need.

The evolution is enabled by creating the initial population of so-called chromosomes for each interest domain on the basis of the random variation of the corresponding ranked

list of words. These random variations are based only on changing the importance of the words, which seriously limits the evolution capabilities especially when being compared to a solution from the Amalthaea [188] system that assumes also insertion of words from other profiles. A user interacts by correcting the faulty predicted relevance ratings, which can be irrelevant, interesting, relevant, important and very important, for new incoming messages. Therefore, the important explicit feedback is obtained, and the fitness values of a particular chromosome can be computed. The population of chromosomes evolves by applying the biological principle of a natural selection and a survival of the fittest. While parents are selected through a roulette wheel selection [181], the population of offsprings is created by applying both crossover and mutation operations on the chosen parents. As the crossover operator, the single-point crossover is used, which randomly selects one cut point, and exchanges the importance of words from two parents relative to that point. To take into account that an offspring may be of less fitness than its parents, the probability of a crossover is introduced to define if the crossover will be performed or the parents will stay unchanged. A mutation is contained in altering the importance of words by the small random amount, where similarly a probability of a mutation defines whether or not a mutation will occur. While the value used for probability of crossover is set to 0.8 since this probability is a serious restriction to the advancement of a population, the probability of mutation is 0.05 in order to prevent the degeneration.

The robustness of the PEA approach is contained in treating misspelled words in the document that should be compared to existing profiles. That is achieved through defining the similarity measure between two words by taking into account the number of divergent characters. While the similarity equals 1 if the two words are identical, it equals 0 if they have no characters in common. A standard string comparison is used only for very short words because the performance of a defined measure, being tolerable regarding spelling errors, decreases for the smaller lengths of words. A document is pre-processed in a way that every its word is compared as explained to words from profiles, and always when the actual word is close enough, it is declared that the corresponding profile word is found. All the found profile words are at the end forming the document vector, which is finally compared to each and every profile by applying the similarity coefficient of Dice [127]. This measure sums the weights of words which are identical for a document vector and a particular profile. Consequently, if there are no components in common it equals 0, and if all components are identical, its value is 1.

The existence of multiple chromosomes that all correspond to a particular information interest is exploited in PEA by performing the explained comparison between a candidate document and every single chromosome. In other words, each chromosome in the current population for an interest profile scores a document, and the final score of the particular document for a given interest domain is found by averaging all scores that are produced by chromosomes, belonging to that domain. A drawback of simply averaging scores lies in ignoring the corresponding fitness values of the responsible chromosomes. A solution from the Amalthaea system [188] that utilises as weights the corresponding fitness values seems to be better that the simple average sum in PEA.

The PEA system can deal with the stronger shifts of interests by inserting new words into chromosomes, which corresponds to the particular information need that has greatly changed. Alternatively, the words that no longer reflect the focus of user's interest can be

removed. Unfortunately, the presentation of PEA system in [269] lacks the explanation how these words can be found, probably by reason of assuming that they are explicitly provided by a user. This can be treated as the drawback of the PEA system, which can be extended to utilise the relevance feedback values for finding these new words from rated documents.

By reason of being the application for actually handling emails, PEA has well-known pre-filters based on header fields. The empirical tests have indicated than the one third of incoming emails is deleted without reading due to the header information. The efficient header analysis therefore aims at reducing the number of emails whose content should be analysed. Consequently, it makes applicable presented sophisticated information filtering techniques, which take care of the misspelt words, and which therefore cannot be applied on the large amount of data in the short time.

Besides necessary duplicate elimination, the PEA system is able to generate templates from the examples, being explicitly provided by a user. These templates are then used to extract the specific types of the information from the particular document by effectively ignoring the non-relevant portions of texts. Unfortunately, the application of such learnt templates is very restrictive, mostly due to requesting many examples for generating the general enough templates.

It is finally worth of mentioning that PEA monitors the user behaviour regarding the reaction to incoming emails, such as deleting, forwarding, saving, replying, and printing. The set of classifiers is defined, where each one is responsible for predicting actions for the particular types of mails. In the case where the prediction of the responsible classifier is the same as the action performed by a user, the fitness of that classifier increases, and reverse. This fitness is important for selecting, again through a roulette wheel selection, which classifier is going to be applied for an email that can be classified by more than one classifier. These classifiers will first result in suggestions to a user and later, after gaining the confidence, in automatically performing actions.

The PEA system represents a successful trial to address the problem of adjusting user models in response to the shift of interest through the synergy of evolutionary algorithms and sophisticated content based filtering strategies. Most importantly, provided filtering techniques are able to work even on a document with spelling errors, being common for emails. It seems appealing to see how the presented system will scale in the case of being applied on the much larger domain, such as World Wide Web, where tricks for reducing the amount of data to be examined based on header analysis can be hardly applicable.

2.4. Web Logs Mining

Modern nowadays web servers typically log the access patterns that encapsulate data about which page, when, and by whom was visited. Even though the idea of mining these logs to for example extract the information about the visitors' preferences sounds very appealing, the immense abundance of logged data makes many sophisticated data mining algorithms completely useless [127]. In spite of these difficulties, being concerned with the necessity to handle tens of gigabytes of raw data, several systems exploit web server logs either to simply visualise the most frequently visited paths or to somehow figure out

which pages and other users might be potentially interesting to the particular visitor. As it is going to be shown in the rest of this sub-section, the visualisation of paths is how Footprint tries to facilitate the browsing tasks, the collaborative filtering based on logged access patterns is something that definitely makes ProfBuilder to be unique, and the building of a model of users' information needs from web pages where a lot of time has been spent distinguishes SiteHelper from other systems.

2.4.1. Footprints

Providing the browsing assistance to a user by making available the past experience of many other humans, who have already browsed the same Web site and tried to solve the similar problems, is the basic motivation for the realisation of the Footprints system [263][264][265][266]. It first pre-processes information from Web server logs to see where users have gone, and then displays the mined, so-called interaction history, data into the various displays in order to provide the requested guidance to novice users. On the one hand, the added value of the Footprint system is in practically demonstrating that the visualisation of data, being found by mining Web server logs, through maps, trails and annotations, can significantly increase the efficiency of browsing. On the other hand, the Web server logs usually do not have data about the underlying information need, which has been responsible for initiating a particular browsing activity. The Footprints system is consequently not capable to make any distinction among different browsing sessions, and therefore it is forced to always simply aggregate all the available log data. The following paragraphs will both briefly present these various visualisation techniques and discuss about the underlying advantages and disadvantages of the Footprints system.

Since Web server logs typically contain data about the sequence of visited pages, the natural visualisation, is to construct a directed graph, whose nodes represent pages and whose edges illustrate the frequency of moving between the connected pages. By reason of illustrating the possible ways, which are used for moving between pages, this graph is named map because of the analogy with maps in the real world. The colour coding is used both for representing the popularity of particular pages and the number of times every hyperlink is followed. While, the first release of Footprints system has utilised Boltzmann algorithm for balancing positive and negative forces in order to create a graph layout where connected nodes are reasonably close together [266], the last version of a system uses hyperbolic geometry, which is especially good for the presentation of highly non-planar and disconnected graphs [263]. Such directed graph accumulates all available browsing paths, and consequently is without any doubt very useful for Web designers to help identify pages, which discourage further browsing and from where no link has been followed. Not making any distinction among the reasons for browsing is unfortunately reducing the usability of these graphs for the end users. As being already noticed in the WebWatcher [253] system, not taking care of the user preferences while recommending hyperlinks, and simply always suggesting the most often chosen links, will only guide users towards the most popular pages. But, frequently visited pages are not wizards, and they unfortunately cannot satisfy all needs. This drawback has been noted by the authors' of the Footprints system, and they plan both to cluster available paths in order to present to a user only the relevant set of paths [266], and to integrate the so-called purpose tool [265]. Therefore, users will have a necessary mean to specify the underlying reasons for a particular browsing activity.

The graph representation of the interaction history seems to be quite impractical when the additional data about the page, such as its title, should be displayed. Users often want to see the title of a page [265], and therefore the additional display, named trail [264] has been realised. The idea is to separate the sequences of interactions, and present them in the form of a tree, having nodes being labelled with the corresponding titles of pages. By reason of also not taking care of the users' preferences, the bias towards the most popular pages can be still detected. While both maps and trails keep the page unchanged and display the interaction data in the separate window, the annotations insert next to the every hyperlink the percent value, showing the amount of users, who have followed that link. The problem of the insertion of these additional texts into a page lies in risking of destroying the alignment or spacing on which the information display may depend [266]. The WebWatcher system [253] unfortunately has exactly the same problem by reason of highlighting the links that are worth of being followed by inserting the additional icons around.

The Footprints system is similar to WebWatcher [253] and Letizia [170] systems by reason of trying to help its users while browsing. While both WebWatcher and Letizia perform the content based analysis of the underlying Web pages and either explicitly or implicitly take care of the driving information needs, Footprints only utilises Web server logs and completely ignores both the content of pages and the users' preferences. In spite of that the developed visualisation means have proven their usefulness [265], and they have additionally opened the opportunities for the application of advanced graph analysis algorithms. These techniques can efficiently mine the experience of the large number of humans, who have already tried to solve the same problems in the information systems. The developed visualisations are finally very important because they can provide natural extensions for many other browsing assistants. Even though other assistants can be more sophisticated when the underlying filtering strategies are taken into account, they might lack the necessary good representation of the found recommendations.

2.4.2. ProfBuilder

The mining of access patterns makes the foundation of the ProfBuilder system [261], which inhabits a particular site, and has the assigned goal of being online responsive to the information needs of users. Both content-based and collaborative filtering techniques, are used to separately recommend relevant, and only locally, accessible Web pages. Also, ProfBuilder shows that these strategies can benefit from the knowledge about the visiting paths of various users. The remainder of this sub-section will show how available access patterns might be utilised by both content-based techniques for implicitly deducing the interestingness of a given page, as well as collaborative filtering for finding similar users.

The one of major challenges for content-based filtering is the efficient adaptation of a user profile to always accurately represent current information interests. A usual explicit communication with a user might be impractical by reason of either being very hard to manually find effective keywords that describe the interest or having to provide reliable ratings that reflect the relevance of a page. The ProfBuilder is aware of these difficulties, and consequently exploits only the implicit indicator of following a hyperlink, which is strengthen by the knowledge about the access patterns of other users. The used heuristic for implicitly deducing the interestingness of a selected page, based on selected link and

known access patterns, can be described as follows. In the case where the visiting path of a current user can be frequently found in the database of old access patterns, the last page in that path probably has the similar content as other pages by reason of being tracked by many users. Such a page cannot be consequently used to learn a lot about users' interests, and the level of a profile adaptation should not be large, which is the same as treating the given page as being not so interesting. The opposite situation is occurring when not so many visitors have followed the particular sequence of pages, which means that the last selected page is usually quite typical for the actual user. It is therefore logical to assume its high relevance, and to significantly adapt profile based on keywords that are extracted from it. The explained heuristic can be summarised by stating that the most can be learnt about a user when its browsing behaviour differs from the behaviour of others, or in other words when not following the most frequently chosen hyperlinks.

According to the point-of-view of content-based filtering, a noteworthy property is the pre-processing of pages, where solely headlines and anchor texts are taken into account. After stop words are eliminated and stemming is performed, the importance of selected stems is computed by using standard term frequency inverse document frequency scheme. The user profile, being also the collection of preference words together with associated importance, is compared with such vector representation of pages by computing a scalar product. The pages with the highest values of a scalar product will be presented to a user as the most relevant pages when content-based filtering recommendations are requested. Although the used assumption about taking only headlines and anchor texts simplifies the filtering process, no experimental evaluation about the computation savings regarding the potential lost in precision is given in [261].

Various existing collaborative filtering approaches usually differ only concerning the algorithms, which are used for comparing users and finding those with the similar tastes. Although users are usually compared regarding either provided ratings or profile words, the ProfBuilder system is unique by finding similar users based on their visiting patterns. The access path of an actual user is simply compared with those that correspond to past users, and the most similar ones are found in order to deduce which hyperlinks are most often followed. Collaborative filtering will simply recommend these pages that have been most frequently selected by the found community of similar users.

Although the illustrated usage of access patterns for both maintaining accurate profiles and finding similar users seems to be appealing, the computation and storage costs have influenced that only paths of the length one can be taken into consideration [261]. On the one hand, content-based filtering will deduce the interestingness of a particular page only based on the number of users, who have gone there from a one given previous page. As various reasons can be behind selecting the particular link on a page, taking care of only one previous page is certainly not enough. On the other hand, collaborative approach will simply search for the most frequently selected links on the current page and recommend them. Even though the performances of such collaborative filtering are not evaluated in [261], WebWatcher [253] defines four different strategies for recommending links. The least successful happens to be the one, which recommends solely based on the popularity of links, and which is unfortunately exactly the collaborative approach from ProfBuilder [261].

The proposed approach to learn user preferences, based on probabilities that the given page should have been visited, and that are obtained by analysing the access patterns of past users, seems to be improvable by including additional implicit indicators. Simply following a link, which almost nobody before has followed, cannot be reliable indicator of interests. According to authors of Letizia system [170], each implicit indicator is weak by itself and only their combination makes them powerful. The ProfBuilder is therefore important by introducing this additional implicit indicator, being based on access patterns of other users, and being pluggable in any system that exploits implicit feedback. Even though the practical considerations have currently limited the pattern size to only one, the optimisation of their storage can enable the usage of longer paths. That will consequently improve both introduced filtering strategies, and maybe made them usable in frameworks that tend to provide optimal solutions by effectively combining the available strategies.

2.4.3. SiteHelper

To assist users, who visit a given Web site, the SiteHelper system [237] is intended to act as a helper for finding relevant information, being stored on a particular Web server. By mining the available server logs, SiteHelper learns about users' information interests, and afterwards utilises the acquired knowledge to recommend relevant, locally available, pages, as well as to send the useful notification when interesting changes occur. Since the modelling of users' preferences by using so-called heuristic coverage HCV rules seems to be unique in a domain of the information retrieval, the following paragraphs will shortly present how the mining of Web logs together with the manually generated hierarchies of words can lead towards the efficient generation of these rules.

The Web server log is utilised to always find relevant pages, being the ones on which a particular user has paid a lot of attention, as well as to ignore everything where little or no time has been spent. While such relevant pages are always available, in the case where a user is willing to explicitly input few keywords, which describe its information interests, additional relevant pages can be found as those that match provided keywords, and that are finally approved as being really relevant. All these relevant pages represent positive training examples, whereas the negative examples are obtained as pages that are found by the mentioned keyword searching, but that are subsequently disapproved by a user. These positive and negative examples are afterwards utilised to generate HCV rules, where only words from a dictionary are taken into account. The used dictionary incorporates, in the form of a hierarchy, only terms that are important for the local site, and that are manually selected to dramatically reduce the number of distinct words, being worth of an attention. Since every word plays the role of a separate variable in the algorithm for the generation of HCV rules, such a simplification is vitally important for more compactly representing the available positive and negative training examples. Thus, the used dictionary finally makes the process of generating these rules to be more efficient.

The essential functionality of modelling users' preferences is established by exploiting these HCV rules, which are both utilised to recommend candidate pages that will become new positive and negative training examples after receiving a user feedback, as well as to detect relevant changes that require push activities. As these rules actually specify, which Boolean expressions words should satisfy to declare that a particular page is relevant or not, they are very similar to those that can be obtained from a decision tree. Experiences

from Syskill & Webert [43][205], NewsDude [40], DailyLearner [42], and many other systems, have unfortunately pointed out that the decision trees are not so suitable for text classification tasks by reason of basing their decisions only on the presence or absence of single words. Due to the underlying similarity between HCV rules and decision trees, the same performance problems can be potentially noticed in the SiteHelper system. It can be consequently nicely extended by other machine learning algorithms, whose combination will tend to offer a superior behaviour. The yet another problem can arise as only visiting time is concerned to be the reliable implicit indicator of relevance. Several systems, such as Letizia [170] and WAIR [60], have already shown that any single implicit indicator is weak, and that their combination is necessary. The Web server log usually does contain additional data, which can for example provide the insight about the number of followed hyperlinks. Such and many other implicit indicators are unfortunately not exploited in the SiteHelper system.

Together with these drawbacks, many additional aspects can be further improved. The one is providing the more diverse results by integrating collaborative filtering techniques, which will exploit Web logs as in the ProfBuilder [261] system. In spite of all mentioned possible improvements, the SiteHelper system has already offered a solid starting point. It mines Web server logs and generates HCV rules. Therefore, it increases the accessibility of the information, being very critical on large Web sites that urgently need housekeeping facilities.

2.5. Bayesian Classifier

The treatment of information filtering tasks by the naive Bayesian classifier belongs to the much broader domain of a text categorisation, which unfortunately knows to be quite challengeable for real-word problems mostly by reason of both being high dimensional, as well as often having the skewed category distribution over labelled documents [277]. A problem becomes even harder since information filtering additionally assumes that the users' preferences are not fixed, and that a system has to be able to keep adapting even after being trained for the long time. On the one hand, this so-called plasticity-stability dilemma can be the real nightmare for neural network inspired approaches that have to decrease learning or adaptation rate in order not to destroy the already learnt knowledge [251]. On the other hand, the naïve Bayesian classifier simply estimates the probability of a category for a particular document based on the probabilities of words for that category. As probabilities of words are got based on all available already categorised documents, the naïve Bayesian classifier hopefully does not have a plasticity-stability problem. The reminder of this sub-section shows that the naïve Bayesian classifier is powerful enough to be solely used in Syskill & Webert to recommend web documents, and in Do-I-Care to decide whether a particular change on the monitored web page is relevant or not for a given user. By reason of not being able to quickly learn and to keep track about already recommended documents, both NewsDude and DailyLearner additionally use the nearest neighbour algorithm for accomplishing these short term learning tasks. The minimum description length, being similar and often confused with the naïve Bayesian classifier, has also been used in NewsWeeder to filter news articles, therefore definitely proving the high popularity of these methods for the text categorisation tasks.

2.5.1. Syskill & Webert

The idea of having a software agent, which will help a user locate only useful and relevant documents in the vast abundance of the information on World Wide Web, is the driving force for a development of the Syskill & Webert system [43][44][177][204][205]. The system is designed to assist a user in satisfying its long-term information goals [177], being achieved by learning a separate profile for each topic of the user's interests, and afterwards using this profile either to suggest which other links might lead to interesting pages or to construct a query for a search engine to retrieve really relevant hits [205]. On the one hand, the ability to examine the hyperlinks on a particular page is making Syskill & Webert to be a local reconnaissance agent, similar to WebWatcher [253] and Letizia [170] systems. The underlying assumption of performing scouting is again in enhancing the usefulness of a particular Web page with many hyperlinks by installing an assistant. It is watching the user actions while browsing this resource, learning its interests, and also recommending which unvisited links should be explored next [177]. The added value is not only in annotating the interesting hyperlinks as WebWatcher does, but also in giving a numerical value representing a certainty that a user will really like the underlying Web page. On the other hand, the usage of search engines, which index the significant part of Web, dramatically increases the chances that the relevant documents will be found even though they are not in the local neighbourhood of a particular page. By using the learnt profile, Syskill & Webert is able to construct a query for a search engine, to examine the obtained results, and to recommend only really relevant ones. The noteworthy properties, being the application of different algorithms both for learning and revising the profile of a user, are going definitely to represent the cornerstone of the discussions in the following paragraphs.

To increase the efficiency of the creation of a profile for a particular topic, instead of only observing the behaviour of a user as Letizia agent does, a user is explicitly asked to rate pages, and provide the set of liked or disliked documents. These documents are pre-processed in order to discover words, having a good discriminating power by reason of occurring frequently in liked documents, but infrequently in disliked, or vice versa. The used measure is information gain [127], which generally shows how well classes, being created based on the different values of the examined feature, correspond to classes that really exist in the analysed collection of objects. Syskill & Webert uses a representation based on Boolean feature vector, which assumes that features are actually words, and that every feature can take only two distinct values, corresponding either to the presence or the absence of a given word from a particular document. The number of selected words, which represent different dimensions in the vector models of documents, should be large enough to provide the accurate enough representations. By experimentally comparing the usability of the learnt user profiles, depending on the number of chosen words, it has been found that having less than 50 or more than 250 features, reduces the accuracy of predicted recommendations. Selecting too many features is probably not good since these features correspond to the noise, and consequently are not good for the identification of relevant documents. The optimal ratio, between the accuracy of recommendations and the complexity of learning for most domains, yields to a usage of 128 distinct features [204]. Both stop words and found formatting tags are automatically excluded from being further selected as the most informative features, whereas the presented system realises do not perform stemming, which is certainly one of the possible improvements.

After the available rated documents are represented as Boolean vectors that store the information about the presence or the absence of the selected most discriminating words, different algorithms can be used for predicting how promising the unseen document will be. The great contribution of the Syskill & Webert system is in investigating a variety of the machine learning algorithms including naïve Bayesian classifier, nearest neighbour algorithm, ID3, perceptrons, and multi-layer back-propagation neural networks [205]. On the one hand, the performed experiments have clearly shown that, not only ID3, but also more advanced decision tree learner C4.5, is not well suited to the task of classifying text documents, probably by reason of attempting to build trees that test as few features as possible when making a classification [204]. On the other hand, no one algorithm is superior over the several tested domains, where either naïve Bayesian classifier, nearest neighbour, or back-propagation neural network performs best. The authors' decision is to use the naïve Bayesian classifier as the default algorithm in Syskill & Webert because of being very fast for both learning, which linearly depends on the number of examples, and predicting, which is independent of the number of training cases [44]. The natural way to improve the quality of recommendations is to combine these multiple algorithms, for example by using both bagging and boosting techniques [127], which will simply ask for every prediction multiple classifiers, and additionally maybe assign different weights to them to illustrate the various levels of confidence. A yet another opportunity, to improve the scalability of a system, is to exploit the resource awareness. This is manageable since these algorithms have very different complexities, where nearest neighbour actually does not perform learning, but has the prediction that depends on the number of examples, and neural networks require expensive and long lasting learning, but have the fast prediction. The idea is to simply escape the application of expensive strategies always when a system is highly loaded.

Most of systems, which depend on the availability of users' ratings, have the problem of keeping users, who will probably lose confidence before a system has learnt enough to be capable to generate the useful recommendations. In the case where the naïve Bayesian classifier is used in Syskill & Webert, the achieved accuracies are hardy even better than random guesses when less than 10 ratings are given [44]. The reasonably good accuracy of around 80% is reported in the case where a user has provided at least 35 ratings [177], being for many users an unacceptably long waiting. As the accuracy essentially depends on words that are extracted from rated documents, the natural workaround is to ask a user to describe its information preferences through exact words. The expected effect is the reduction of the number of irrelevant words, which make the learning task much easier, and probably leads to higher classification accuracy. At the begging, there are only few rated documents, and a system should rely more on the profile given by a user than on the knowledge learnt from the examples. As more training data becomes available, a system should gradually increase a belief in its own hypothesis and gradually decrease a belief in an initial user profile. This reasoning is for the naïve Bayesian classifier realised by using the conjugate priors rule for updating profiles, where the weight of the users' estimates is equivalent to observing 50 examples [44].

By using the words that are directly provided by a user, three different strategies can be defined for selecting most informative features. Two extremes are strategies that either utilise only words obtained from a user as classification features, or ignore these users' words and select all features by computing information gain values. The middle solution

is to utilise words provided by a user to replace the same number of words that have the lowest information gain [44]. Obtained results again encourage the combination of these various strategies for selecting words to be indexed, since they behave very differently in various domains. While features selected by a user do not seem to be good discriminators for the pages returned by the queried search engine, they discriminate very well between interesting and uninteresting pages for example in biomedical domain [44]. A simple idea of combining these strategies might exploit the properties of the underlying domains, for example by deciding that always when not so discriminate features can be automatically extracted, a priority should be given to the users' ones. Maybe this domain awareness can even improve the accuracy during runtime by learning which strategy is good for which domains.

The Syskill & Webert system is also important by reason of presenting the novel extension of naïve Bayesian classifier that is especially suitable for information filtering tasks [43]. This extension is concerned with handling probabilities for single features, which have always to be not zero by reason of being all multiplied to obtain the joint probability. In the case where both the number of training examples is even smaller that the number of features and a user is usually providing only liked documents, many single probabilities can be zero. Two known strategies for introducing the needed bias from zero are the Laplace correction, which always increases the number of matches for one and the overall number of examples for two, and the ε approach, which replaces the probability estimates that are zero with the small constant ε. The different properties of these two strategies, where Laplace correction has much stranger bias, are exploited by realising the hybrid approach which combines them. The combination is realised on a feature level, where for not discriminative features, having low information gain values, the Laplace correction is used by reason of having stranger bias. In the case where features tend to discriminate well between classes, the probability estimates should be kept weakly biased, and consequently ε approach is definitely the right correction strategy. The performed experiments have shown that while in various domains different strategies perform best, the explained hybrid approach is always a clear winner [43].

The importance of making the selection of as good features as possible has besides motivated the usage of lexical knowledge, which can hopefully eliminate words that are not good to be considered as features [204]. A used lexical database is WordNet, having around 30.000 commonly occurring English words. When there is no relationship between a particular word and the other words from a topic, being determined by using WordNet, that word will simply not be selected. Experiments have shown a substantial increase in accuracy when using lexical information [204] especially when there are not many training examples. This can be fundamental workaround for users, who might be willing neither to rate many pages before a system can give the reliable predictions nor to provide the initial knowledge in the form of relevant words.

Although being not related with combining different strategies, it is really worth of mentioning how the Syskill & Webert system solves the problem of creating queries that are submitted to search engines. Since search engines cannot accept queries having many words, it is usually not possible to send all the selected informative features. The Syskill & Webert system therefore simply always sends only seven words that have the largest information gain and seven words that are the most frequent in liked documents when

stop words are not taken into account. This solution is fundamentally much simpler than the application of graph partitioning techniques, being suggested as a possible solution for splitting the large sets of words in the Fab system [29]. As soon as a search engine has returned results, they are treated simply as links, and Syskill & Webert can annotate them by using the trained classifier, which gives predictions that a user will like the underlying page.

The advantage of representing different users' interests through multiple profiles is also experimentally proven in Syskill & Webert [177]. In the case where all interests are mixed into one single profile, neither multi-layer neural networks nor naïve Bayesian classifier does reach an accuracy level, which differs substantially from simply guessing that a user would like a particular page [177]. The found explanation is that words, which are informative in one domain, are irrelevant in another one, and that probably learning such complex profile requires unreasonably many training examples. In the domain of information filtering, where users are obviously unwilling to rate hundreds and thousands of documents, restricting the scope, by learning a separate profile for every topic of interest, happens to be essential for simplifying the task of successfully learning user's preferences.

The importance of monitoring a particular Web source is also introduced as the one of possible extensions of a system. The goal is to simply notify a user when a link that leads to the interesting page is added to a page being monitored [204]. The idea from the Let's Browse system [171] of exchanging users' profiles in order to get the effect of browsing through the eyes of an expert is also mentioned [177]. The discussion about Syskill & Webert can be concluded by stating that its authors mentioned almost ten years ago many ideas that are still alive, and which have made the foundation for many other specialised intelligent computer systems that primarily try to adapt to the user's information needs in a personalised way. The noticed solutions, for improving the Syskill & Webert system as far as a combination of various strategies is concerned, maybe have a necessary potential to improve both the accuracy of found recommendations and the exploitation of available system resources.

2.5.2. Do-I-Care

As a result of applying the naïve Bayesian classifier, being the same as the one used in the Syskill & Webert system [205], for helping a user find only relevant changes to Web sites that are monitored, the Do-I-Care agent [178][246] was invented. Since persistently revising the known information sources, until the right information is not available, is an onerous task, the idea is to provide the automatic means for monitoring previously found pages to detect when they change, as well as deciding which noticed changes are worth enough of being reported to a user. As it will be pointed out in the remainder of this sub-section, in spite of potential drawbacks, the application of the naïve Bayesian classifier seems to be appealing for learning which detected changes are interesting for a particular user.

The cycle of the autonomous actions starts with periodically visiting a user-defined set of target pages in order to identify any changes since a particular page was retrieved last time. While the changes are detected simply by comparing the previously known version of a page with the current one, the naïve Bayesian classifier, being separately trained for

each interest topic of every user, is used for deciding which changes are interesting. The worth enough changes, are explained with found informative words, and afterwards sent to a user both through the specially designed web interface and also simply as an e-mail message. The used classifier is finally learning by receiving a user relevance feedback on the interestingness of reported changes, which consequently become training examples.

The challenge of finding informative words, which will be used for the representation of found changes, is addressed by computing the mutual information, being also known as information gain in data mining community [127]. Although the accuracy of the naïve Bayesian classifier highly depends from the distinctive power of the chosen words, the application of both stemming and lexical knowledge bases is left for the future versions of a system [246]. While the customisation of selected words is possible by giving the list of stop words, experimental observations from the Syskill & Webert system [204] lead to the hypothesis that users can be usually very good in identifying really informative words for modelling changes. The presented Do-I-Care agent is unfortunately lacking from any experimental evaluation of the achieved classification accuracy, as well as the number of chosen words. The future experiments should check whether the observations obtained by examining the Syskill & Webert system are still valid in the domain of monitoring Web pages.

Since some types of changes are obviously uninteresting, but common enough to be a nuisance at best or overwhelming at worst, Do-I-Care agent defines not only the classes of interesting and uninteresting, but also so-called never-show training examples. While changes, which are predicted to belong to the never shown class, will be never presented, a user has the possibility to identify a particular change as never-show. Not only that this change will never be displayed again, but also a system will hopefully learn not to bother a user with similar changes. Although the introduction of the never-show class of changes increases the flexibility of a system, it unfortunately makes learning more difficult. The experimental results from Syskill & Webert [204] have indicated that at least 30 training examples are needed for achieving the classification accuracy, which is better that simply guessing. The learning task of Do-I-Care agent is with no doubt harder, which potentially can result that many users will give up before a system reaches the acceptable accuracy.

The drawback of utilising the naïve Bayesian classifier, as the single machine learning approach for modelling user preferences, mostly lies in its unfeasibility to quickly enough adapt to rapid changes in information needs. The ability of the Do-I-Care agent of being able to automatically forget old ratings after the user-defined period of time [178] seems to be not powerful enough solution. Thus, the modelling of these rapid changes in user's interests through the application of an additional strategy, such as the nearest neighbour algorithm in News Dude [40], should be a reasonable choice. The preferred properties of this additional modelling strategy are not only its ability to quickly adapt to novel user's interests, but also its relatively low complexity, since the naïve Bayesian classifier is very expensive to be trained by reason of computing the information gain values on every user basis. Hopefully naïve Bayesian classifier and nearest neighbour algorithm have different strengths and weaknesses, and one hybrid approach can outperform each single strategy.

The nice ability of the Do-I-Care agent lies in enabling the collaboration among like-minded users without introducing any, even tiny, additional efforts. Since all noteworthy changes are reported also through a Web based interface, one Do-I-Care agent can simply

monitor the pages where other Do-I-Care agents report results. The gathering of changes from other Do-I-Care agents, which correspond to other humans, increases performances, since the expertise of various users is utilised. A single Do-I-Care agent should perform only post-filtering on changes, found by other agents, in order to select only the relevant ones. The problem of finding other Do-I-Care agents, which correspond to like-minded users, is assumed to be manually solved and is not addressed in [178][246]. The possible solution can be based on collaborative filtering, which finds similar users by comparing the existed user profiles, being encapsulated into the various Do-I-Care agents.

It is reasonable to conclude that the Do-I-Care agent provides interesting solutions for monitoring known Web pages and when needed notifying its user. The cornerstone is in the application of the naïve Bayesian classifier for learning user interests and afterwards deciding which changes are worth to be pushed. Even though the ability to send the right changes and to escape faulty notifications, which are almost as worse as none, critically depends from classification accuracy, the presentation of the Do-I-Care agent lacks from any experimental evaluation. The presented ideas seem to be appealing, but the noisiness in training examples can prevent the agent of reaching the accuracy that is necessary for the practical application. The given comments for improving the classification accuracy by using multi-strategy approach, which will be maybe aware of current runtime situation, can become critical for a system to stay usable.

2.5.3. News Dude

The application of a multi-strategy machine learning approach, consisting of separate models for short-term and long-term interests, makes the foundation for the News Dude system [40][41] that is able to compile a personalised daily news program for individual users. As demanding schedules often prohibit people from a continuous computer access, the News Dude agent is intended to become the part of an intelligent radio, which uses a synthesized speech to read only the selected news stories to a user. The selection of really relevant stories seems to be unfortunately very difficult as a user is usually interested in a range of topics, its interests can occasionally rapidly change, and finally the novelty of a story is something that makes it interesting. These challenges are addressed through the specially designed hybrid model of a user that both exploits the strengths and omits the weaknesses of the nearest neighbour algorithm and the naïve Bayesian classifier. Thus, it is able to efficiently represent the multiple interests of a user in diverse topics, reasonably quickly adapt to its changing interests even after the long preceding training period, and reliably avoid a presentation of the same information twice by tracking stories that a user has recently heard.

The short-term interests are modelled by nearest neighbour algorithm mostly because of its ability to quickly adapt to the user's novel interests [40]. While most other learning algorithms often require the large number of training examples to determine the strong patterns, the nearest neighbour algorithm can successfully identify, based only on a single story, following future stories from the same thread. Every new story, which should be classified, is compared with old stories by computing a cosine distance measure between corresponding term frequency inverse document frequency vectors. The requirement to model only the short-term user interests is fulfilled by taking into consideration only one hundred most recent old stories, being also important for making the prediction of nearest

neighbour algorithm to be independent from the number of training examples. Old stories, having at least given minimal similarity with the new one, will be selected and named as voting stories, and they will be afterwards used to predict the relevance for the new story. As the new story is always compared with the most recent old stories, nearest neighbour algorithm can additionally provide needed consolidation activities, which ensure that the same story is never shown twice to a single user. The drawback of the nearest neighbour algorithm unfortunately is in its unfeasibility to classify every single news story because no good voting stories can be always found, and additional classification algorithm has to be used.

The additional learning approach, which is responsible for modelling long-term user's interests, is the same naïve Bayesian classifier that has been successfully used in previous authors' system Syskill & Webert [205]. The noteworthy difference is concerned with not selecting the most informative words, which will be then features for classification, by computing information gain as in Syskill & Webert, but by manually choosing 200 most domain specific words. The core motivation, for not performing anymore the expensive computation of information gain, lies probably in the good practical experience with informative words, being manually selected by users in Syskill & Webert [204]. The drawback of not having the information gain values for different words is the inability to utilise a multi-strategy approach for smoothing zero probabilities, which has shown the superior performances in [43], and in being therefore forced to always apply one single technique, such as the Laplace correction. Stories are unfortunately still represented as the Boolean vectors, which only contain information about the presence and the absence of words, whereas the reliability of the classifier is improved by additionally insisting that at least three words, being the indicators of interests, have to all clearly indicate either the relevance or the irrelevance of a particular story, in order to permit classification [40].

The added value of the News Dude system lies in effectively combining the realised nearest neighbour algorithm with the naïve Bayesian classifier in order to inherit only the advantages of these approaches. On the one hand, the nearest neighbour algorithm tends to have high precision by reason of directly comparing the new story with most recent old stories. On the other hand, the naïve Bayesian classifier achieves excellent recall by using the generalised model of a user. These complementary strengths are in the News Dude system exploited simply by always first asking nearest neighbour algorithm to classify a new story, being the way of using the high precision of that approach. In the case where the nearest neighbour algorithm cannot decide, the classification task is delegated to the naïve Bayesian classifier, having a high recall. A default score, being something between relevant and irrelevant, will be finally assigned only when the naïve Bayesian classifier cannot also respond. Since the stories with such a default score will be recommended to a user after relevant but before irrelevant stories, a system performs the active learning that will explore the new areas of user's interests.

The drawback of the established combination between the nearest neighbour algorithm and the naïve Bayesian classifier is in its dependence from in advance fixed thresholds, which define when a particular strategy cannot perform classification, and either the other should be asked or the default store will be assigned. In the case where for example the threshold, defining whether the nearest neighbour algorithm is able to classify new story, is wrongly set, the whole combination of strategies will not work correctly. The one of

the possible improvements can be concerned with trying to learn the optimal thresholds during a runtime based on a user feedback. When the nearest neighbour algorithm is not giving good results, the requirements, regarding the needed similarity for one story to be selected as voting, can be increased. As a result, the competence of the nearest neighbour algorithm will be reduced, the naïve Bayesian classifier will be consequently more often asked to estimate the relevance, and hopefully the hybrid approach will shown a better behaviour.

The News Dude system is also important by reason of effectively combining the explicit and implicit feedback indicators to enable efficient learning. On the one hand, the explicit communication assumes that a user can interrupt the synthesizer at any point and provide the feedback values, such as interesting, not interesting, explain, more similar stories, and already known. On the other hand, the point in time a user provides feedback or interrupts a news story is informative, and can be utilised as the implicit feedback. The underlying assumption is that users will listen longer stories being considered interesting, and therefore the amount of time that a story has been heard is useful. These explicit and implicit indicators are combined by manually defining the set of rules for assigning the score to the story based both on the proportion pl of a story a user has heard and on the provided explicit feedback value. In the case where a story is said to be uninteresting, the assigned score is $0.3 * pl$, for interesting stories, the score is $0.7 + 0.3 * pl$, and when the user has asked for more similar stories, the score is 1. While the constants 0.3 and 0.7 are currently intuitively set, it seems to be worth of establishing a tuning set of stories, which will be used to determine optimal values for these constants that will afterwards maximise the classification performances.

The unique contribution of News Dude is in defining so-called concept feedback [41], being the user's opinion about the usefulness of the generated explanation. That opinion can be used to directly modify underlying models, which can consequently both reflect more accurately user's preferences and require less training examples. The underlying logic is to create rules, which define how to change the model of a user when receiving a concept feedback value for a particular type of the explanation. On the one hand, when an explanation is based on giving the informative words that are responsible for a particular recommendation, the artificial training example, having only these explanatory words, is created together with the received user feedback value. It is afterwards added to both the nearest neighbour algorithm and the naïve Bayesian classifier. On the other hand, when the feedback value is obtained for the explanation, which says that a particular story is recommended by reason of being similar to another old story, that old story will be again either added or removed from the short-term memory of the nearest neighbour algorithm, depending from either positive or negative received feedback value. Although the usage of a concept feedback seems to be appealing for performing the direct manipulations on learnt models, the more extensive evaluation, being unfortunately not performed in [41], is necessary before making any statements about its usefulness.

While Syskill & Webert system [205] has tried to manage everything by always using naïve Bayesian classifier, News Dude manages to further improve classification accuracy by additionally utilising the nearest neighbour algorithm. A realised hybrid model allows for taking advantage of the accelerated learning rate of the nearest neighbour algorithm, while retaining an ability of the naïve Bayesian classifier to prioritise stories based on the

user's general preferences. The realised sequential combination of these two strategies is unfortunately based on the pre-defined rules, which strictly specify the cases where every approach will be utilised. It consequently critically depends on the quality of the defined selection rules, and lacks an ability to improve itself during a runtime based on a received user feedback.

2.5.4. Daily Learner

The ongoing trend towards personalised information access has resulted in the further evolution of the News Dude agent [40][41] into Daily Learner system [42][45], which either provides the selected news through a web-based interface or supports wireless information devices, such as PDA. On the one hand, several components of the News Dude agent, such as utilised multi-strategy hybrid model and nearest neighbour algorithm, are completely the same also in the Daily Learner system. The sequential combination between nearest neighbour algorithm, being again responsible for modelling short-term interests, and naïve Bayesian classifier, taking care of the long-term user's preferences, is still based on the same static rules. The realisation of the nearest neighbour algorithm is also unchanged, and it is additionally used for avoiding that the same information is presented twice to the user. On the other hand, significant improvements are introduced for both modelling the long-term interests and simplifying the feedback gathering from wireless devices, mostly to enable the successful deployment of Daily Learner system in real world scenarios. The naïve Bayesian classifier is improved to take care of the term frequencies while both selecting informative words and modelling text documents. The costs of the wireless information transmission together with the usability constrains are influenced that Daily Learner system aims to minimise the required interaction between user and device, as well as the amount of data being transferred between device and server. The following discussions will be focused on the improvements that these novel aspects are brining to Daily Learner system.

It is well accepted that the classification accuracy highly depends on the features that are selected to represent training examples [204]. The solution, based on information gain, from the Syskill & Webert system [205], has achieved excellent classification results, but unfortunately together with extremely high computation costs, by reason of having to select informative features separately for every user. Such an approach for the selection of important features is consequently of a very limited practical use, and it cannot be deployed in any scalable system due to its prohibitive computational complexity. A News Dude agent [40] is fully aware of these problems, being concerned with the computation of information gain values, and therefore proposes the fully manual selection of informative words. Even though being practically with no costs, the manual selection of features is unfortunately applicable only on small domains. The insertion of new domains can be also problematic because of requesting the additional human efforts. These are the main reasons for adopting the feature selection technique in Daily Learner system to first choose from every story ten words, which have the highest term frequency inverse document frequency values. The 150 most frequently selected words will then form the needed feature set, which will be used for representing documents to be classified.

Such selected words are the same for every user, and they should be recomputed only from time to time in order to reflect the changes in the underlying collection of the

documents. The price, being paid for the application of this much more inexpensive computation process, lies in losing the dependence on particular user's preferences. There are no two users that are the same, and consequently it is reasonably to expect that for different users various words will be informative. The extensive evaluation of Daily Learner system in [45] unfortunately does not discuss effects of this simplified approach for the selection of features. It is especially worth of trying to apply the improvements, discussed in [204] where the approach based on information gain has been presented, and that are for example concerned with the usage of lexical knowledge base. The existence of these different approaches for feature selection also naturally imposes the idea of trying to combine them. For very important users, the expensive approach, based on an information gain, can be reasonable, whereas for other users, these other cheaper selection schemes should be taken into consideration.

The second noteworthy modification of the used naïve Bayesian classifier is in modelling text documents not anymore as Boolean, but as term frequency vectors. Not only that the presence or absence of each informative word is modelled, but also the number of times each feature occurs in every document is taken into account. These term frequencies are used to modify the formula, used for computing the probability of a certain document belonging to a particular class, by giving the higher influence to terms with larger frequency. The advantages of not using anymore Boolean vectors for modelling candidate documents are unfortunately not experimentally examined in [45]. A computation of probabilities when term frequencies are taken into account is obviously more expensive, and the question is whether this price is compensated in a quality domain. Maybe both modelling approaches can be combined in order to be able to decide in a given situation which one is a better choice. Even in the case where one approach always provides better results, it can be also constantly computationally more expensive, and consequently, when system is highly loaded, it cannot necessary provide the optimal ration between quality and response time.

The News Dude utilises as implicit feedback the amount of time that the user has listened the story, being feasible by reason of using the synthesizer for reading the stories to the user. Daily Learner is not based on the usage of synthesizer, and consequently such a time-coded feedback is not possible. The pricing scheme for wireless information access, where users are charged for all transmissions, hopefully provides another heuristic for obtaining the implicit feedback. In order to minimise costs, users have to always explicitly ask for each paragraph of a story, where it is logical to assume that they will never request following paragraphs, when finding the story being not interesting. It therefore becomes feasible to again establish a direct correlation between the amount of a story being read and the level of interestingness. The novelty of wireless version of Daily Learner is in basing feedback only on implicit indicators, being necessary to improve the usability of a system. Due to the pricing scheme, as soon as the user selects a headline in order to request the first paragraph of a story, story is marked as interesting and the initial score of 0.8 is assigned. This score will be increased as following paragraphs are requested proportionally with the amount of read story. In the case where a story headline is skipped, that story will be marked as irrelevant only when at least one other headline has been selected. In other words, if user does not select any story, implicit feedback will not be generated, since maybe the loss of connection, and not the relevance of all stories, is responsible.

The novelty of Daily Learner is also in providing the query search functionalities, which will determine the final ranking by combining both the keyword matching and the user's personal interest profile [42]. A list of results for a query is therefore personalised, and for a particular user more relevant headlines tend to be transmitted first, which is important since headlines are sent in small batches. The realised combination of these two ranking functions is unfortunately not addressed in [42] in spite of nice opportunities to learn which ranking scheme is more applicable in which situations.

Since Daily Learner inherits many parts from Syskill & Webert and especially from News Dude systems, the comments that relate to more sophisticated combination of different machine learning algorithms are still valid. Daily Learner System additionally introduces a novel strategy for selecting informative words that will be informative features, and therefore opens new opportunities for establishing effective combinations. Although the necessity to address issues, such as efficient database management, computational complexity, and scalability to large number of users, is clearly stated as design goal in [45], proposed approaches lack from needed corresponding experimental evaluations. This critical presentation of Daily Learner has tried especially to stress the opportunities for improving the system as far as these scalability issues are concerned. It is absolutely clear that the Daily Learner system periodically accesses various news providers, downloads news stories and stores them in a local database in order to be able on request to compile personal newsletter, but it remains a mystery which strategies are used for that, and especially how they can be combined to support better scalability when the number of both users and news stories increase.

2.5.5. NewsWeeder

The creation of a personalised list of news articles, based on learnt preferences for a particular user, is the cornerstone objective of the NewsWeeder system [160]. While the users' interests are learnt exclusively from their explicit feedback, a simple combination between content-based and collaborative filtering strategies is responsible for the creation of needed recommendations. After briefly giving the overview of realised strategies, the following paragraphs will focus discussions on possible improvements and extensions of a deployed combination.

To predict a user rating for an unseen document, NewsWeeder offers one collaborative and two content-based filtering strategies. On the one hand, collaborative filtering uses the ratings of other similar users in order to infer the interests of active user towards the new article. Since a sufficient data on multiple users unfortunately has not been available, the evaluation of a system in [160] is performed solely for content-based strategies. On the other hand, two different content-based strategies are implemented, where the first is based on the application of minimum description length principle for finding descriptive words, and the second is a well-known nearest neighbour algorithm with the famous term frequency inverse document frequency scheme.

As the accuracy of most classification algorithms critically depends from words that are selected to represent both training and testing documents [204], a special attention is given in modelling as good each category as possible. The used assumption is the same as in Syskill & Webert [205], stating that each word has either truly specialised distribution for a given category or exhibits only random background fluctuations. While information

gain was the solution to select only distinctive words in Syskill & Webert, NewsWeeder decides which words should be left out regarding a minimum description length principle, which finds the right balance between simpler models and larger models that produce smaller errors when explaining training data. The selection of right words is facilitated by also automatically excluding words that occur less than three times in a whole collection. After every category description is established, the similarity between a given document and a particular category is inversely proportional to the length of a description, needed for the representation of that document by using the formed model of a selected category.

The nearest neighbour algorithm establishes the comparison of term frequency inverse document frequency vectors by computing the cosine of angle between them in order to emphasise the stronger information content in a word appearing. On the one hand, the accuracy is boosted by additionally eliminating the most frequent words since they tend to be less informative. The performed experiments have shown that removing between 100 and 400 most frequent words, always yields precision value, which is notably better than not removing these words at all. On the other hand, the algorithm is speeded up by forming a prototype for each category, which makes a prediction phase to be independent from the number of training data. While the straightforward prediction of a rating for new article should only find the closest prototype and use its rating, NewsWeeder performs a linear regression on the ratings being predicted by all prototypes. A predicted rating will consequently depend not only from the closest prototype, but also from the remainder of a collection, which is the goal that many information retrieval systems try to achieve.

The first noticeable drawback of NewsWeeder system lies in the needed huge amount of training data, being necessary for the accurate selection of informative words in the case where user is providing five different ratings. Although the content-based filtering that uses minimum description length principle is reported to outperform significantly the nearest neighbour algorithm, the evaluation is done by monitoring for one year two users that have rated thousands of articles. Since users are very inpatient beings, many of them are willing neither to wait several months nor to rate huge sets of documents. The natural workaround is already proposed in News Dude [40] system, which successfully combines nearest neighbour algorithm, being capable to quickly adapt to the novel user interests, with naïve Bayesian classifier, being similar to here used minimum description length principle. The very slow learning rate of the utilised machine learning algorithm was probably responsible why most of the forty registered users have given up [160], and the combination with another content-based strategy that is able to compensate that weakness seems to be the only possible solution.

Even though the idea of combining content-based and collaborative filtering strategies is given in NewsWeeder system, the implemented combination, being based on weighted summing with both non-adaptive and arbitrary set of weights [160], is at least immature. In the case where the system assumes the availability of the huge number of user ratings, being necessary for machine learning, it seems natural to try to learn which strategy type is more suitable for which users. While some users prefer accurate recommendations that content-based filtering can provide, others maybe like the inbuilt serendipitous nature of collaborative techniques. A system, which tends to provide the comprehensive provision of information, has to take care of such and the similar tastes of users to survive a course where many concurrent applications are only few clicks away.

2.6. Content-Based Filtering

The conventional content-based filtering techniques, which simply aim to recommend items being similar to those that have been liked in the past by a given user, are not only very often combined with other strategies, but also sometimes more or less solely utilised. They are especially successful in the domain of text documents, which are usually easily analysable by machines. A remainder of this sub-section will present various systems that serve different purposes, ranging from the recommending scientific papers to facilitating the browsing through search engine results, mostly by applying a content-based analysis. Both famous CiteSeer and Adaptive Web Site Agents systems can be quite successful in finding good scientific papers mainly by exploiting directly or indirectly various content-based techniques. By maintaining content-based user profile that is adapted based on the received feedback, Lira is searching for the relevant web pages. Letizia and Let's Browse achieve the local reconnaissance by analysing the content of web pages, which are behind hyperlinks on a given page. Content-based filtering is also combined with the frequent set mining in WebMate to help the user refine its queries, and with back propagation neural networks in PIAgent to achieve the high precision in selecting relevant news articles. The several useful visualisation schemes are made possible in Insyder solely due to a content-based technique, which estimates a relevance of documents on a segment basis. CREDO is finally illustrating that the formal concept analysis, being vitally supported by content-based filtering, can create the so-called document lattice that facilitates browsing through search engine results.

2.6.1. CiteSeer

To improve the dissemination and retrieval of the scientific literature from many noisy and disorganised environments such as Web, NEC Research Institute has developed the CiteSeer system [51][52][53][113][162][163][164], which performs information filtering and knowledge discovery functions that keep users up-to-date on relevant research [53]. The cornerstone idea behind CiteSeer system is basically very simple, and it is contained in downloading papers that are made available on the World Wide Web, indexing them to enable keyword querying, and performing data mining on semantically unstructured and semi-structured documents in order to extract metadata, such as title, author, abstract, citations, and so on. Following paragraphs will try to present the deployed sophisticated algorithms and comment possible ways of improving them especially from the scalability point of view.

Scientific papers are located by sending queries with various keywords, such as postscript, pdf, technical report, conference proceedings, and so on, to multiple search engines [162]. By doing that, the coverage of World Wide Web is significantly improved, and therefore there are greater chances that more papers will be retrieved. That something is a research document is checked by testing for the existence of either reference or bibliography section [164], and accepting only publications written in English [52]. The consolidation strategy, which should ensure that there are no duplicates, is based on computing the percentage of identical sentences between all documents. Even though this sentence based technique can very reliably identify duplicates, it is obviously not scaling well, both because the number of different sentences grows quickly and the comparison of sentences seems to be expensive. Maybe the consolidation solution from Fab system

[29], which represent every document with only the ten most important words, in spite of maybe not being so reliable, looks as better option for the system, which had more than 250.000 scientific papers [53] five years ago.

After the downloaded papers are transformed into plain text by using the conversion functionality implemented in the New Zealand Digital Library project, a whole indexing process can take place [113]. While one of the first releases of the CiteSeer system [52] has performed the stop word elimination, the last versions [162] index also stop words by reason of allowing higher precision search. This is especially truth when authors' initials, which are normally ignored by reason of being stop words, are given as the part of a query. Since the queries with authors' initials are very frequent in CiteSeer, the response time is optimised by cashing the inverted lists that correspond to single letters. The intermediate solution for handling the stop words, which is worth of being tried in the CiteSeer system, can be the one where only single letters are not eliminated because of their importance for authors' initials.

The realised functionality, where CiteSeer system shows its real power, is concerned with automatically parsing and identifying citations to the same paper in spite of being in different formats [113]. This problem is made less severe by performing normalisation of citations, being actually conversion to lowercase, change of hyphens to spaces, expansion of common abbreviations, such as "conf." to "conference" and "proc." to "proceedings", and removal of extraneous words, such as "in press" and "to appear", that do not occur in all instances of the citation. In spite of the performed normalisation activities, the great seriousness of this task has resulted in four different strategies, which are capable to identify citations to the same article [163]. The first one uses the variation of Levenshtein distance, which means that citations are treated as strings, and the difference is simply the number of insertions, deletions, and substitutions required to transform one citation into the other. While the second strategy is based on computing the number of same words in two citations, the third one is additionally taking care about the number of same phrases, being every sequence of two neighbouring words. The forth used strategy utilises different heuristics, such as that the names of authors always proceed the title of a paper, to extract the content of various fields of each citation. The performed experiments on manually clustered collection of around five hundred citations [163] have shown different behaviour of these strategies in both quality and response time domains. On the one hand, the algorithm based on word and phrase matching was found to have best accuracy, which is even sufficient for an unassisted use in an automated citation indexing system. On the other hand, the algorithm that extracts the subfields of the citation does not have so great accuracy, but has extremely short response time. The CiteSeer system does not even try to exploit the found strengths and weaknesses of these various strategies for matching citations, and always uses the third strategy that is the most accurate, but at the same time very expensive. The natural way of improving the overall scalability of CiteSeer is to take care of the current load of a system, and in the case where not enough resources are available, instead of using the expensive strategy for the comparison of citations, selects the one that will perform matching in the reasonable amount of time.

The importance of the exploitation of various strategies is better understood in the case of the computation of a similarity between different papers. This activity is in the CiteSeer system supported by three different strategies [113], which are also combined in

order to obtain greater accuracy than any single method alone can offer. The first strategy for comparing scientific papers is based on measuring the edit distance between the headers of articles. The assumption, behind actually using the variation of Levenshtein distance on headers, being everything before the abstract of a paper, is that every header contains very important information about the document, and that the presence of words in the similar arrangements indicates articles of similar origin [113]. The second strategy exploits the well-known term frequency inverse document frequency scheme for performing the comparison of the content of papers. In order to reduce the complexity of the comparison process, CiteSeer currently uses only the top twenty components of each document, being done since the truncation may not have a large effect of the quality [113]. The third approach represents the unique contribution of the CiteSeer system, since it utilises the citation structure in scientific papers. The cornerstone assumption is that if two papers cite some of the same previous publications, then these two articles may be related [51]. One step further is in weighting the citations based on their occurrences in the whole collection, and doing the analogous things as the term frequency inverse document frequency scheme does in a document retrieval domain. This specific approach, named common citation inverse document frequency scheme, models that when a very uncommon citation is shared by two documents, this should be weighted more highly than the sharing of a citation made by the large number of articles [113].

The developed strategies are combined as the weighted sum, where it is manually set that the weights, being assigned to the first two strategies, are the same, and are at the same time two times smaller than the weight, which correspond to the third strategy. These manually set weights actually model the reasoning that similarity measure based on citation matching is two times more important than other two measures. The obvious drawback of such a combination, being also noticed by the designers' of the CiteSeer system [113], lies in its unfeasibility both to exploit the present runtime situation and to utilise the properties of the currently compared documents. The optimal weights cannot be static values, and it is worth of waiting to future publications about the CiteSeer system, which will hopefully describe learning techniques for automatically determining the best weights.

The last but not the least important property that will be presented is concerned with notifying the user when the relevant paper has been found. On the one hand, CiteSeer system can be used in the passive pull mode, which usually assumes that user is first submitting the keyword query to retrieve the initial set of relevant papers, and then browsing the literature by following the links between the articles made by citations [113]. On the other hand, user can specify its preferences, being relevant keywords, liked papers, and selected citations. Always when a paper, which matches at least one of these heterogeneous preferences, has been found, CiteSeer will push that paper to the user via email. The problem which is unfortunately not addressed is concerned with not setting in advance, but adapting during runtime, the threshold that separates relevant and irrelevant documents.

One of the last publications about the CiteSeer system [53] presents that both explicit and implicit user feedback values can be used for profile adaptation. While the explicit communication with the user includes selection of relevant documents and citations [51], the used implicit indicators of interests are such as viewing details about the article, and

downloading the given paper. The usage of these implicit indicators actually means that user browsing behaviour is observed, which sounds appealing for the development of personalised information systems. The automatic profile adaptation, being one of the last introduced CiteSeer properties [53], is still an active research area, and the future version of a system should practically demonstrate its usefulness.

The CiteSeer system represents the successful practical demonstration that the set of heuristics, such as that the tile of a paper is something that has the largest font on the first page [53], and that author names proceed the title of a paper in a citation, can provide the surprisingly accurate extraction results. By automatically parsing citations, CiteSeer can easily create the citation graph and afterwards identify hubs and authorities. Since hubs are essentially the scientific papers containing many citations to authorities [162], they actually correspond to introductions to a particular field, survey, tutorial or review style articles. The successfulness of the Google search engine [58], which also uses hubs and authority analysis as the auxiliary ranking function, gives the additional motivation for making the similar graph analysis on the citation level of scientific papers.

The high popularity of CiteSeer system, which has already served millions of requests [53], has unfortunately uncovered its serious scalability problems. The various more or less powerful strategies, for either performing the extraction of citations or comparing different papers, are already realised, whereas one comprehensive resource aware combination of them is something that is missing. Taking care about available resources can hopefully be one mean for improving the scalability of CiteSeer in the future.

2.6.2. Adaptive Web Site Agents

This agent system plays the role of an intelligent person, who knows well the content structure, and the access patterns of a particular Web site. It then exploits that knowledge to guide a user and recommend relevant documents [201][202][203]. The utilised idea is to augment a Web site with an intelligent agent, being able to help users locate the items of interest and efficiently explore parts that contain the pertinent information. The major mechanism is the application of different strategies for making recommendations, as well as their integration into a single system. The remainder of a discussion about Adaptive Web Site Agents is focused on critically presenting the advantages and disadvantages of the realised combination among different strategies.

Every strategy has its unique properties, which are always more or less applicable for satisfying the information needs of various users. Some users for example in the domain of scientific publications prefer reading papers that are cited by a particular author, while others relay more on papers having the similar content. The straightforward idea to utilise this observation is to first realise all these different strategies, and then select the one that is the most suitable for a particular user. By using the obtained feedback a system should be able to first learn about the preferences of a user, and afterwards exploit them to better select strategies that will recommend documents.

The Adaptive Web Site Agents realises the following four strategies. The first is based on computing a cosine distance between the term frequency inverse document frequency vectors that correspond to documents that are compared. Since the documents that should be recommended are scientific papers, the second and third exploit the graph structure of

citations in order to find hubs and authorities [58]. The forth borrows the ideas from the association rule mining, and decide that two documents are related when being frequently accessed in combination. In order to be able to fairly combine these various strategies, which compute relevance scores on completely different ways, the normalisation based on a maximal known value is utilised. In other words, the first strategy will consequently devide the found relevance score by the maximal cosine distance value, corresponding to any two compared documents. The second and third will perform normalisation by using either the maximal number of citations in any article or the maximal number of times one article is cited. The forth will finally utilise the maximal known probability that any two documents are downloaded in the same session.

In order to discover the most suitable recommendation, each and every strategy will be asked to recommend one article and estimate its relevance. On the one hand, the found relevance of a result, together with a past user's satisfaction with a responsible strategy, defines a probability that the corresponding recommendation will be chosen. Taking care of the past successfulness ensures that the malicious strategies, which always return high relevance values, do not dominate, since a user's satisfaction with such strategies will be probably small. On the other hand, the reliability of a strategy is learned by increasing or decreasing it by a constant factor each time any recommendation is accepted or ignored. Such a selection scheme, which stochastically chooses a single recommendation, allows a trade-off between always accepting a result with the highest overall combination between its relevance and the reliability of a responsible strategy, and having a system learn about a user by presenting alternatives.

Since every single strategy has to find recommendations, the significant load to system resources might be imposed. The document to be returned to a user will be unfortunately ready only after the slowest strategy has finished its searching activities. Even in the case where the current resource situation discourages the application of a particular strategy, that strategy has to deliver a recommendation and all other strategies will have wait for it. Deployed strategies consequently cannot perform any sophisticated processing that will sometimes last longer, and the robustness of a whole system critically depends on every installed strategy. While more utilised strategies increases the chances of delivering better recommendations, stability decreases by reason of having more parts that have to work correctly. Although the realised strategies all provide the same functionality, self healing, in the case where one particular filtering strategy is sometimes not delivering results, is not supported.

The realised simplification of strategies is concerned with taking care of only the last accepted document, while new recommendations are searched for. Sometimes, this may dramatically increase the performances of some strategies that have to compute a needed relevance only with respect to a single document. For example, the fourth strategy is then only searching for the frequent sets [127] of the size two. Although the great saving of resources is naturally achieved, it cannot respond to requests for finding an article that is most frequently accessed in a combination with several others. Such tasks require longer frequent sets, where more accepted documents have to be taken into account. Obviously, the given simplification radically increases the performances, mostly by enabling the pre-computation of results. But, it also represents the greatest drawback of a system by reason of severely limiting the set of strategies that can be utilised.

Different users also have interests to various topics, and manually generated keywords, which typically exist in scientific papers, are utilised to avoid recommending off-topic documents. Simply before asking installed strategies to generate recommendations, both already seen and off-topic documents are eliminated. On the one hand, a system can learn which topic words are relevant in which extent for a given user, based on the accepted and rejected documents. Depending on the acceptance or the rejection of a document, the corresponding weights in a user profile are either increased or decreased. On the other hand, one document is treated as off-topic in the case where containing no words that are relevant for a given user above the certain threshold. Such a globally set threshold, for in advance cancelling some documents by reason of being off-topic ones, can unfortunately sometimes be too restrictive. User preferences are maybe not still learnt well, but many documents, which can be excellent for an active learning about the novel user preferences, are in advance eliminated. This simple solution for the elimination of unwanted off-topic documents obviously has drawbacks, whose treatment is unfortunately not possible in the realised framework, having only very simple strategies.

A noteworthy property is in establishing the communication with a user mostly via the implicit feedback techniques, such as the acceptance or the rejection of recommendations. Such a decision increases the usability of a system as users do not need to explicitly state its interests, whereas it also slowsdown the learning process while not every downloaded document is really relevant. The advanced users have opportunities to manually set which words will be relevant for determining that a particular document is off-topic or not, and also to configure which strategies will be turned on or off.

In spite of the noticed drawbacks, which are dominantly concerned with the inability to installed more advanced strategies without reducing the performances of a system, the Adaptive Web Site Agents system is important for experimentally indicating how various strategies can be combined to both exploit already learnt user preferences and explore its novel interests. The introduced stochastic combination of strategies has a great potential that can be utilised not only for learning about the different user interests, but also for exploring the unknown capabilities of novel strategies.

2.6.3. Lira

An exploration of the World Wide Web, which adapts to individual users, is realised by the Lira system [29][31][32] that uses the learnt profile to actively search dynamic document collections. The Lira system is intended to search the Web each night, present the best found pages each morning, and learn from obtained ratings to better perform in future cycles. While this is generally the same functionality, as being offered by the Fab system [28][29][30], its implementation is much simpler, mostly by reason of not even trying to combine different strategies for searching the Web, extracting features from the retrieved documents, and adapting the user profile. The following paragraphs will briefly present these various strategies, which are separately evaluated in Lira, and for which has been shown to usually have complementary strengths and weaknesses.

The Lira searching of the Web is guided by the actual user profile towards potentially relevant pages, whereas crawlers that are typically used in search engines blindly retrieve everything that can be found. The heuristic for determining which hyperlinks are worth of being tracked assumes that Web pages, having a high similarity with user profile, deserve

a special attention, realised by further expanding their links. As the pure best-first search strategy that maintains the list of best pages to be visited next can have problems with the very rapid expansion, the beam search provides a solution by limiting the number of links to be considered [29]. Although such a searching, being guided by a profile, seems to be simple and efficient, the obvious drawback is concerned with the necessity to traverse the Web separately for every user. This requirement has fatally limited both the number of users, as well as the amount of resources that can be given to each searching session, which has finally led to the idea of retrieving Web pages at once for all users with similar interests, as being done in Fab [28][30].

The pre-processing of pages is performed by stop words and mark-up tags elimination, stemming, and term frequency inverse document frequency weighting that additionally does the normalisation based on the length of a document. A solution of keeping only ten highest weighted words seems to be practical, since retaining many terms for the massive evaluation, and carefully selecting a smaller number of terms can yield comparably good results [32]. Although experiments have shown that the best results are obtained when between 40 and 100 words are preserved [29], the sub-optimal solution, which keeps only ten words, is reasonable mostly by reason of making less expensive the computation of cosine distance while comparing user profile and Web pages, which searching process performs very often. The noteworthy pre-processing properties are also both weighting higher words being inside mark-up tags, and using structural features, such as the length of a document or the number of pictures it contains. The available publications about the Lira system unfortunately do not evaluate the successfulness of these various heuristics, whose usability obviously depends on the type of a page, and that consequently opens an opportunity for establishing the efficient combinations that take care of the properties of a document to be pre-processed.

The last, and probably the most interesting set of strategies, is responsible for adapting the profile to better represent user interests, and consequently to better guide future Web retrieval activities. Although nearest neighbour algorithm is mentioned as one possibility for estimating the relevance of a Web page through finding and utilising the most similar already rated pages, this approach is not seriously taken into consideration by reason of its high execution costs and the requirement of having to keep vector representations of all already rated documents. Much closer are investigated and experimentally compared relevance feedback technique and gradient descent adaptation approach. On the one hand, relevance feedback adapts a profile by using the vector representation of rated documents, where the level of adaptation depends on the provided rating. On the other hand, gradient descent first compares the predicted ranking of documents, which is based on the current profile, to the ranking that is obtained directly from the user. Error vectors are afterwards computed as a difference between any two incorrectly ranked documents, and finally they are added to the user profile. Although the obtained experimental results have shown that the more complicated gradient descent scheme yields slightly better performances [29], relevance feedback is exclusively deployed because of its appealing speed and simplicity. These two adaptation techniques obviously have complementary properties, regarding the resource requirements and obtained accuracy, which can be used to maybe establish one comprehensive adaptation approach that will select strategies based on currently available system resources.

Even though Lira system attempts to learn incrementally by selecting documents to be seen on each iteration depending on the already received feedback [32], the used single profile results that different user interests pollute each other, and consequently leads to a slower adaptation. The performed studies have also shown that users are complaining that pages returned by the system are often very similar to each other, which leads towards the idea of integrating an additional filtering strategy that can provide more diverse results. In spite of these drawbacks, Lira represents one interesting trial to satisfy information needs, without asking users to form a query and explicitly articulate what they are interested in.

2.6.4. Letizia

The intelligent assistance to any user, who is browsing the Internet for the interesting information, is provided by the autonomous interface agent, named Letizia [168][169][170]. It tracks a user's behaviour and utilises various heuristics to anticipate, which links might lead to potentially relevant documents, and which should be ignored by reason of pointing to junk or not existing pages [169]. A cornerstone property of the Letizia system is in asking a user to neither explicitly state its interests by defining a query nor provide the explicit feedback about the real relevance of recommendations. Although this explicit communication with a user can dramatically speed up learning, a priority in designing the Letizia system has been given to both letting a user to browse without being interrupted and asking for help only when being unsure which link to follow. The advantages and the disadvantages of such a design decision are going critically to be given in the following paragraphs.

By reason of very often having the great difficulties to express its information needs through the exact words and phrases, the idea of trying implicitly to deduce the user's preferences can be appealing. The Letizia system is demonstrating that many heuristics, being based on the observation of the users' browsing behaviour, in spite of being weak by themselves, can contribute to a judgment about the relevance of a particular document. On the one hand, the good implicit indicators of interests are such as saving a reference to the document, following a hyperlink from the document containing that link, repeatedly returning to a particular document, and the long reading time relative to the length of a corresponding document. On the other hand, returning immediately, without having either saved the target document or followed its further links, can be assumed as the indication of disinterests. Every heuristic is weak by itself in a way that for example only the long reading time can also mean that a user has forgotten on browsing and started to do something else, but being accompanied with additionally following a hyperlink from that document, becomes much stronger indicator of interests.

As soon as relevant documents are identified by the explained pure behaviour based approach, they will be represented as the collection of keywords that will be afterward used for estimating the usefulness of available hyperlinks. The importance values of these words are finally computed by using the well-known term frequency inverse document frequency scheme, which is quick to compute and therefore enables the pre-processing of many candidate pages while the user is thinking [168]. More advanced techniques are probably not deployed since the Letizia system does not have to offer the absolute best recommendations in order to be useful, but only should give a suggestion that is better that nothing. In spite of being not perfect, these suggestions will hopefully increase the

efficiency of browsing by greatly reducing the wasted movements. Furthermore, the nowadays desktop computers that are used for browsing become more and more powerful. They thus open the opportunities for the application of more sophisticated pre-processing techniques, being able to reliably compute the importance of different words and phrases from the liked documents. Building the more accurate user profile can only increase the successfulness in identifying the really relevant hyperlinks, and that is one of the possible directions for improving the Letizia system.

The obvious motivation, for doing the concurrent and the autonomous exploration of hyperlinks relatively to a user's current position, is in helping a user escape many of the dead-end links that waste its time. Letizia scouts out new pages before a user commits to entering them, any by performing these reconnaissance activities [169], it helps a user guess whether a given link will lead to the reward of another interesting page or to the disappointment of a junk page. The not so visible benefit that the Letizia system provides is in remembering a user to explore the brother links at the same level, which might be overlooked because of the complexity of a page. This is especially the consequence of the interface structure of many nowadays Web browsers that encourages a depth first search, where a user completely forgets the other links at the same level and exibits a tendency to follow only downward links. Letizia is compensating this drawback of Web browsers by employing a breadth first search that will analyse all links at a given level. At that way it can remind users of neighbouring links that might escape their notice.

The one of possible drawbacks of the Letizia system can be the used assumption about the persistence of interests. The strength of such an assumption is that while searching for one topic, a user might accidentally uncover the information of a tremendous interest on another seeming unrelated topic, and Letizia will bring that topic also to a user's attention. The big weakness unfortunately lies in bothering a user with something that is maybe not anymore interesting, but that Letizia does not known by reason of not having the explicit communication with a user. The user's interests are typically changing very radically, and the adaptation, by slowly forgetting words that have not been found for the long time in the identified liked documents, is simply not powerful enough.

A user modelling is in Letizia realised through a single profile, where all interests are mixed and actually pollute each other. This is obviously a problem that might be resolved either by asking a user to explicitly specify its profiles, but that is unfortunately against the principle of Letizia of not disturbing a user, or by automatically creating a new profile always when the diverse enough document is identified, as being done in WebMate [74]. As far as deciding how many links should be recommended, Letizia always recommends on request the certain percentage of hyperlinks currently available, irrespective to their relevance. In other words, it does not take care, whether many links are relevant or only few of them lead to the interesting documents, while deciding how many links will be recommended. This is obviously not very good solution because on pages with many promising links naturally more links should be suggested than on pages with not so many good links.

The presentation of the Letizia system might be concluded by stating that it represents a nice attempt to completely exclude the explicit communication with a user and to utilise only the implicit indicators of interests. On the one hand, the performed evaluation has shown that most of these implicit indicators are very weak if being taken alone. On the

other hand, their combination is more powerful, and it is able to much better identify the user's interests. In other words, while each and every available strategy in the domain of estimating the user interests is not reliable enough, their combination aim to provide the performances that are sought.

2.6.5. Let's Browse

The MIT Media Laboratory has developed an agent, whose job is to choose, from the links reachable on the current Web page, those that are likely to best satisfy the interests of multiple users. The agent is named Let's Browse [171][172], by reason of providing the assistance to the group of humans in browsing, by suggesting hyperlinks likely to be of common interests [172]. Following paragraphs will show how the local reconnaissance, being used in Letizia [170] to scout out web pages in order to help a single user more efficiently browse, can be generalised and applied to multiple users.

Before the agent can start to recommend hyperlinks that are of common interests, an initial profile for every single user should be somehow created. A system unfortunately does not have the time to start from bad initial profiles and to slowly adapt them towards real preferences, because groups are usually impatient, and good recommendations have to be quickly generated. Users are also very often not interested in filling explicit interest questionnaires, and consequently their information preferences have to be somehow implicitly determined. The applied approach, for automatically creating the initial profile for every user, actually pre-processes in advance between ten to fifty pages, which are found by performing a breadth first search around the homepage of the corresponding user. The pre-processing is performed by utilising the well-known term frequency inverse document frequency scheme in order to extract from every analysed Web page the given number of most important words, which will be included into the resulting profile.

To make an analogy with a strategy for selecting hyperlinks used in the Letizia system [170], which is comparing the content of the candidate pages to a single available user's profile in order to figure out which links are promising, the Let's Browse collection of the multiple profiles should be somehow first combined. The simplest known strategy, being actually deployed in the Let's Browse system, is to linearly combine available profiles to get the single average profile that can be compared with candidate pages at the same way as in Letizia. The drawback of this simple strategy lies in letting that the group average dominates, and consequently the strong interests of a single user are discouraged. The strategy, being able to overcome this problem, can be based on working with separate profiles, and comparing the candidate page separately to every available single profile. Since recommended pages are those that scored the best for any profile, such a selection algorithm might be unfortunately always dominated with the same profile. The possible workaround provides yet another strategy, which in turns selects a profile that will be utilised for generating the recommendations. Although this selection scheme ensures that nobody is dominating, the common interests of a group are often ignored because always one single profile controls which pages will be recommended.

The available publications about Let's Browse system [171][172] nicely discuss about these various strategies for using multiple profiles, whereas the possibility of combining them is completely ignored. Because these strategies obviously have their own strengths and weaknesses, it is reasonable to expect that the hybrid approach might have the best

performances. The briefly presented strategies also differ as far as the execution costs are concerned, where a strategy that is keeping separate profiles that are compared to every candidate Web page, is obviously the most expensive one. This further means that while combining strategies, the available resources can be taken into account in order not to select the expensive strategy when a system is really overloaded. Such a resource aware combination of strategies can be surely one of possible directions for improving the Let's Browse system.

Yet another possibility for the Let's Browse system is to try to learn to better represent the preferences of multiple users, since the current implementation works with the static profiles that are not changed during runtime. As being discussed in [171], one adaptation policy might be to simultaneously adapt every profile based on the selected page at the same way as Letizia handles a single profile. Under the assumption that a system knows who has initiated the selection of a particular hyperlink, it is able to adapt the profile of a responsible user in a larger extent. The interesting opportunity for using multiple profiles is to actually run the Letizia system with some other profile. An underlying logic is to use a profile, which corresponds to a person that is an expert in the given domain of interests, while browsing web pages in that domain. The expected effect is the so-called browsing with another person's eyes [171], where that person has the needed expertise for selecting the right hyperlinks.

The Let's Browse system is also important because of the experimental observation that when a particular page should be pre-processed in order to be compared with the multiple users' profiles, as many words as possible should be extracted. This is fully in contradiction with the statement from the information retrieval that low important words represent noise, and that is even better not to include them when formally modelling documents. In an environment with multiple users, low important words, in spite of being not very good indicators of a content, become significant in the case of being shared by other users, or in other words, they increase the chances for the larger intersection among various profiles. This is the reason why the single user system Letizia models pages only with ten most important words, whereas Let's Browse utilises fifty words for the same purpose.

The Let's Browse system has shown that multiple users can browse at the same time and that their common interests can be taken into account while assisting browsing. The noteworthy property of this system lies in either realising or postulating various strategies for combining multiple profiles and using them while browsing. These multiple strategies hopefully open many opportunities for combining them, and developing a system that is able to handle more users, as well as at the same time to provide better recommendations after a shorter delay.

2.6.6. WebMate

WebMate [74] is a personal agent that runs on a user's computer where it monitors user's browsing and searching activities, and learns from them in order to help a user in information retrieval activities. The noteworthy properties of WebMate are the modelling of a user through multiple profiles, the query refinement based on so-called trigger pair words, and the extraction of relevant words from the selected parts of liked documents.

The main design decisions, which are behind the realisation of these properties, are going shortly to be presented in the following paragraphs.

To keep track of user's interests in different domains, WebMate utilises the multiple profiles, which are created only from the documents which are marked by a user as the relevant ones. Every liked document is pre-processed on a standard way, which includes stop word elimination, stemming, and, like in Amalthaea [188], extraction of words from headlines by reason of giving them higher weights. The importance of words is computed by using the state of the art term frequency inverse document frequency scheme, and the given number of most important words will form the vector representation of a document. Not all available words are included in the vector representation probably because of both the performance reasons and the assumption that words with the low importance actually represent a noise, which will decrease the accuracy [282].

By reason of keeping the interaction with a user as simple as possible, while a user is only responsible for providing feedback values, WebMate will select a profile that should be adapted. The strategy for autonomously handling profiles always creates a new profile for every liked document until the maximal permitted number of profiles per user is not reached. As soon as a particular user has already maximal number of profiles, WebMate will merge two most similar profiles, and consequently will ensure that there is free space for the creation of the needed new profile. As far as the merging of profiles is concerned, that is simply achieved by first finding the union of all words in them and then preserving words with the largest importance in order not to exceed the maximal permitted number of words per profile.

The created user profiles can be used for compiling a personal newsletter, which is formed either by spiding the list of Web pages that a user wants to be monitored or by constructing a query from the top several words in the current profile and sending it to popular search engines. The retrieved documents are pre-processed as explained, and all documents whose similarity with the current profile is greater than some threshold will be included in a newsletter. The drawback of such a simple strategy for forming a newsletter lies in its unfeasibility to control the diversity of the selected documents, being in [243] stressed as something that is very important for better satisfying user's information needs. The lack of diversity has as the natural consequence that many very similar documents are often included in a newsletter, which is probably the outcome that users do not expect to get.

It is well known that many users have great difficulties in finding the right words for expressing their information needs. The WebMate system helps users refine their queries by building in offline mode so-called trigger pair models. These models are actually the pairs of words with a property that always when the first word appear in a given article, the second word has to be found in that document after the first one at the distance that does not exceed the certain maximal value (being in the performed experiments set to 500). By analysing the documents from the available corpus, for every word, the set of most co-occurring words will actually form trigger pairs, which can be used for a query refinement. The noticed problem with trigger pair words lies in their domain dependence, which means that the trigger pairs have to be created for every domain separately. More seriously, a user often has to manually specify which domains are relevant because of the ambiguous nature of entered query keywords.

The last but not the least important uniqueness of the WebMate system is concerned with the extraction of words from the liked document by reason of refining the search. The scenario starts by defining the original query words, which are used for retrieving the initial set of documents. A user then identifies the relevant documents, and WebMate should somehow extract words, which are important in the selected documents and which will be used for a query expansion. Instead of analysing the whole documents, which can be quite expensive when they are large, WebMate believes that the neighbourhoods of original query words in the selected documents are more important. The performed experiments [74] have shown that chunks of five words before and after original query words are more useful than the whole documents. These observations are especially encouraging when the usage of short snippets, which nowadays search engines usually return as explanation, is concerned for the extraction of refining words.

In spite of being relatively old system, WebMate has pointed out that the users need much better support for finding the needed information than an ordinal search engine can offer. Even though the utilised strategies for creating newsletters are obviously immature, an idea of monitoring the relevant Web sources and then collecting potentially interesting documents is still alive. Finally, the authors of the WebMate system have stressed that many different strategies for handling users' profiles are possible, which might naturally open the huge variety of ways for establishing the possible combinations among them.

2.6.7. PIAgent

As the information has become the one of most significant resources for business and research, both periodically scanning different information sources and pushing the found relevant articles to interested users, have also motivated a development of PIAgent [155]. While a usage of various extractor agents each supporting a particular information source is more or less typical for agent-based filtering systems, the uniqueness of a PIAgent lies in its application of back propagation neural network for separating relevant articles from others. Such a neural network approach has both its strength in optimistically providing excellent classification accuracy, as well as its big weakness in often expensive training. After briefly presenting a PIAgent, the remainder of these discussions will be focused on possible combinations with other filtering strategies, which might offer an opportunity to exploit only the advantages of this appealing neural network technique.

The periodically retrieved articles are transformed into Boolean vectors by eliminating stop words, and afterwards performing stemming. As both pre-processing steps depend on a language, some translation tools are necessary for supporting every possible article. PIAgent utilises freely available service from www.babelfish.com, whose word-by-word translation is good enough to support statistic analysis that fully ignores interrelationships between words. The obtained vector is finally presented on the inputs of a neural network, which encapsulates the interests of a particular user, and which decides whether the given article is worth to be presented. The main difficulty is concerned with the training of that network, which requires hundreds of articles that are explicitly rated by a user. Initially all network weights are randomly set, and consequently everything can find its way to the user, who can easily become disappointed with the underlying filtering engine. The given experimental results [155] unfortunately only report that on average accuracy of 80% is achieved, but with using 150 rated articles from each user. A critical problem relates with

probably much lower accuracy that can be achieved for everybody, being not enthusiastic to rate so many articles without getting something in return.

The yet another difficulty can be induced from a requirement that every distinct word has its own input neuron, which is set from a corresponding value in a presented Boolean vector. Having many input neurons, makes more difficult to find the right configuration with the acceptable classification accuracy. Although these problems are not mentioned in [155], they are fairly discussed in the original paper [49], where this neural network approach is first presented. It is there reported that even the relatively small network with only 425 input neurons had to be reconfigured for 30% of its users by reason of not being accurate enough. Such a reconfiguration can be particularly unpleasant for users, who are still not getting good results, and who do not known anything about a learning that stuck into its local optima.

In spite of the mentioned potential problems that back propagation neural network can have with text classifications for real time usage, this approach can be extremely accurate for some lucky users, who have additionally enough patience to provide a lot of feedback for training its network. It is therefore reasonable, instead of completely forgetting on this neural network inspired approach, to think about its synergy with other strategies that can provide better guaranties, and that are much less demanding regarding a users' patience. These alternative strategies can provide satisfactory results until the neural network is not trained well for a particular user. The sophisticated combination of strategies might even take care of the preferences of individual users regarding different strategies, where some users prefer collaborative and some other content-based filtering techniques.

It is finally very surprising that the communication with a user in the PIAgent is based only on explicitly rating articles. A heavy exploitation of implicit indicators of interests is common for really many modern systems, such as Letizia [170] and WAIR [60], having a learning that requires often a huge number of training examples. Measuring the reading time, and observing actions, such as printing or saving articles, can be the natural improvement of PIAgent system. Its brave trial to train back propagation neural network, having for every distinct word from a vocabulary a separate input neuron, in order to specialise and adapt to the personal preferences of each user, has deserved in spite of its serious weaknesses a peace of attention.

2.6.8. Insyder

The Insyder system [212] represents the result of efforts to increase a users' ability to accept the greater amount of found articles by deploying different presentation schemes. The underlying assumption is that there is no best visualisation [210] and therefore a user should have a choice to decide which particular representation is the most suitable for the actual task and the found amount of results. Consequently, the cornerstone of the Insyder system is a support for different visualisation schemes, where some of them are useful in representing a whole result set, and others especially aim to facilitate the inspection of the single documents [175][176]. On the one side, the examples of the visualisation solutions which support the representation of multiple documents are a result table, being similar to Microsoft outlook look and feel, and a scatter plot, that actually implements the appealing skyline idea [55][152]. On the other side, there are also the visualisations which illustrate

the distribution of the search terms inside the different document segments [175], and that are thus suitable for concluding about the relevance of the particular parts of found results.

Even though the essential idea of Insyder is in deploying many different visualisation schemes in order to obtain a similar effect as Grouper [282] reaches through a suffix tree clustering, there are many other aspects, making the Insyder system be worth of a careful investigation. In the following paragraphs the realisations behind the creation of a query, information retrieval and result ranking, are going to uncover a mystery lying within the Insyder system. A proposed solution represents the unique effort to overcome nowadays problems, being typical for the large scale information retrieval systems.

It is well known that users often have big difficulties in formulating their information needs. On the one side, some of them do not known even elementary how the information retrieval system is working. Consequently, they cannot create good enough queries. On the other side, others actually do not know what exactly they are searching for [211], and thus no precise formulation of information needs is possible. The designers of the Insyder system are fully aware of all these problems, and they are addressing them mainly on two different ways. The one way is the usage of manually created treasuries or semantic nets, which should help in finding synonyms, acronyms, broader and narrower terms [189]. As a side effect, this manually generated knowledge base also helps in avoiding the spelling mistakes. An obvious drawback of the usage of manually created treasuries is in a limited applicability to only those domains for which such treasuries already exist. Consequently, the Insyder system currently only supports a retrieval of the business related information from the Internet [189][211].

The yet other way is a usage of specialised visualisation techniques which facilitate a formulation of a query. These algorithms are actually graphically presenting query terms together with their intra relations. Additionally, they suggest terms for a query expansion. The terms for a query refinement can be obtained either from already mentioned semantic nets or through the usage of relevance feedback techniques. A relevance feedback is used on a classical way that a user is first selecting the liked documents, and afterwards these documents are analysed in order to extract the most important terms from them. Finally, the extracted terms are suggested to a user for the refinement of a query. Therefore, a user is completely controlling what is happening with its query, and absolutely nothing is done automatically.

Finally, each and every user is stimulated to enter as many query terms as possible as a wider text field for entering query terms is provided [189]. This goes together with many already performed experimental studies, which have shown that the users are on average providing more query terms when they have a feeling that a system is expecting more terms as an input. The wider box is a way how the Insyder system is sending this message to its users, and therefore fighting against many short queries. Unfortunately, these short queries dominate in nowadays search engines, where the average length of a query is not more than only few terms.

The additional added value of the Insyder system is besides contained in a component, being responsible for the extraction of the data from the Internet [189]. In order to create good starting positions for retrieving only the potentially relevant documents, Insyder is using search engines to which the actual query is forwarded. But, Insyder goes one step

ahead as compared with traditional meta-search engines. After the results that are found by the selected search engines are retrieved, they are used for spreading by extracting the links from them. The underlying documents are finally retrieved to be carefully examined.

By performing the explained spreading Insyder might potentially retrieve much more relevant documents [189]. Moreover, because each document is downloaded before being recommended, the problem of recommending something that does not exist anymore is overcome. Unfortunately, the query processing in such a manner is not anymore possible in on-line mode, because the duration of filtering will be by sure longer than few minutes. That can be a main reason why in the papers about the Insyder system the response time of a system when new queries are posted is not taken into consideration.

From the information filtering perspective, the most interesting is a ranking algorithm that the Insyder system offers [212]. A used ranking algorithm is fundamentally different from the well known term frequency inverse document frequency approaches, by reason of estimating a relevance of a single document solely by using its content and not relative to a whole article collection. Even though a typical information retrieval researcher has to be somewhat sceptical as far as one such ranking is concerned, the designers of Insyder have compared their ranking scheme with the other approaches, and have found out that it clearly outperforms them.

An idea behind the ranking of Insyder is in separating a whole document into multiple segments, which are nothing else that the sentences or paragraphs. After the segments are identified, the relevance of each segment will be estimated simply by checking how many query terms there occurs. As far as query terms are concerned, Insyder assumes that more advanced users might additionally assign the importance values for every query term in order to specify that some query terms are more important than the others. These weights, when available, can be used afterwards to better assess the relevance of each and every document segment. The found relevance values for every segment are additionally used when every document is visualised to illustrate how every segment is relevant to a query. Therefore, they can greatly facilitate a decision making process about the relevance of a document that a user has to make by itself.

As far as putting together the relevance values of separate segments into the overall relevance of a document is concerned, the deployed algorithm takes care of not only the mean relevance values that different the article segments have, but also of the maximal relevance value that is reached in one segment. In order to stimulate the selection of the documents where the high relevance score is found inside a single segment having most query terms, the maximal relevance value has a priority over the mean relevance. In other words, the heuristic behind says that it is more important that one segment has the high relevance to a query than that many segments have an average relevance. This reasoning is only logical under the assumption that users prefer the articles where query terms are concentrated in one its part.

This short summary about the Insyder system can be finished by stressing that Insyder represents a successful trial to overcome information retrieval problems in one particular application domain, dealing with the business analysis. The deployed retrieval algorithms unfortunately completely ignore the response time. Users are nowadays impatient beings, and the applicability of such a system will be very critical, where concurrent applications

are only few clicks away. Performed extensive user studies are all based on in advance prepared results, being the only possible way of not having a situation where the retrieval time dominates. Finally, the Insyder system is offering one novel ranking scheme, which represents a trial to fight against term frequency inverse document frequency approaches, being dominantly applied algorithms in the few last decades. Even though the first tests with this novel ranking show its superiority, the more extensive studies with much larger collections are necessary in proving its added values.

2.6.9. CREDO

The CREDO system [63][65][66][67][68] addresses the issue of improving a usability of the unmanageably large response sets of Web search engines, which make the usage of document summaries ineffective, time-consuming, and often quite frustrating for a user. The idea is to exploit a formal concept analysis for creating the conceptual representation of retrieval results in the form of a document lattice [66]. By doing that CREDO achieves practically the same effects as the suffix tree clustering of the retrieval results [282]. Both approaches have their inbuilt strengths and weaknesses. On the one side, the suffix tree clustering can lost valuable or relatively rare clusters due to the applied heuristic, which is necessary to ensure the linear complexity with respect to the number of objects to be clustered. On the other side, the formal concept analysis is always creating the complete set of clusters, but unfortunately does that with a relative inefficiency as the construction of a whole lattice usually has the serious computational problems, both of space and time [65]. The remaining paragraphs will critically give the architecture of the CREDO system, and therefore try to analyse whether the price, being paid in the space and time domains, is reasonable.

After a search engine, being currently Google, has responded to the forwarded query, the produced results are collected and parsed. Because downloading original documents from Web may take unacceptably long, CREDO considers just the information returned by a used search engine, which is usually only a title and a short snippet. While the small representations of documents are hopefully not reducing the quality of produced clusters dramatically [282], they are very important for speeding up a construction of a complete lattice, whose size critically depends on the number of distinct terms per document [65]. In order to be able to carefully control which terms will be selected to represent a given document, as soon as text segmentation, word stemming and stop wording activities [66] have identified all words, word weighting should estimate the importance of every word in each document. The goal of word weighting is to help identify words that characterise a document to which they are assigned, while also discriminating it from the remainder of a collection. The one group of the well known methods for doing that is based on a term frequency inverse document frequency scheme. It specifies that the multiple appearances of a term in a document are more important than the single appearance (term frequency), and that rare terms are more important than frequent terms (inverse document frequency). Yet another group of, maybe much more effective, methods uses term-scoring functions that are based on the difference between the distribution of terms in the collection of the retrieved documents and the distribution of terms in a whole collection [63]. This second weighting idea is simply based on the assumption that while the frequency of the really important terms will be higher in the retrieved documents than in a whole collection, the other terms will occur with the same frequency, or randomly, in both document sets. The

same term scoring function is in [63] utilised for the automatic extraction of terms from top retrieved documents for the purpose of a query refinement by a so-called retrieval or pseudo-relevance feedback. A goal in mind is to overcome a severe vocabulary problem, where a user is expressing its information needs with terms that do not necessary appear in all relevant documents.

In the case where such word weighting should be implemented in the CREDO system, the arising problem is concerned with the usage of a search engine to find retrieval results. On the one side, the obtained snippet representation of the found results is unsuitable for performing any term frequency analysis, because the frequent terms in a snippet are not necessary also frequent when a whole document is taken into account [282]. On the other side, the information about the frequency of terms in a whole collection remains hidden, which makes inapplicable both presented schemes for word weighing. Obviously, a better communication with a search engine has to be established [282], which will provide the data being needed for the appropriate scoring of words. In the CREDO system a solution is hopefully nicely found through the interaction with Google not via its Web interface, but though the exchange of appropriate XML based SOAP messages.

By using the found weights, the step of a word selection should decide which words are valuable enough to be included in a construction of a lattice. One simple and efficient solution is in selecting those words whose weights are at least for one standard deviation larger than the mean value of the weight of all words in a given document [68]. The step further is to take the advantage of presented multiple term scoring functions to increase the effectiveness and the robustness of a word selection. The utilised idea is to ensemble multiple term scoring functions by trying to guess which terms are the right ones through taking a majority vote. When a term erroneously wins a high (low) rank by one method, the same term gets a low (high) rank in other methods, so a majority procedure can rank correctly that term [67].

After the most important terms are found, a document lattice can be built. Every node of a lattice is a pair, composed of a subset of documents and a subset of the index terms. In each pair, the subset of terms contains just the terms shared by the subset of documents, and similarly, the subset of documents contains just the documents sharing the subset of terms [65]. On Figure 2.6 the five different documents are represented by a lattice, having seven nodes. For example, the node, which has the assigned terms artificial intelligence, cataloguing and expert systems, and documents with ids 2 and 5, caries the meaning that in the whole collection only the documents with ids 2 and 5 have these three terms as the maximal set of common terms. It is not hard to conclude that the size of the constructed lattice can easily explode not only regarding the number of documents to be represented and the number of distinct terms per document, but also with respect to the density of a document term matrix.

Once a document lattice is constructed, it can be used not only for browsing, as it is the case with the pure, through the hierarchical clustering built, structures, but also for the easy formulation of a query [65]. Each node might be seen as a query formed from the associated terms, and that resulted in given documents. For example, from the lattice that is given on Figure 2.6, it is obvious that a query with terms artificial intelligence, expert systems and information retrieval will result in the document with id 1 to be found. The discussion about a lattice construction has to be unfortunately concluded with a statement

that its size may largely exceed the number of documents. Due to the inherent complexity of the algorithms for building a lattice, the full document lattice may be constructed only for small to medium size collections, usually having up few thousands of documents [65]. It becomes therefore quite logical why CREDO retrieves only the first hundred of Web search engine results, being necessary in order to build the necessary lattice fast enough. That is obviously the disadvantage of the browsing solution based on lattice, especially in the case of being compared to the suffix tree clustering, which is able to cluster up to one order of magnitude more documents under the same time constrains [282].

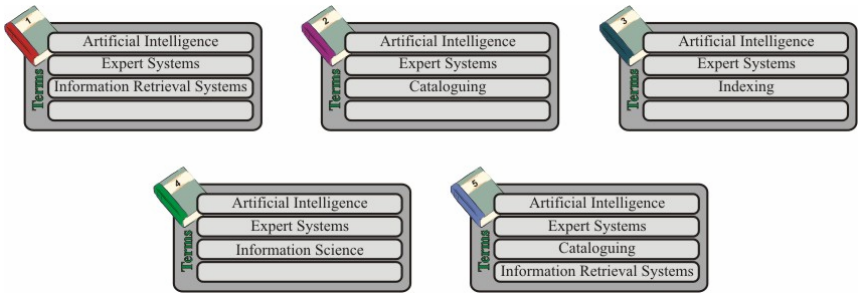

Figure 2.5: Document term matrix

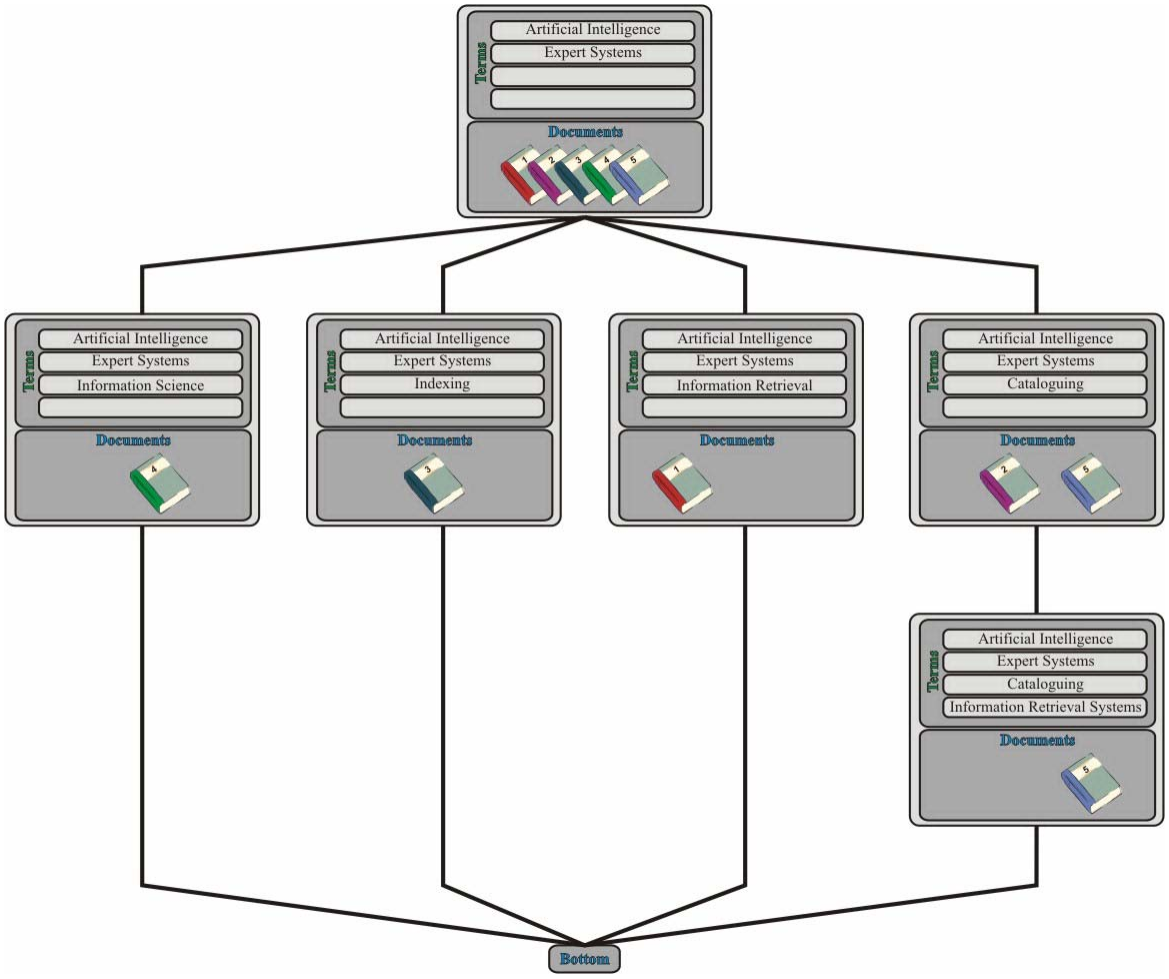

Figure 2.6: The example of concept lattice for document term matrix on Figure 2.5

The usable visualisation of a created document lattice is difficult due to the conflicting issues of size, layout, and limited screen area [66]. The possible solution is to display the information contained in a lattice in the varying levels of details depending on a distance from a focus. The one concrete realisation of that solution, being used in CREDO system, is to utilise a tree, where a selected node defines a focus that will present its children. Unfortunately, an obvious disadvantage lies in the considerable amount of the duplication of the information when nodes have multiple parents [66].

The presented CREDO system is valuable by reason of demonstrating how the formal concept analysis can be utilised as a powerful tool for eliciting context and concepts from the unstructured or semi-structured documents [66]. As far as its main building blocks are concerned, a benefit of combing multiple term scoring functions has been experimentally proven [67]. The simple combination of term scoring functions has clearly outperformed every solution that is based on any single scoring function. This result can be interpreted as an additional reason to always try to solve complex problems through the combination of different strategies that are all fully capable to autonomously and efficiently handle the imposed problem.

2.7. Clustering Techniques

Clustering is the unsupervised process of grouping the data into clusters so that objects within a cluster have high similarity in comparison to one another but are very dissimilar to objects in other clusters [127]. Obtaining a high intra-cluster together with a low inter-cluster similarity unfortunately hides many challenges, which are mainly concerned with the amount of data that should be clustered, sensitivity to outliers, dependence from the order in which objects are processed, ability to discover the clusters of the arbitrary shape, and so on. A different treatment of these requirements has resulted in the wide palette of clustering techniques, where partition-oriented [97], hierarchical [92], and density-based [93][234] are probably most commonly utilised. Since there is no perfect and universally applicable clustering algorithm [128][226], there are also various combinations, which for example try to achieve the optimal trade-off between the accuracy of hierarchical and the speed of partitioning methods [247]. The remainder of this sub-section will show that an intended purpose of clustering primary defines which method should be applied. On the one hand, Scatter/Gather aims to interactively perform the clustering of either a whole document collection or its selected part in order to facilitate browsing. Accurate but slow agglomerative hierarchical clustering is thus speeded up by either doing sampling or only locally searching for clusters that are candidates for merging. On the other hand, Grouper and DisCover do the post-retrieval clustering of search engine results, being illustrated on Figure 2.7. While Grouper searches for best clusters by using suffix tree representation of snippets [112], DisCover performs a fuzzy clustering [181] that assigns documents in the clusters based on their features.

The Google results for the query term Agent, which are represented as snippets, are on Figure 2.7 clustered into six different classes, named Information, Multi, Mobile, Broker, Travel and Trading. The clustering is manually performed regarding the meaning that the term Agent has in the underlying snippets in order to illustrate how an ideal hierarchy of clusters can help in browsing the search results. On the one hand, without formed clusters

a user that is interested for example only in Multi-Agent systems has to glance through a whole rank list of results. By taking into account its limited perception capabilities some of the relevant results will be probably missed. On the other hand, the formed hierarchy of clusters helps user focus its valuable attention only on results that might be potentially interesting. Many irrelevant results, such as for example those concerning Travelling or Trading, represent a noise for a user that is interested solely in Multi-Agent systems, and the built hierarchy facilitates the elimination of these results from a further consideration.

Figure 2.7: Post retrieval clusters of snippets, obtained from Google for "agent"

2.7.1. Scatter/Gather

The Scatter/Gather system [79][80][129] is intended to help in situations in which it is hard, if not impossible, to formulate a searching query precisely. Such situations most typically occur when the user is not looking for anything specific at all, but rather may wish to discover the general information content of a corpus [80]. The needed underlying browsing activity can be greatly facilitated through the application of fast clustering algorithms, which will scatter the whole collection into the small number of document groups, or clusters, and present short summaries of them to the user. In order not to be biased with features that are only important for a collection as a whole and not for the user, the interaction with the user ensures that one or more groups are selected for further study. The selected groups are gathered together to form a sub-collection, which is again scattered or clustered into new document groups. These scatter and gather steps are repeated under the user's control, who selects relevant groups to be first gathered and then again scattered, and therefore escape the mentioned pitfall of pre-retrieval clustering techniques of being biased by features that are irrelevant for the user.

The scatter and gather steps are illustrated on Figure 2.8. After the first scatter activity, the whole collection of documents from New York Times News Service, August 1990, is

clustered into eight different clusters, which are named as Education, Domestic, Iraq, Sports, Oil, Germany, Arts and Legal. In the real application of Scatter/Gather, the cluster content is nicely summarised by giving both most important words from the underlying documents and most typical headlines. The experimental results from [129] have shown that users are able to interpret the cluster summary information well enough to select the cluster with the largest number of the relevant documents in most cases. The fundamental advantage of Scatter/Gather regarding the static clustering hierarchies is in interaction with a user, who can always select multiple clusters for a further examination. In the first iteration, the selected clusters are Iraq, Oil and Germany, and their content will be first gathered and then scattered into eight new groups, named Hostage, Deployment, Pakistan, Markets, Africa, Politics, Germany and Oil.

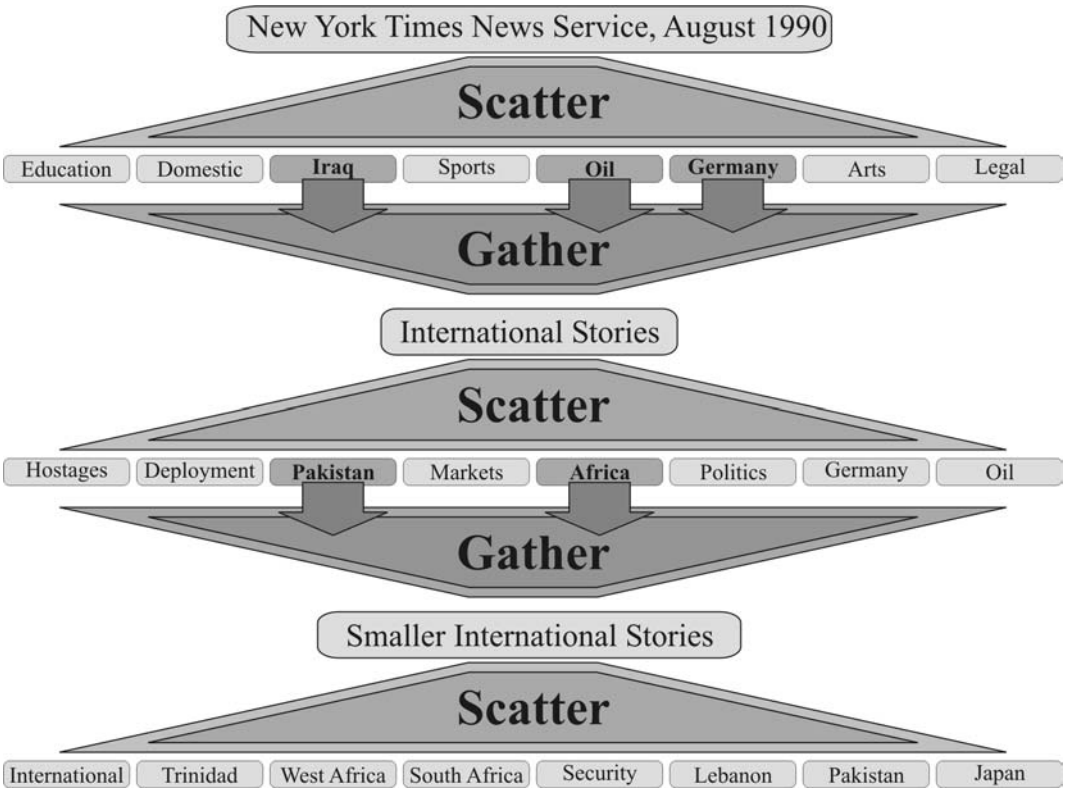

Figure 2.8: Illustration of Scatter/Gather

In the second iteration on Figure 2.8, user is selecting clusters Pakistan and Africa in order to be again gathered and that scattered in the more fined groups. The finally found groups are named as International, Trinidad, West Africa, South Africa, Security, Lebanon, Pakistan and Japan. This example of the usage of Scatter/Gather illustrates that a user only needs to select the clusters of interests, which are first gathered and than again scattered to provide the more fine-grained information. A user is not forced to describe its information needs by providing the exact terms, which have to be present in the searched documents. This information need can be obviously satisfied in spite of not knowing the occurring exact words and corresponding synonyms.

The performed user studies have shown that this Scatter/Gather browsing tool is less effective than a standard similarity search, when users are provided with the exact query

and when the goal is to find particular documents [129]. In the case where the analogy with classical book is made, Scatter/Gather is similar to the table of contents, whereas the standard similarity search corresponds to the index. Obviously, while Scatter/Gather is a better choice when the overview is needed, the index is more suitable when the searching with exact terms is required. Therefore, the Scatter/Gather can be successfully combined with similarity search by helping a user to formulate accurate query, which is afterwards resolved through the application of conventional searching methods. Additionally, the Scatter/Gather can be utilised to organise the results of the queries that retrieve too many documents. It has been shown that such a grouping actually manages to improve the precision respecting the similarity search ranking alone [129]. The performed experiment is contained in first finding n most relevant documents through similarity search, then applying Scatter/Gather on these n documents, and finally selecting all documents from the highest scoring cluster that has most of the relevant documents. The selected k ($k < n$) documents are used for computing Scatter/Gather precision, which is compared with the precision when k most relevant documents, found by similarity search, are chosen. The obtained larger Scatter/Gather precision values provide evidence validating the cluster hypothesis, which states that relevant documents tend to be more similar to each other than to non-relevant documents [129].

The efficiency of Scatter/Gather approach critically depends on the ability of the deployed clustering algorithms to quickly perform scatter activities. The novel clustering algorithms, names Buckshot and Fractionation, are developed because existing methods fail to meet needs of being in the real time applicable on large document collections. On the one hand, k-means partition algorithms have linear time complexity respecting the number of documents that should be clustered, but the quality of produced clusters can be problematic. On the other hand, hierarchical algorithms generate better clusters, but together with increasing the time being needed for clustering. The proposed Fractionation is based on speeding up agglomerative single link hierarchical clustering algorithm [127] by performing the search for the closest clusters, which should be merged, only locally. By doing that the time complexity is not any more quadratic regarding the number of documents to be clustered, but becomes linear. The Buckshot is modification of k-means partition algorithm where the initial cluster centroids are not selected randomly. They are produced by classical agglomerative hierarchical clustering that is applied on sufficiently small, randomly selected, sample of documents.

In spite of linear time complexity, the clustering in real time becomes unfeasible for very large document collections, having millions of documents [79]. In order to be able to guarantee that Scatter/Gather approach will be applicable to any collection of documents, some pre-processing activities are needed. The found solution is to pre-cluster the whole collection into a fine-grained cluster hierarchy, and afterwards the gather step, instead of collecting all documents, operates on so-called meta-documents. These meta-documents correspond to internal nodes in cluster hierarchy, where each meta-document includes only the given number of most typical words from the corresponding sub-tree. The less important words are intentionally truncated because clustering meta-documents with their dense representations, having all words in the corresponding sub-tree, takes almost as much time as clustering the documents represented by those meta-documents individually [79].

As far as the combination of the different filtering strategies might be concerned, the Scatter/Gather system is important by reason of pointing out that no single clustering algorithm can be always the best choice. While Buckshot is a fast clustering algorithm suitable for the online re-clustering in Scatter/Gather, Fractionation is yet another, more careful, clustering algorithm whose greater accuracy makes it very suitable for the static offline partitioning of the entire corpus [80]. Obviously, these clustering algorithms have their unique strengths and weaknesses, and in Scatter/Gather system is clearly, in a static manner, defines in which situations which algorithm should be utilised.

Even though the scalability is greatly improved by introducing these pre-clustering activities [79], the requirement to support large dynamic document collections seems to be very problematically manageable. The necessary offline pre-clustering can require days to process collections with millions of documents, and consequently the costs of performing necessary re-clustering activities are unacceptable. Therefore, the noticeable drawback of Scatter/Gather system is obviously concerned with its inability to be applied on highly dynamic and very large collections, such as Internet. Its applicability has to be thus restricted only on relatively static collections, which do not require the expensive re-clustering activities to be performed frequently.

2.7.2. Grouper

Instead of trying to exactly find the most useful results, Grouper is utilising the idea of better representing a huge number of found results by automatically clustering them into meaningful categories. The usage of these dynamically created categories should further increase the efficiency of accessing information that the Web has brought, and therefore make the previously impractical tasks to be even more practical. The authors of Grouper are promising that a user's ability to explore the amount of found results will be enlarged at least for one order of magnitude, as compared with the case where the popular ranked list representation is used [282]. While a very patient user might sift through one hundred documents in the ranked list presentation (GroupLens has discovered that 79% searches for movies end before 80 items [159]), Grouper should allow a user to browse through at least one thousand documents. In remaining paragraphs the basic concepts, being behind the Grouper system, will be briefly explained in order to get better impression how users can suddenly consume much more results.

The cornerstone idea is in applying fast online clustering that will create clusters with results and present cluster descriptions to a user instead of the results. In an ideal case the similar results are placed in the same cluster, and that will consequently speed up finding other relevant documents once the first one has been found. The performed user study has shown that while the rank list representation is faster when only one relevant document should be found, in the case where as many relevant articles as possible are searched for the categories of Grouper are a better solution [282]. Because users are often interested in finding many relevant documents, the advantage of Grouper against the commonly used ranked lists, is obvious.

The clustering algorithm that is applied is named the Suffix Tree Clustering (SFC) by reason of being based on a suffix tree data structure for identifying the phrases that are common to different documents [284]. Consequently, the utilised clustering assumption simply states that the more common phrases between two documents mean more similar

documents. Finally, such documents will have greater chances to be finally placed in the same cluster. There are many algorithms for the construction of a suffix tree [80] mostly because a suffix tree is a structure that is commonly used for quickly performing different string matching operations, such as finding the longest substring that appears in the given strings, determining how many times given string occurs in another one, and so on [112] [282]. The application of suffix tree in Grouper, instead of treating a single character as atomic element, sentences are inserted where single words are not further separated.

The naïve algorithm for a suffix tree construction is unfortunately with the quadratic complexity as far as the length of a string that is inserted is concerned. Consequently, the naïve algorithm is not applicable enough for the fast construction of a suffix tree, having hundreds and thousands of documents. Hopefully, the fruitful history of the construction of a suffix tree [112] has produced many other algorithms with a linear complexity, being necessary for escaping a situation where the construction of a suffix tree becomes a bottle neck. Among these different suffix tree construction algorithms, the Ukkonen's algorithm [112] seems to be the best one by reason of processing strings that are inserted naturally from left to right. More importantly, its incremental nature is especially important for an online application, where as soon as one string is available, it is inserted into a suffix tree.

The known problems concerning the construction of a suffix tree relate to the amount of data that can be efficiently presented. In a case where a suffix tree becomes too big to be placed in a memory, both its construction and afterwards its usage become unfeasible. The workaround is found in the application of other structures, such as suffix arrays [282] [285], which are more efficient when storage requirements are concerned, and which at the same time can provide almost all the functionalities as a regular suffix tree. Under the assumption that both the length of every document that should be inserted into a suffix tree is bounded by a reasonable constant and that not so many articles should be added, the limitations of a suffix tree are not really serious for the STC algorithm. By taking into account additionally that STC actually clusters short snippets that are returned by search engines, as well as that not more that one thousand of short snippets should be clustered, the storage requirements of a suffix tree will not be further analysed.

The exploited fundamental property of a suffix tree is the effortless identification of the longest phrase that is shared between the different documents. After a suffix tree is constructed, all these shared phrases have the separate internal nodes in a tree, and thus can be easily found. As far as finding of documents is concerned, they are located into leafs that have as a parent the node with that particular phrase. The STC algorithm will compute the reputation of each internal node [285] based on a phrase that is assigned to that node and the number of documents that are located in the corresponding leafs. The phrase is analysed based on its length, where short phases pay penalties, and additionally the term frequency inverse document frequency value of each phrase term is taken into account. Experiments have shown that on average 55% of phrase clusters, being among 10 top clusters, are based on phrases containing more than one word. One of the possible further improvements is concerned with not ignoring the found relevance of each document [282]. Thus, many low relevant documents will be disabled in having the high influence when the reputation of each internal node is computed.

The phrases that are assigned to internal nodes are also very good in describing these nodes, and consequently the corresponding clusters. On the one side, users can have the

enormous difficulties in understanding the cluster description, being created from the corresponding centroid, in the case where k-means clustering is deployed. On the other side, it is straightforward to understand that all the documents, which are assigned to a particular cluster, share a given phrase. The performed experiments have shown that even much shorter 2-grams are providing the same performances as much longer suffix tree phrases [282]. But, 2-grams do not have so excellent descriptive power in presenting the content of a cluster to users as longer phrases, which can be easily found only by suffix tree, and not by other alternative techniques.

The last part of the STC usage is concerned with merging the internal nodes in order to reduce the number of clusters that are presented to a user. An idea is to help a user, and not only to replace scrolling through rank list of results with the examination of the large number of cluster descriptions [283]. Because initially each internal node represents one cluster and in each step the two most similar clusters are merged, this is nothing else than agglomerative hierarchical clustering. The similarity of clusters is computed based on the number of common documents that belong to clusters which are candidates for merging. Obviously, the complexity of this clustering process is quadratic. This is the consequence of the disadvantage of the agglomerative hierarchical clustering, which is known as the technique that produces the good results but unfortunately with quadratic (single-link and average link) or even cubic (complete-link) complexity regarding the number of articles that should be clustered.

The quadratic complexity is again unacceptable for STC, and the simple solution is found in limiting the number of clusters that will be merged to a particular constant, let's say 500. By doing that only the best clusters, being the ones with the highest reputation, are going to be merged, and the complexity of the merging will not any more be with a quadratic complexity, but will be a constant. The reasoning behind is also concerned with avoiding a situation where many low scoring clusters influence the merging process. Not only that they are slowing down the merging process, but even they are spoiling the final clusters that are presented to a user.

Even though the idea with merging sounds appealing, the experimental studies have shown that for some tasks the merged clusters are even making the things to be more difficult for a user [282]. Consequently, a new version of the interface has been provided, where base clusters are not merged [282], and they are presented to a user based on their reputation, together with the number of documents that they contain. The attempt is to select the subset of high quality phrases that also provide a high coverage (the percent of documents that include at least one of phrases) of a result set. A user has additionally the opportunity to require clustering of each base cluster with the goal of simplifying the browsing through large sub-clusters. Finally, a noteworthy property is that the developed system is able to suggest the multi-word phrases to a user for a query refinement and that a user does not need to enter the required number of clusters.

STC algorithm has been compared with other liner time clustering algorithms, such as single-pass, k-means, Buckshot [80] (being nothing else than k-means where initial clusters are found by agglomerative hierarchical clustering on sample) and Fractionation [80] (being an approximation of hierarchical agglomerative clustering, where the search for the two closest clusters is performed locally). The comparisons have done first on the collection of manually downloaded snippets, then on the collection of whole documents

that correspond to these snippets, and finally on the collection of medical abstracts. The performed tests show that while STC clustering algorithm performs best on the collection of snippets and corresponding documents, on the collection with medical abstracts the k-means performs slightly better [282].

The reasons for the STC successfulness are found in the support of putting the same document to multiple clusters, being reasonable because documents often have multiple topics. Consequently, documents might be easily similar to more than one cluster, and therefore it is artificial to confine each one only to one group. Allowing a document to appear in more than one cluster also acknowledges that documents are complex objects that may be grouped into multiple potentially overlapping, but internally coherent, groups [282]. The experimental results show that STC places every document on average in 2.1 clusters and 72% of the documents are placed in more than one cluster. On the one side, where overlapping clusters are not allowed, the quality of STC decreases by more than 20%. On the other side, the assumption, stating that allowing each document to belong to multiple clusters increases the quality of other clustering algorithms, has been checked by modifying k-means and Buckshot. Their modifications simply allow in the last iteration that each document can be assigned to the multiple clusters. The obtained results have shown that both k-means and Buckshot performs better when the overlapping clusters are permitted [282], especially on short and noisy documents.

Also an interesting experimental observation is concerned with a comparison between the quality of clusters when the short snippets and the corresponding whole documents are used [285]. It is surprising that a quality does not drop significantly in the case where short snippets are used in spite of being on average 10 times shorter than corresponding underlying documents. Surprisingly, Fractionation and single pass have performed even better on snippets. This can be explained only under the assumption that snippets are very good in describing the underlying documents by extracting the meaningful words and phrases from the original documents.

The STC belongs to the group of post-retrieval clustering algorithms, which instead of pre-clustering a whole collection, cluster a result set. They thus can focus on features that are dominant in a result set, while pre-retrieval clusters might be influenced by features dominant in a corpus as a whole, but not frequent in a result set. The practical experience shows that these post-retrieval clustering algorithms outperform the conventional approach of creating in advance clusters for the whole collection of documents. Finally, one should be fully aware that clustering of even a very small portion of Web nowadays becomes very unfeasible mostly due to the highly dynamic changes in the underlying documents [282].

As far as the usage of a suffix tree is concerned, once it has been constructed, it can be efficiently used for finding the most important phrases in the underlying collection of documents. These phrases can be used for improving the other clustering algorithms that utilise the vector based representation of documents, which ignores the information about the order and the proximity of words. A straightforward way is to select the given number of most important phrases by using a suffix tree, and to add these phrases as additional features in the vector representation of documents. On the one side, the five hundred most important phrases are added to the vector representations that are used by the k-means algorithm. The obtained results have shown an increase in the quality of obtained clusters.

On the other side, insisting only on single word phrases is reducing the average precision that the STC algorithm achieves for more than 20%. This definitely proves that the information being presented in the order and the adjacency of words is important, and can contribute to the creation of clusters with better quality.

The novelty of Grouper is in combing a suffix tree structure, which is commonly used for the efficient representation of texts, with the hierarchical clustering algorithms. In that way the documents that share many common phrases will be most probably put in the same cluster, and at the same time an article can belong to many clusters whose phrases possesses. By doing that, it allows users to find the information they are looking for more easily, helps them realise faster that their queries are poorly formulated and reformulate them, and reduces the fraction of the queries on which users give up before reaching the desired information.

While the Grouper system uses the variation of single-link agglomerative hierarchical clustering for merging, it will be worth to explore how other known clustering techniques perform, i.e. try to combine suffix tree representation of documents not only with single-link agglomerative hierarchical clustering. In the case where the storage requirements of suffix trees become a bottleneck, the mentioned suffix array should be taken into account.

2.7.3. DisCover

The motivation for the development of the DisCover system [154] is the same as for Grouper [282] and CREDO [66] systems, i.e. the ranked list returned by most nowadays search engines is highly inefficient since the number of retrieved results might be in the thousands even for a typical query. An inherent ambiguity in interpreting the query terms, together with the used ranking that is not reflecting always the users' needs, is potentially making many of the top retrieved documents to be irrelevant to a user. The problems of ranked lists are in DisCover, similarly as in Grouper and CREDO systems, addressed through the organisation of Web search results into a hierarchy of topics and subtopics, which facilitates browsing and locating the results of interests [154]. While Grouper is achieving this organisation through the suffix tree clustering [282] and CREDO through the formal concept analysis [66], the DisCover system utilises the hierarchical monothetic clustering algorithm. The utilised algorithm, which is monothetic by reason of assigning documents based on the single feature, progressively identifies topics, being either single words or phrases. By doing that it tries to increase coverage maximally while maintaining the distinctiveness of already selected topics. The following paragraphs will first briefly present the architecture of the DisCover system, and subsequently discuss the advantages and disadvantages of the used clustering algorithm.

Similarly as in Grouper and CREDO systems, the application of DisCover starts with formulating the query that is forwarded to the used search engine, being currently Google. As soon as the search engine has produced results, being represented only by the title and the short snippet, they are parsed in order to identify the important terms. Together with always used stop word elimination and stemming or morphological generalisation steps, the additional heuristic rules will ignore any single word that is less than three characters in length. Furthermore, they will also eliminate terms that occur in less than two percent of the documents because they are unlikely to impact the formed hierarchy [154]. As far as the presentation of stemmed terms is concerned, it is also worth of attention that each

stem is replaced by the most frequently occurring original term. Therefore, the sometimes difficult understanding of the meaning of an underlying stem is overcome.

Even though the DisCover uses the general purpose tool, developed by IBM Watson Research Laboratory, for the extraction of noun phrases, the inbuilt ability of a suffix tree clustering algorithm to identify long phrases seems to provide better results. While the phrases that are found by DisCover are usually very short, a suffix tree, used in Grouper, is able to generate both much longer and more descriptive phrases [282]. Additionally, the terms are in DisCover selected without any weighting, which can be seen as the great disadvantage for finding out really important terms. This is especially the case where any comprehensive scheme, such as the one that is based on multiple term scoring functions from the CREDO system [66], is taken as an adequate example.

The cornerstone property of the used monothetic document clustering algorithm is in finding the optimal ordering or the permutation of the set of concepts [154]. The next selected concept is the one that increases the coverage maximally, being the number of documents that are included in the hierarchy, and that is maximally distinct from already selected concepts. More formally, for every so far not selected concept c_P, the coverage gain is computed by finding how many new documents, which are still not covered by the existing hierarchy, will be inserted into the hierarchy by reason of both having c_P and at the same time not having any of the already selected concepts. Every new document also contains novel concepts, which are still not selected, and thus it is possible to compute how many new concepts will be implicitly integrated into the hierarchy by reason of being present in selected new documents. The number of new concepts actually defines the measure of distinctiveness, with the reasoning behind that the greater number of novel concepts means that more novel documents are inserted and that the hierarchy becomes more distinctive. Unfortunately, the influence of coverage and distinctiveness in selecting new concepts is in DisCover currently set in the pure static manner, where it has been experimentally found that the coverage is 4 times more important than the distinctiveness. The natural improvement of the DisCover system can be concerned with integrating the learning capabilities that will manage to determine the optimal ration between coverage and distinctiveness. More importantly, such a ratio should be found based on the different types of queries, as well as user's preferences.

The performances of DisCover are tuned in order to obtain the linear complexity with respect to the total number of concepts, at the similar way as in Grouper. On the one hand, Grouper utilises the suffix tree clustering algorithm that has the linear complexity only due to the artificially limited number of clusters that will be taken into consideration for merging [282]. On the other hand, DisCover is limiting the number of concepts that will be initially included into the hierarchy. The incremental nature of its clustering algorithm enables that a user might progressively increase the number of concepts, and therefore it directly controls the compactness of the hierarchy.

The added value of the presented DisCover system is surely in using a unique scheme for selecting concepts. Most essentially, this scheme both takes care of the coverage and the distinctiveness of a resulting hierarchy. A made comparison with similar Grouper and CREDO systems has clearly shown the advantages and the disadvantages of this solution, which opens the opportunities for a new scheme. It can combine all these algorithms, and therefore can provide the best results.

2.8. Conclusion

Based on available scientific publications, thirty different systems have been presented from the view point of the utilised strategy for addressing information filtering tasks. The introduced algorithms range from the appealing evolution strategies and reinforcement learning to standarly exploited content-based and collaborative filtering techniques. Also mentioned are Bayesian classfier, exploited for the efficient learning of user preferences, web log mining which uses the data that users left behind on web servers, and clustering techniques, utilised for facilitating browsing through filtering results.

A necessity to combine collaborative filtering with content-based techniques has been clearly pointed out in Section 2.1. Even GroupLens, being mainly responsible for a glory of collaborative filtering, has had to admit that the integration of content-based strategies as filter-bots tends to provide better results. A dependence between these two techniques has been strengthen in Fab, which first uses content-based analysis to build profiles that are then utilised by collaborative filtering while searching for similar users. P-Tango has seen the combination of strategies as a weighted sum, whose weights are adapted in order to minimise past errors. The various distance measures have been investigated in Ringo, which unfortunately does not combine them although there is no clear winner. Content-based and collaborative strategies have together managed to provide better TV listings in the commercial PTV system.

By presenting WebWatcher, Section 2.2 has first noted that very various strategies for recommending links are possible. While some of them ignore user preferences, the others can even learn which hyperlinks suit better which needs. WebWatcher has finally shown that the simple combination of these strategies, based on majority vote technique, tends to provide superior performances. The contribution of WAIR has been found in exploiting various implicit indicators of interests, such as reading time, book-marking, scrolling and following hyperlinks. These indicators have been combined by three layer neural network that is trained to estimate relevance.

A great opportunity for integrating different filtering strategies has been illustrated in Section 2.3. The Amalthaea has for example hundreds of filtering agents, each being one individual that is fighting to survive in the evolution environment. Although the deployed strategies can be completely different, Amalthaea has used only content-based one that is exploiting cosine similarity measure. The similar observation holds for NewT, which has additionally demonstrated that the whole population evolves faster due to the application of a Baldwin effect. A sequential combination of strategies, where the first does filtering on headers and the second can correct spelling mistakes, has shown how PEA can handle the huge abundance of informal email messages.

The usage of Web logs, which are generally available in abundance, has been analysed in Section 2.4. A Footprint system has first demonstrated that really various visualisation schemes might be utilised while presenting where users have gone. These browsing paths have been exploited in ProfBuilder while finding similar users by collaborative filtering. Although ProfBuilder has also introduced content-based analysis that estimates relevance based on browsing paths, these two strategies have not been combined. The importance of the manual dictionary, which makes a feature space to be smaller, has been found to be very important in SiteHelper that uses heuristic coverage rules for modelling.

Section 2.5 has analysed the usage of naïve Bayesian classifier for user modelling. The comparison of different machine learning algorithms has been generously given while presenting Syskill & Webert system. It has been noted that there is not the best strategy among naïve Bayesian classifier, nearest neighbour algorithm, ID3, C4.5, and multi-layer back-propagation neural network. Syskill & Webert has also introduced two strategies for smoothing zero probabilities, named Laplace and ε correction, as well as a combination of them based on the information gain value for a particular word. A solution for the push functionality, by using the same naïve Bayesian classifier as Syskill & Webert, has been given as Do-I-Care agent. It might additionally provide collaboration among users, where unfortunately the similar users have to be manually found. NewsDude and Daily Learner have improved the performances of naïve Bayesian classifier by additionally integrating a nearest neighbour algorithm. A specified sequential combination, which first asks nearest neighbour algorithm, and in a case of its failure, naïve Bayesian classifier, has happened to better model both short and long term user preferences.

Section 2.6 has presented systems where content-based aspects play a dominant role. Three content based techniques, being based on Levenshtein distance on headers, on term frequency inverse document frequency scheme, or on citation structure, are combined in CiteSeer as weighted sum with fixed weights. A probabilistic selection of results has been used in Adaptive Web Site Agents while learning which strategy is preferred by whom. A selection probability has been there computed as the combination between the predicted relevance of a result and the reliability of a responsible strategy. The high popularity of Letizia has shown that simple heuristic rules can be used to effectively combine multiple implicit indicators. Let's Browse has introduced different schemes for combining profiles to support collaborative browsing. A query expansion by adding new words, which have been found in pre-computed trigger pair models, has been stressed as an important aspect for WebMate. A novel content-based algorithm, which uses a separately found relevance of each document segment while estimating the overall relevance, has been introduced by the Insyder system. CREDO has finally pointed out on different term weighting schemes while performing post retrieval clustering based on the formal concept analysis.

Increasing the users' ability to perceive more results has been addressed in Section 2.7 as a post retrieval clustering. Scatter/Gather has introduced clustering algorithms, named Buckshot and Fractionation. The former is a k-means appraoch where initial centroids are found by a hierarchical clustering on a sample. The latter is an agglomerative hierarchical algorithm which merges clusters only locally. Different techniques for extracting feature words, based on suffix tree clustering, n-grams, and frequent sets, have been additionally compared in Grouper. DisCover has demonstrated that coverage and diversity should be taken into consideration while ranking the formed clusters.

Common to almost every analysed system is a conclusion that there is no universally best applicable filtering strategy, and that usually the combination of different techniques tends to outperform any single solution. The given discussions have thus tried to carefully estimate the advantages and disadvantages of existing methods for combining various strategies. Such investigation has been done in order to motivate the cornerstone of this thesis that mainly aims to coordinate the various strategies towards optimally exploiting available system resources while processing the received filtering requests.

Chapter 3
Problem Description

The time when the needed data were searched for in libraries is probably over. Internet has not only made the publishing process much easier since almost everybody can put its documents to be online available, but also has dramatically increased the accessibility of the information. A dilemma whether the searched data really exists in a particular library is no more actual question, whereas many new challenges are born. On the one hand, the huge abundance of online accessible data has made a process of locating the right Internet source to be much harder. Even though modern search engines are more powerful than ever before, there is the immense number of web pages that are still waiting to be indexed. Many dedicated online databases are also not reachable by the automatic indexing robots, and consequently their underlying content remains a mystery for everybody that solely utilises search engines. On the other hand, there is an increased necessity to have the right information at the right time in the nowadays commercial society. The wonderful project proposal offer is simply worth nothing when being discovered too late by search engine robots. Users are thus forced not only to utilise search engines, but also to consult other known Internet sources, such as Cordis online database when the relevant project offers are searched for.

The unavoidable difficulty with a manual usage of different online databases naturally exists mainly due to the necessity to learn the query language separately for each of them. There are thousands of databases available, and it is practically impossible to become an experienced user for even the tiny fraction of them. Even though most of these online databases have standard query interface that usually everybody can use without any pre-training, there are many sophisticated options whose usage requires a previous education. As these advanced functions are often necessary for getting the best that a corresponding online database can offer, typical users have a dilemma whether to invest some learning efforts to obtain better results. In the case where the first results that are obtained without advanced options are not so good, but seem to be appealing, users have either to try to create more sophisticated queries or to look for the another online source. Such advanced queries will maybe yield much better results, but in order to be sure in that, a user has to read about the available additional options, as well as to analyse the existing examples of sophisticated queries. It can unfortunately happen that in spite of spending several hours on figuring out how the found online database should be best used, the results are still not excellent, and a user can have a feeling that its valuable time was wasted.

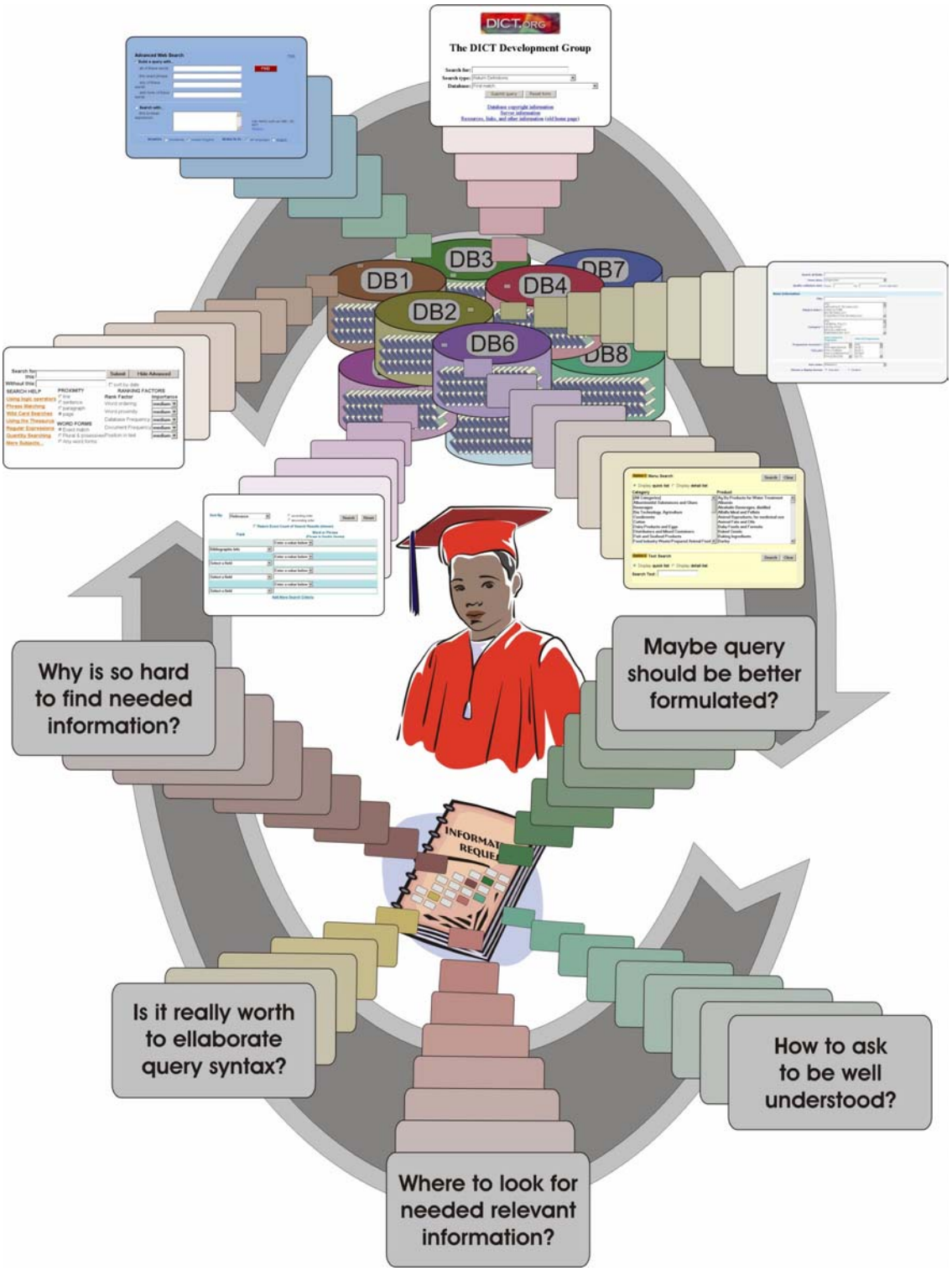

Figure 3.1: Modern Information Retrieval Challenges

Only some of the challenges that exist due to numerous online databases are illustrated on Figure 3.1. Even a presented small scale case with only 9 available online databases seems to be quite illustrative when the creation of the well formed information request is concerned. Every database has its own query interface, assuming a particular associated

language, whose syntax has to be elaborated in order to build information requests. The most typical questions and dilemmas, which a user certainly has in mind while searching for needed data, are given on Figure 3.1 as building blocks inside the request. Figure 3.1 finally illustrates that a user is inside a circle where all over again requests are sent to the same or to the different databases. Sometimes a user decides to elaborate the advanced query syntax for a particular online source, and to create more sophisticated queries for it. In yet another case, a user gives up from a given database, and concentrates its attention on finding the better one.

Without any intelligent support users are forced to learn different query languages and to select databases, which will be checked whether they contain needed data, either based on their own past experience or maybe by following the suggestions of their friends. As such a situation seems to be far away from being promising, the cornerstone idea of this thesis is to help users successfully cope with these information retrieval problems. This is going to be accomplished by introducing both manager and filtering agents as direct and indirect assistants that will provide the necessary intelligent support to a user. Manager and filtering agents will build sophisticated intermediate layers between a user and the sources of information, which will free a user of a difficult thinking where the necessary data is located and how it can be retrieved. In the rest of this thesis, the introduced layers will be referred as cooperation, exploration and coordination tasks. They are responsible for finding the right databases to be queried, for autonomously refining queries obtained from users, and for deciding which strategy is the best for a given source, respectively.

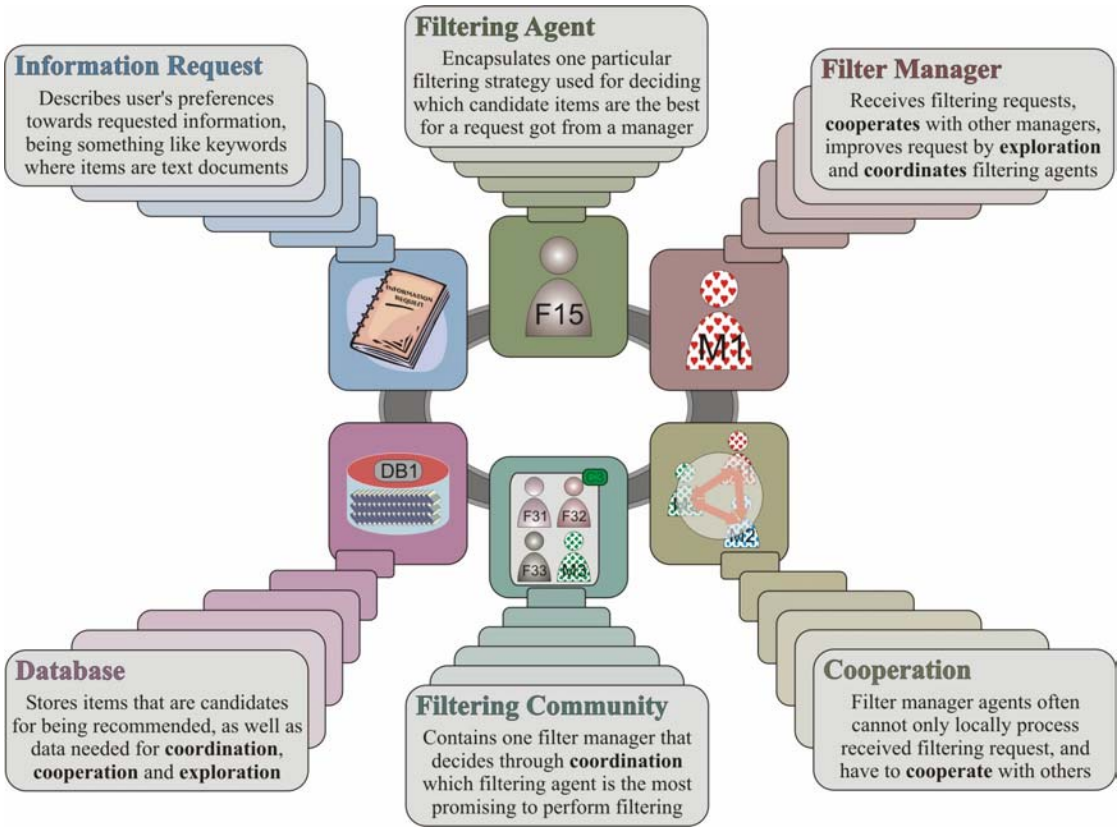

Figure 3.2: Meaning of the symbols that will be used in this thesis

Before presenting a proposed filtering framework that will help recognise which novel challenges are hidden behind cooperation, exploration and coordination, main symbols are introduced on Figure 3.2. Besides symbols for cooperation and a group of a manager and several filtering agents that is named filtering community, all other symbols are more or less typical for agent-based information systems. A circle in a symbol for cooperation illustrates that no more user has to separately query every known database, but managers are cooperating to find where the most promising data is located. Filtering community, or shortly only community, represents a playing ground for coordination, being the process where a manager decides which filtering agent becomes responsible for an actual request.

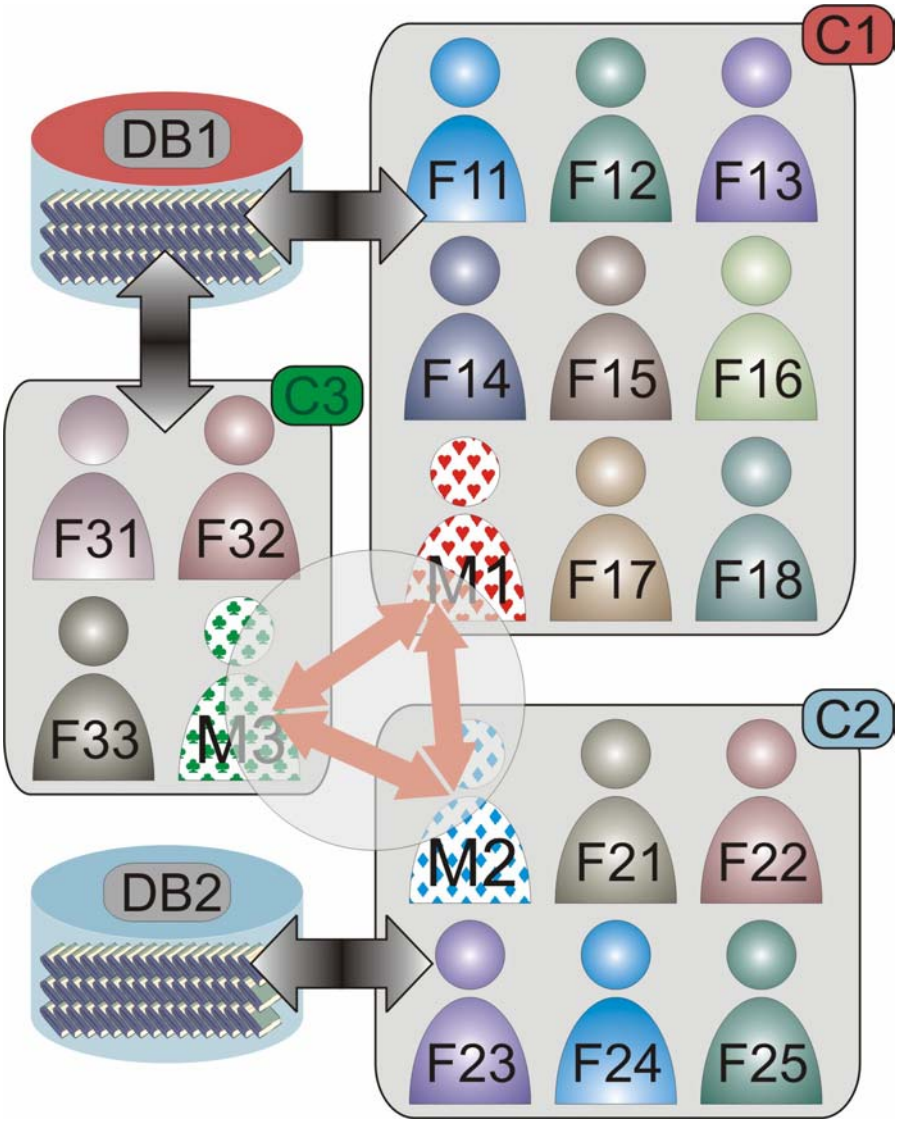

Figure 3.3: Three filtering communities are installed around two distributed databases

Figure 3.3 presents a filtering framework, having three communities that are installed around two databases. This scenario, in spite of its simplicity, can be used to practically introduce cooperation, exploration and coordination activities. The cooperation starts as

soon as one of filter managers has information request that its community cannot locally process for example by reason of not having the access to the necessary database. During cooperation the most promising managers, which either have the needed information or are known as reliable cooperation partners, are searched for. The expected final outcome of this cooperation activity is to autonomously figure out where to look for the requested information. Only some of complex problems that cooperative managers have to handle are concerned with estimating how promising each known manager can be for processing the received request, deciding how many other managers to ask, composing a final result set based on suggestions obtained from the asked managers, and learning from the user's feedback about the real capabilities of the responsible managers and their communities. These and many other cooperation challenges will be deeply analysed in Section 3.1.

After one manager decides to locally process the particular request, the next activity is concerned with trying to improve the received job through exploration. The basic idea is in making available the expertise of many other humans whose requests were processed in the past. As many useful data, such as additional relevant keywords, can be found in these old requests, exploration tries to incorporate these data into the actual request, and therefore increase its chances to find the relevant information. The underlying exploration challenges, which will be the main topic in Section 3.2, are related first with searching for the usable old requests that are similar to the actual one, then in recognising which their elements might be useful, and finally in intelligently reusing them in order to make better the actual request.

While the only involved actors in cooperation and exploration are manager agents, in the last piece of the introduced framework, being named as coordination, both managers and filtering agents take part. Since every manager maintains in its community multiple filtering agents with highly different capabilities, the coordination role is to decide which filtering agent can most successfully respond to a received request. The great challenges behind are numerous, and are not only illustrated in Section 3.3, but also make the basic motivation for this thesis. The worth of mentioning are especially coordination attempts to optimally use available system resources, to somehow take into account the properties of a received request while selecting a responsible filtering agent, to balance exploration and exploitation by sometimes giving a chance to not so good filtering agents to prove themselves and improve their qualifications, to try to recover after the selected filtering agent has failed to deliver expected results, and finally to utilise both the response time measurements and the user feedback values to efficiently learn about the real capabilities of filtering agents.

Figure 3.4 illustrates one of possible scenarios, while processing a given information request, including cooperation, exploration, and coordination activities. The information request can be initially received by any manager, and on Figure 3.4 that is a manager M_2. Since M_2 is not able, locally inside its community C_2, to successfully fulfil this request, for example because database DB_2 does not have necessary underlying data, M_2 decides to cooperate with M_1, for example by reason of having a sold experience when sending similar request to M_1 in the past. A request is thus forwarded to M_1, and afterwards M_1 decides to accept the request and offer its help to M_2. After accepting the request, M_1 is trying to improve it by exploring which old requests, processed locally inside community

C_1, can be used for its improvement. Finally, M_1 selects through coordination process that filtering agent F_{13} should be responsible to deliver expected results. Such a selection can be based on the experience that M_1 had with F_{13} in past situations where the similar load of system resources was measured and practically the same requests were processed. The finally chosen F_{13} will be responsible for doing filtering and finding which objects from database DB_1 are the most proper of being returned as recommendations. Although it is not illustrated on Figure 3.4, found results will be returned via M_1, whose filtering agent has processed a request, to M_2 that is finally responsible for deciding whether the results are good enough for a user.

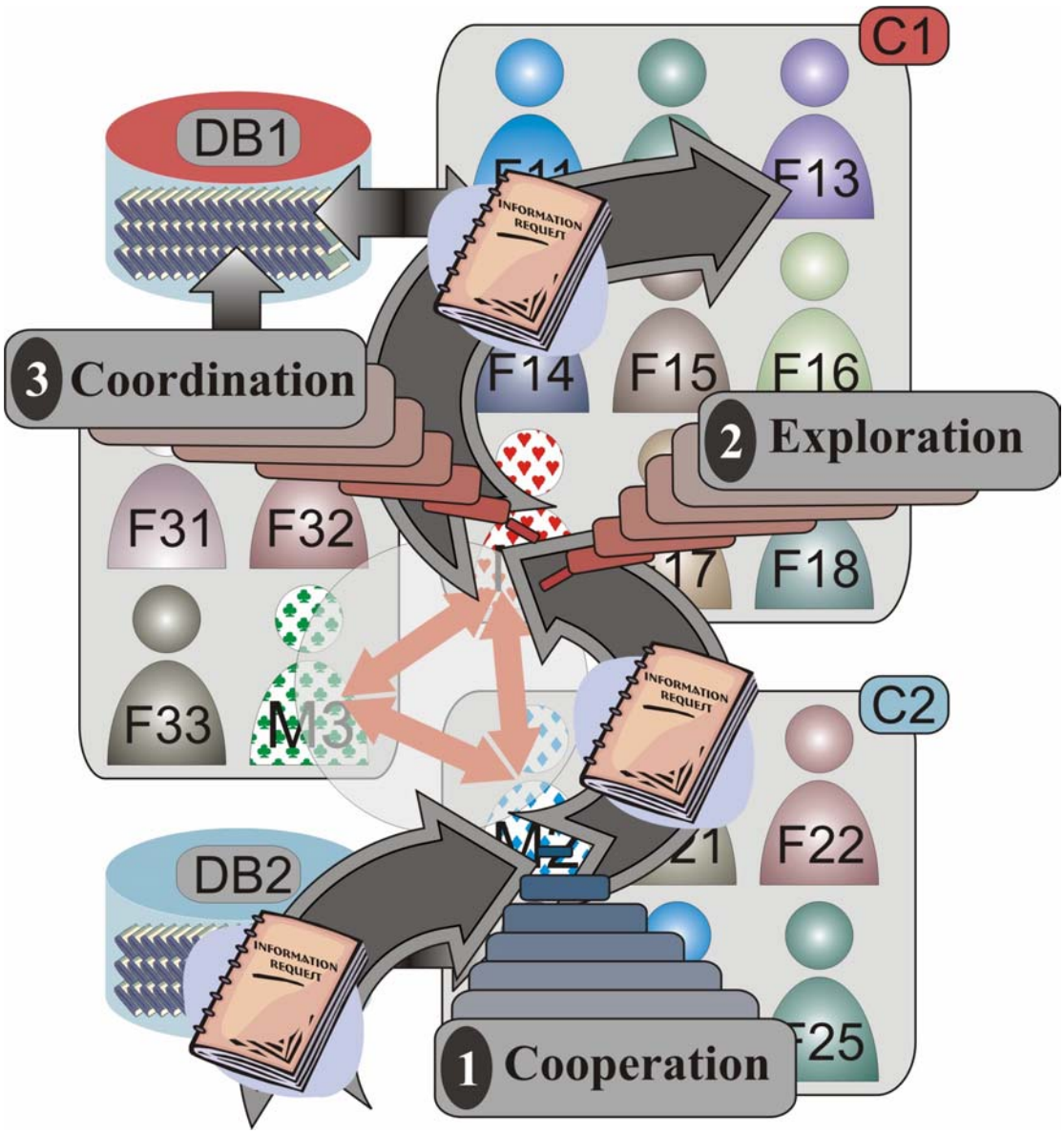

Figure 3.4: Cooperation, Exploration and Coordination challenges while processing request

The remainder of this section will be first focused on illustrating main cooperation challenges through much more detailed scenarios. After also separately discussing about critical exploration problems that have to be addressed, the section is finished with the comprehensive presentation of coordination issues, whose solutions represent the main contribution of this thesis.

3.1. Cooperation Challenges

The unavoidable consequence of the nowadays overall information overload is that an open problem becomes answering where the relevant information is deployed. Internet users are typically forced to manually make decisions about the most promising online sources for retrieving the needed information, being usually a time consuming activity. The problem of locating useful sources is even more severe by reason of more or less frequent changes in the content of each source. Consequently, users will often search for the needed information at the wrong places, they will waste plenty of their valuable time, and maybe even they will not find what they are looking for. The remainder of this sub-section will first offer on example, which will illustrate without deeply digging into the matter, how intelligent agents can help users locate and retrieve only relevant information. Although agents provide necessary means for these hard information retrieval problems, they introduce many novel challenges. These challenges will be therefore analysed in one scenario, being separated into updating, estimating, dispatching, composing and adapting steps, which any comprehensive cooperation mechanisms have to address.

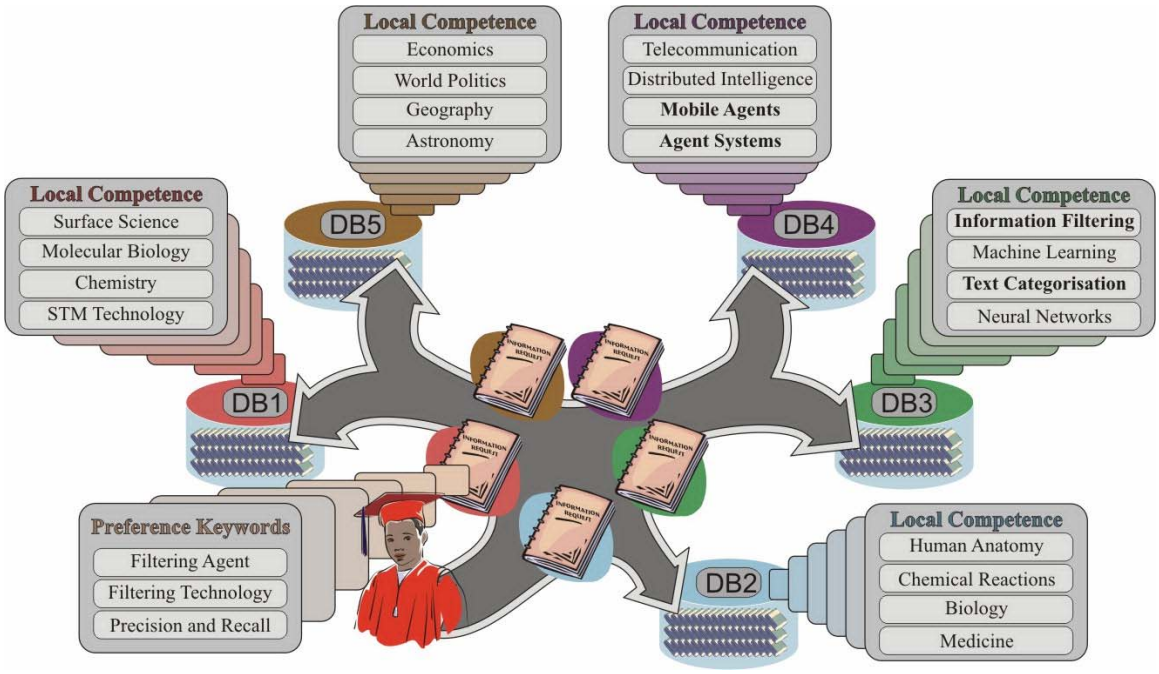

Figure 3.5: Without agents user has to send the adapted versions of its information request separately to every known sources for which there is a belief that can provide useful results

Example 3.1: The purpose of an ongoing example is to give, on a relatively high level, a situation where every online database has its own competence regarding the underlying

data that are locally stored. Although there are nowadays millions of such sources that are accessible on Internet, even a very simplified case with only five online databases will be enough to present main difficulties that users have somehow to overcome to find needed information. Without any intelligent support, users are forced to send their information requests to every known database, and naturally to be disappointed many times when the results with the low quality are obtained. The yet another, even more serious, difficulty, while directly communicating with the numerous distributed online sources, is contained in a necessity to manually adapt information request to be understandable by a receiving party. In other words, each database has its own language that can be understood, and the different versions of the semantically identical information request are necessary. These requests, being separately sent to every online database, are on Figure 3.5 illustrated by a different background colouring of the practically the same graphical representation of the information request.

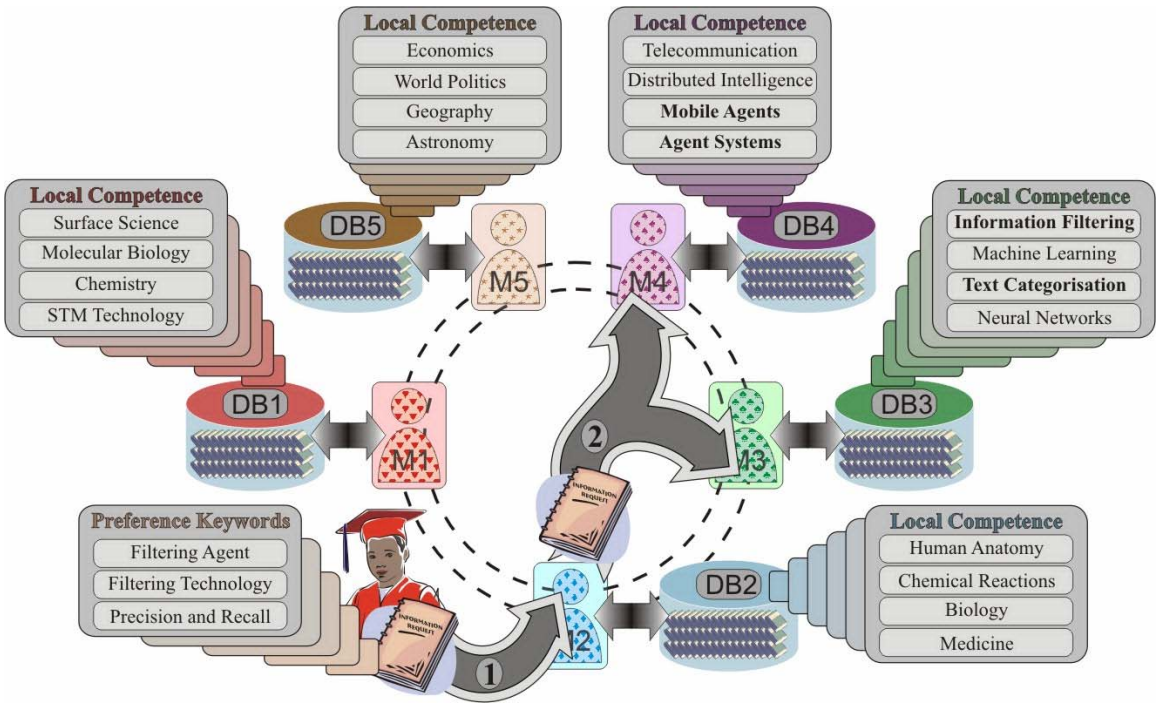

Figure 3.6: The manager agents, being installed around online databases, are autonomously finding where the most promising local competence exists for processing the actual request

To overcome the necessity to manually reformulate the information request for every online database that should be asked to find results, manager agents are installed around them, as given on Figure 3.6. The roles of the managers are concerned with thinking how the assigned database should be queried, as well as with being aware of the existence of each other. The former role is freeing a user from the obligation to learn the language that should be utilised for creating the information request for a particular database. The latter role enables sending the request to any manager that will become responsible for further finding other managers, being hopefully better suited for the actual request. On Figure 3.6 a user sends its request about filtering with agents to manager M_2 that unfortunately has completely other competence, relating biology and medicine. Without manager agents

probably no results will be got, whereas installed managers will enable the propagation of this request to M_3 and M_4 that have a necessary competence for processing the request about filtering with agents. Manager M_3 happens to be a specialist for text categorisation and information filtering, and M_4 has the necessary background knowledge about agents. Managers M_3 and M_4 can therefore find results that can fulfil the expectation of a user. The challenges of finding which managers are best suited in a particular situation are out of the scope of this high level example, and will be analysed in the specially designed scenario.

The results that a user is expecting, and that are found by M_3 and M_4, are returned via the initially triggered M_2, as on Figure 3.7. Manager M_2 is therefore responsible for the final selection of results, which usually assumes the elimination of results that are not worth of sending back to a user. These delicate decisions about the selection of the results cannot be explained in this simple example by reason of being based on the knowledge that managers have about each other, and will wait until the corresponding steps of the following scenario are not presented.

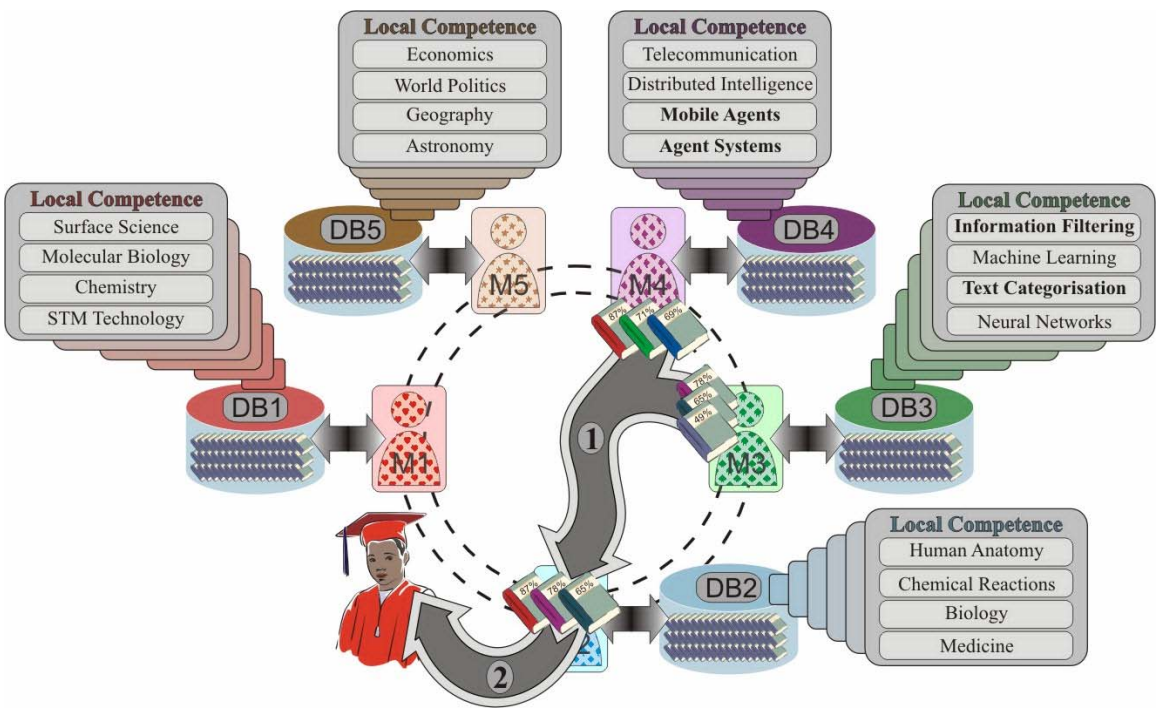

Figure 3.7: User is getting the expected recommendations by collecting the found results and selecting only the most valuable ones that are worth of being returned

To make more understandable figures that cover each and every step in a scenario, the meaning of novel symbols is introduced on Figure 3.8. The most often used symbol is the one that correspond to a description of a particular community, and that is stored by other managers to enable dispatching the request to the right parties. This description can be a good one when representing well the content of the corresponding community. By reason of the frequent changes of the content, the community description might become obsolete

or bad. As being clearly noticeable on Figure 3.8, a goodness of a particular description is illustrated by the different level of noise on its graphical representation. While a graphical representation of a good community description is quite clear, a bad one has a significant amount of noise. Figure 3.8 also gives symbols that will be utilised for marking the found filtering requests or recommendations, as well as for showing that a user is providing the feedback for a particular recommendation.

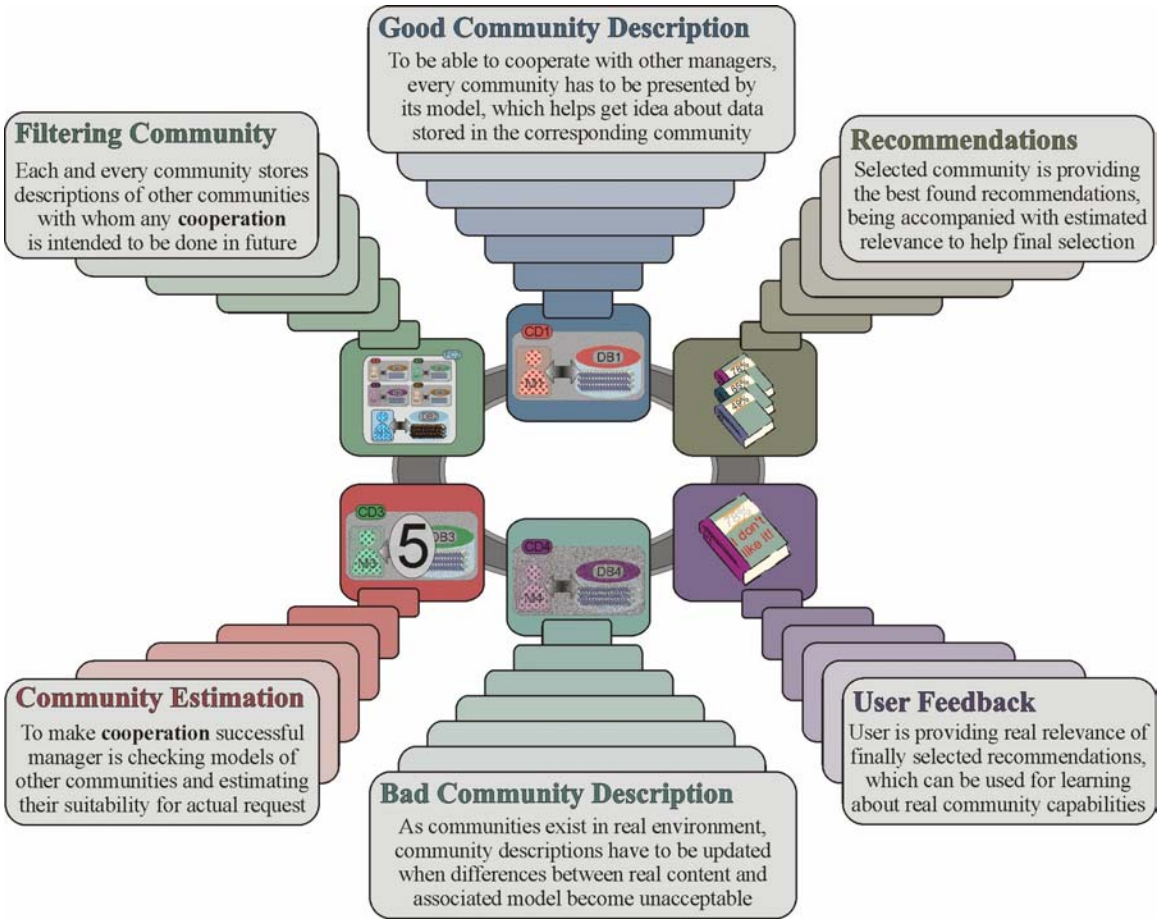

Figure 3.8: Meaning of the symbols that will be used through cooperation scenario

Because one figure is worth as thousand words, each step will be illustrated through a separate figure. Figure 3.9 gives a playing ground for scenario, where this playing ground is composed of five different online sources or databases DB_i, $i \in \{1,...,5\}$, around which distributed filtering communities FC_i are installed. Each and every community has one manager (M_i) agent that is responsible for every single cooperation activity. To enables cooperation, manager M_i is representing other communities by their descriptions CD_j, $i \neq j$, which illustrate their underlying content. For example M_3 has CD_1, CD_2, CD_4 and CD_5 as the descriptions of corresponding communities with whom any cooperation is possible.

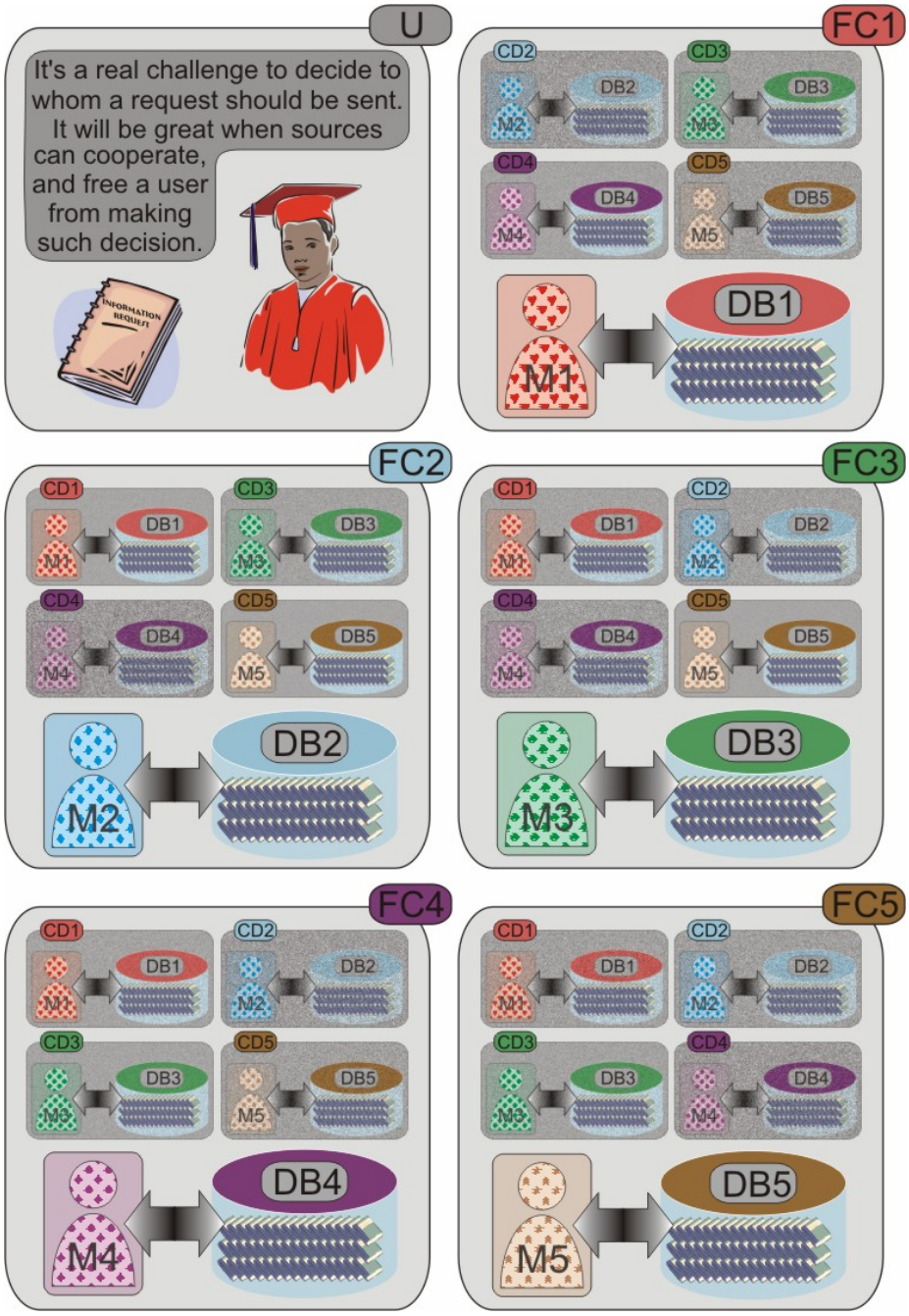

Figure 3.9: Cooperation playing ground

3.1.1. Updating

The scenario begins by letting a user to send a request, which specifies its information needs, to any manager that is known. The creation of an initial request and providing a feedback about the relevance of found results are the only activities that a user has to take care, whereas everything else is inside the responsibility of a manager that has received a request. The first thing, which by a user chosen manager has to do, is to check what it knows about other communities or managers, because these other communities maybe have more relevant data that can better satisfy the imposed request. While checking what

is already known about other communities, it might happen that the data about some communities are obsolete, and that these data have to be updated. In order to achieve an acceptable efficiency, one needs special heuristics, for determining when a particular update activity has to be performed. These heuristics have to handle a trade-off between having a perfect knowledge about all other communities and doing a reasonable number of expensive updates. These updates can be usually managed only by asking a particular manager to send its new community content description, which can impose a significant network load.

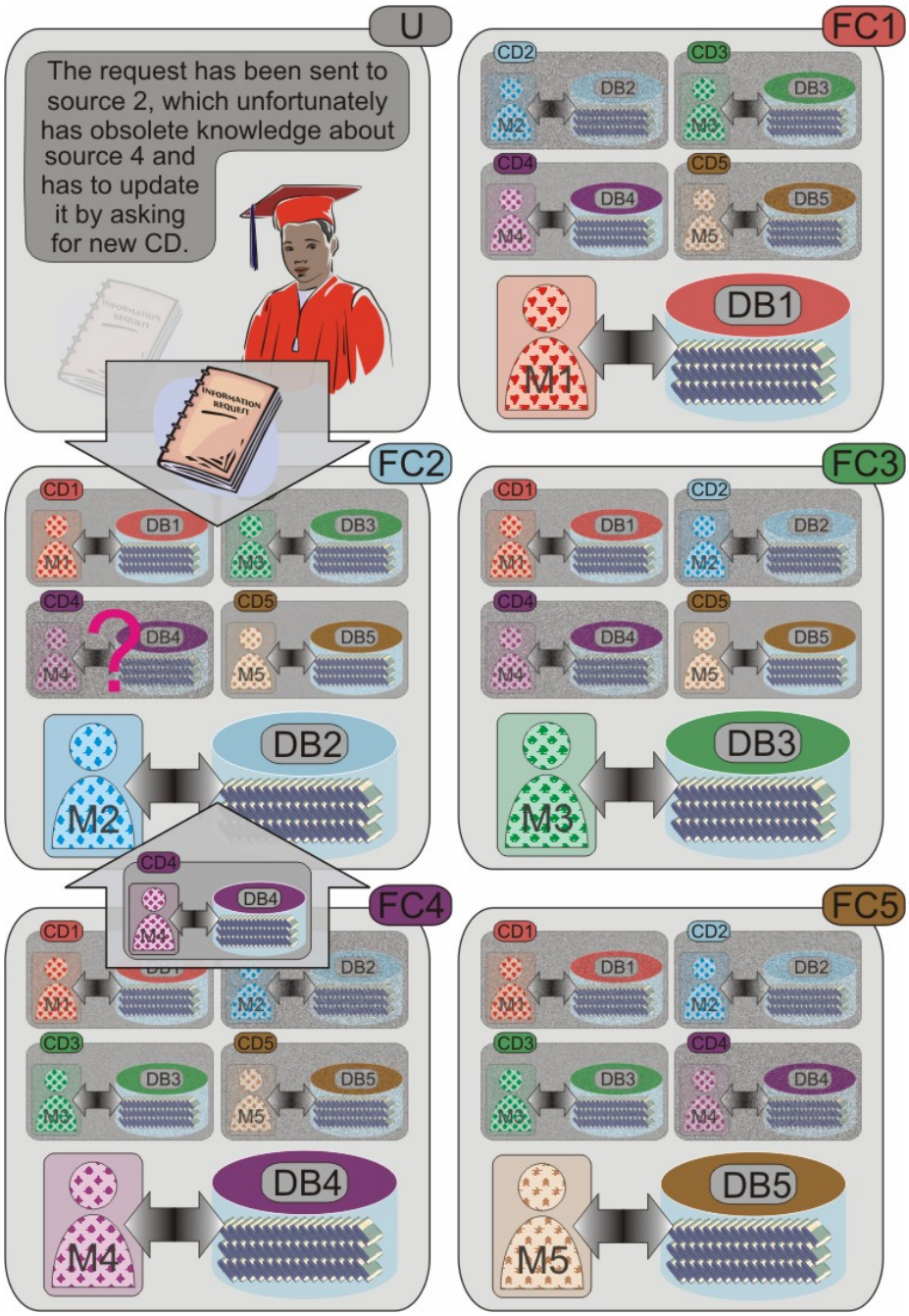

Figure 3.10: Updating obsolete descriptions

On Figure 3.10 a user has selected manager M_2 to receive its information request and to be responsible for providing guaranties about the quality of the delivered results. By examining the actuality of descriptions that M_2 has about other communities, M_2 founds that CD_4 is obsolete. While the pure authenticity of CD_4 inside FC_2 is on Figure 3.10 illustrated by a high level of noise in its graphical representation that corresponds to the bad community description, the actual suspiciousness of M_2 regarding CD_4 is noted by a question mark. M_2 is finally deciding to discard the current version of CD_4, to ask a responsible manager M_4 to send its new description, and to overcome the obsoleteness of CD_4.

3.1.2. Estimating

After reaching a state of having the acceptable knowledge about other communities, the estimation of how good every community can be for the particular request, is needed (Figure 3.11). This estimation naturally assumes the additional usage of the available past experience, which shows how reliable each source has been in providing really relevant results. The idea is to somehow make the balance between what a particular community says about itself and the lessons that have been learnt through the previous cooperation with it. The dilemma, which has to be addressed through the estimation step, is concerned with deciding either to use for the particular request currently better suited communities or to rely more on the communities that have better satisfied the users' needs in the past (Figure 3.12).

On Figure 3.11 a manager M_2 inside community FC_2 has analysed CD_1, CD_3, CD_4 and CD_5 in order to create estimations, being represented as numbers from one to five, where larger number means better estimation. On the one hand, high numbers, being assigned to CD_3 and CD_4, mean that the corresponding FC_3 and FC_4 either have very promising content for resolving the actual request or are known by manager M_2 as very reliable in the past. The usage of FC_3 and FC_4 might be therefore a solid decision while resolving the actual request. On the other hand, low estimations for CD_1 and CD_5 show that FC_1 and FC_5 either do not have the necessary local competence or have failed often while processing the past requests. The selection of FC_1 and FC_5 thus seems to be not very appealing in a particular situation, and consequently low estimations are assigned to them.

The challenges of combining both reliability and current content suitability are given on Figure 3.12, where the estimations are computed for two very different communities. On the one hand (Figure 3.12a), community FC_4 is known to locally contain data that do not match exactly what is specified in the received information request. At the same time, the strength of FC_4, lies in its great reliability in providing results, being liked by users in past. On the other hand (Figure 3.12b) community FC_3 has almost reverse capabilities, as being known by its not so great reliability in the past, as well as possessing much better underlying content than FC_4. Since gaining the user confidence by delivering

exactly what has been requested is found to be quite important, a little better estimation is assigned to FC_3 (FC_3 is estimated by 5, and FC_4 by 4). Such a decision gives a little advantage to a community FC_3, having the right underlying content, and therefore better chances to find expected results. The formal reasoning, being behind created estimations, will be one of main topics in Chapter 4.

Figure 3.11: Estimation of how each community is promising

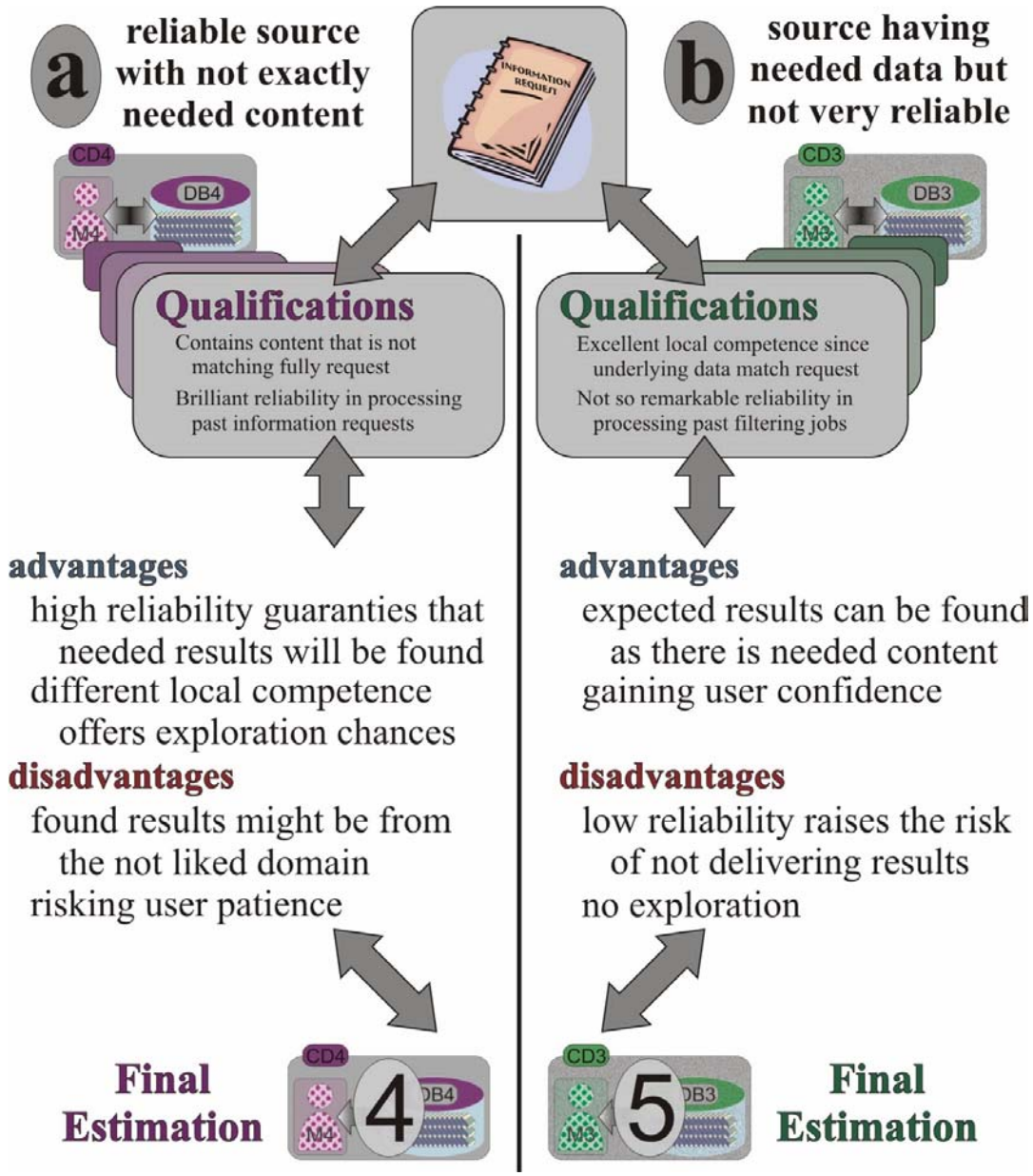

Figure 3.12: Estimating how each community or source can be used for a particular request requires that the suitability of available content is combined with the past experience with that community

3.1.3. Dispatching

By using the formed estimations about how promising are communities, a decision to dispatch request or job to some of them should be made. But, not all of jobs are the same, and not for all of them the same coverage concerning the found estimations exists. On the one hand, for some jobs, many communities can provide quite good results. On the other hand, very specific jobs can be well processed only by the specialised communities, and asking others is the pure waste of resources. A decision, how many communities should contribute, is hopefully connected with how good are estimations. It is maybe reasonable not to dispatch a job to a community with a bad estimation, but it is necessary to ensure that every job will be processed at least by somebody.

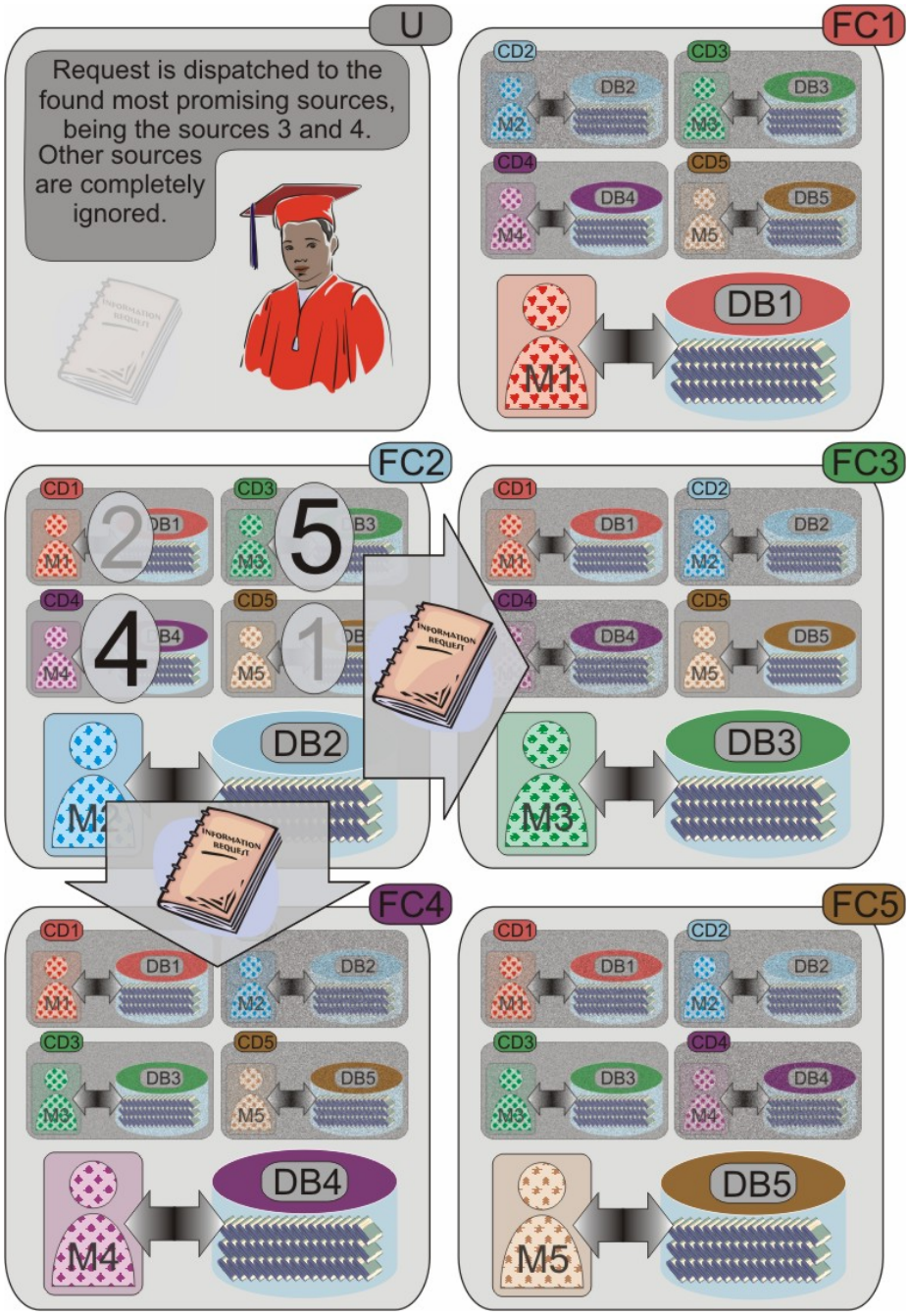

Figure 3.13: Information request is dispatched to the found most promising filtering communities

On Figure 3.13 a decision to dispatch the received request to communities FC_3 and FC_4 has been made because the previously computed estimations for both CD_3 and CD_4 were noticeably better than those corresponding to FC_1 and FC_5. Communities FC_1 and FC_5 will not therefore give any contribution, being reasonable because of both their low chances to provide the high quality results, as well as the existence of more promising and already selected FC_3 and FC_4.

3.1.4. Composing

The last piece of a puzzle, being known as finding the needed information through a distributed filtering, is mainly concerned with putting the found results together. In this composing activity, being performed by the manager that was initially chosen by a user, not only the quality of results but also the community successfulness in the past should be taken into consideration. The main problem is to find out how to compare following two results. The one was found by not so reliable community, but for that result a responsible community says that it is exactly what a user is expecting. The yet other is found by the community that is known as a very successful one, but at the same time that community is declaring that this particular result has weak chances to satisfy the user's information needs. In other words, the question is how to combine by a community predicted result relevance with the reliability of that particular community. The possible brainstorming, being behind an answer to this question, is given on Figure 3.14 through a comparison of two results found by two different communities. While the first result (Figure 3.14a) has relevance of 71% and is found by FC_4 whose suitability for a given request is estimated as 4, the second result (Figure 3.14b) is not so good, having the relevance of 65%, but is found by a better community FC_3 with 5 as the estimation.

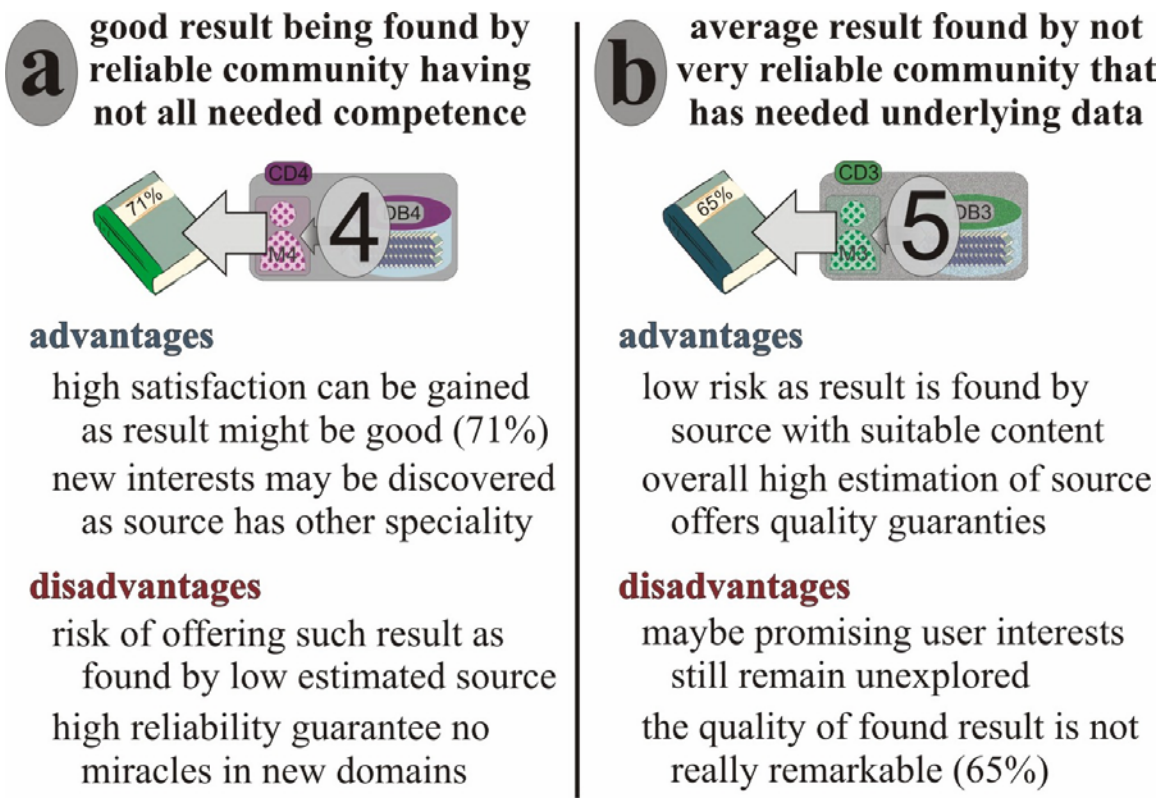

Figure 3.14: Deciding which result is better to be returned to a user requires that both result relevance and the estimation of the suitability of responsible community are taken into consideration

On Figure 3.15 community FC_3 has found 3 results for which the locally estimated relevance values are 78%, 65% and 49%. FC_4 is similarly offering also three results with

the assigned relevance values of 87%, 71% and 69%. The responsible M_2 is collecting all these results, and deciding to return to a user the ones whose relevance values are 87%, 78% and 65%. The results with relevance 71% and 69%, found by FC_4, are estimated to be less worth than the one with the relevance 65% from FC_3. Such a decision illustrates a reasoning that better initial estimation of FC_3 (FC_3 is estimated with 5 and FC_4 with 4 on Figure 3.11) gives advantage to that community, and also to lower scoring results found by FC_3.

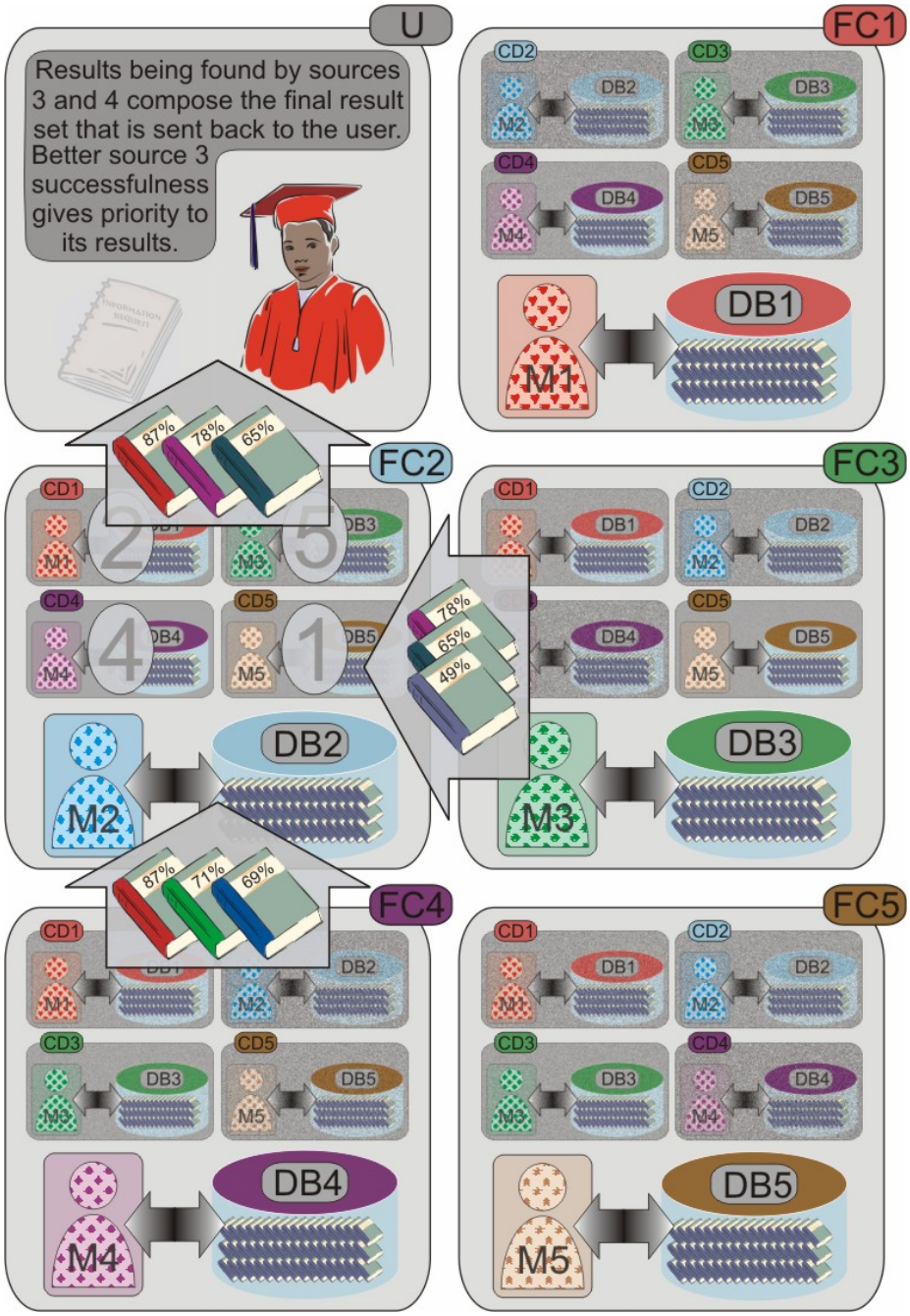

Figure 3.15: Composing a final result set by deciding which results are worth to be returned to a user

3.1.5. Adapting

As soon as a user feedback about the real result relevance is received, the measure of a successfulness of a community in finding the accurate results can be adapted. An ultimate goal of these adaptation activities, is establishing a much more realistic picture about the potentials of available communities. The final expected effect is that installed managers are hopefully going to learn to even better do cooperation among communities for future filtering jobs.

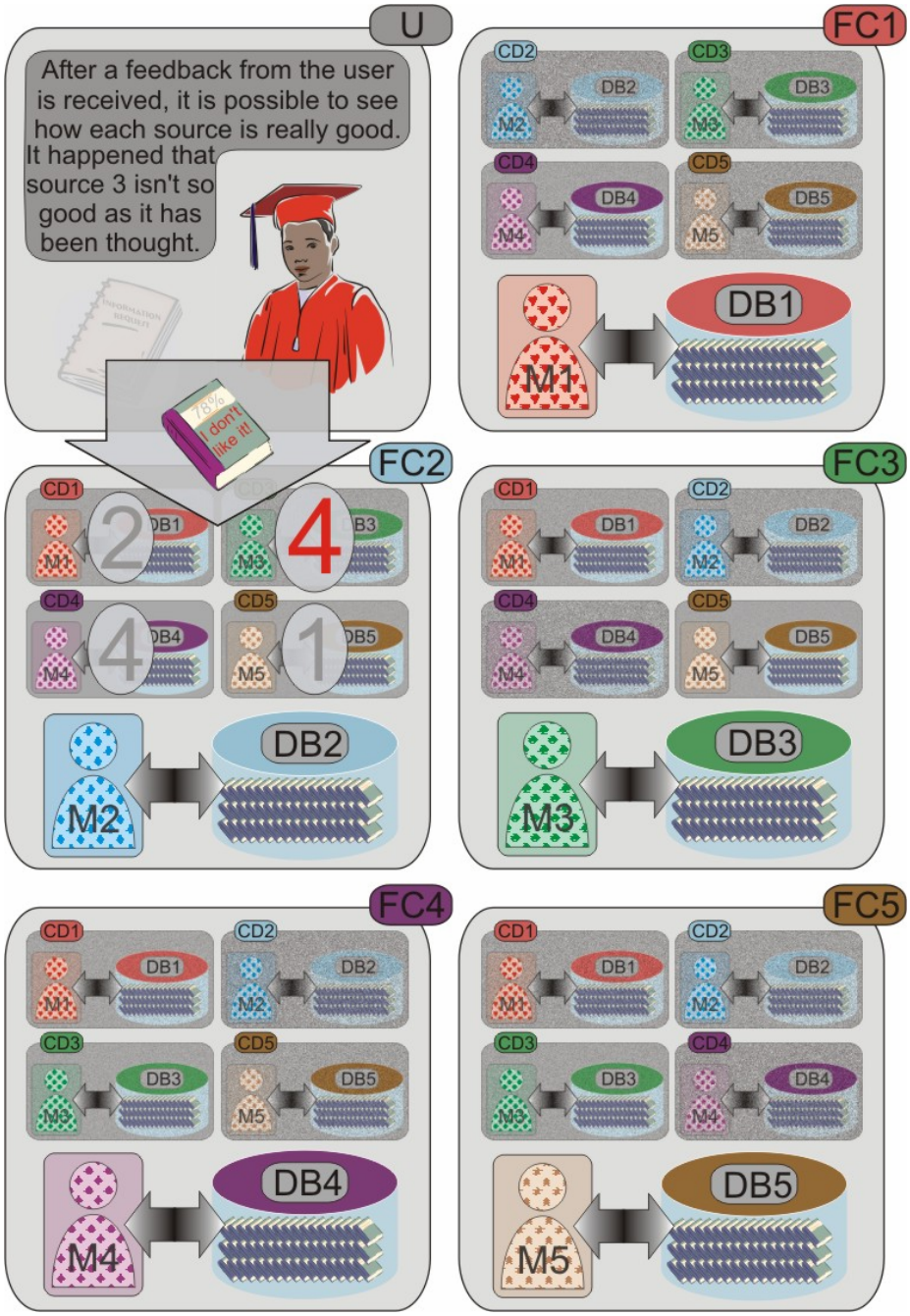

Figure 3.16: Adaptation of knowledge that FC_2 has about FC_3 after receiving a user feedback

On Figure 3.16 a user is sending a negative feedback for a result with the initially estimated relevance of 78%. Since this result has been originally offered by community FC_3, manager M_2 is lowering its reputation about FC_3. That adaptation of knowledge about FC_3 is illustrated by decreasing the estimation value for FC_3 from 5 to 4, with the symbolic meaning that in the future for the same request the applicability of FC_3 will not be anymore estimated with 5 but with 4.

3.2. Exploration Challenges

A human intelligence has produced a huge number of wonderful creations, which are unfortunately sometimes deeply hidden in numerous requests being sent to every filtering engine. The real challenge thus becomes finding ways how these intelligent creations can be first successfully recognised in order to be separated from the tons of irrelevant data that make many problems on the really beginning of exploration. The logical step further is to try to effectively exploit the discovered creations in improving the quality of future recommendations by integrating them into the actual request, which will become able to intelligently explore potentially relevant information areas. To illustrate these exploration challenges through one scenario, the novel symbols are first introduced on Figure 3.17. A three step scenario, where the usable old requests are first searched for, then mined, and finally utilised for exploration, will be afterwards given in separate sub-sections.

Figure 3.17 gives two novel symbols that are introduced for marking both information or filtering request in general and the request that is found to be useful for the exploration of an actual one. As far as the symbols for filtering requests are concerned, there are differences between graphical representations both in their shape and in their filling colour, where the used analogy says that more similar requests have more similar shapes and filling colours. The differentiation between requests both by their shape and their colour is intentionally done to illustrate the challenge of comparing requests, having hundreds of attributes that have somehow to be taken into consideration while deciding about the similarity of any two requests. These comparing challenges will be more deeply discussed in Section 3.2.1, where the similar old requests are searched for.

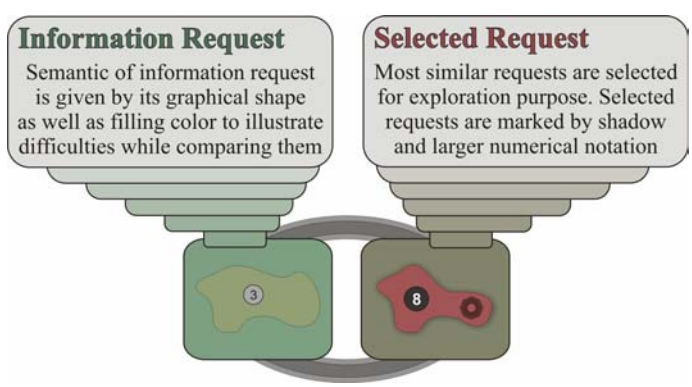

Figure 3.17: Meaning of novel symbols that will be used through exploration scenario

The high level idea about exploration is presented on Figure 3.18, which shows the step-by-step transformation of the actual request under the influence of available old

requests. The actual request before and after exploration is not fully the same, since some of the elements from the old requests are added. It still remains a mystery how the known old requests slightly change the actual request, hopefully in a direction of being able to better satisfy the user's information needs. This mystery will be hopefully partially uncovered in the following scenario, being clearly separated into searching for usable old requests, using similar requests, and adapting actual request steps.

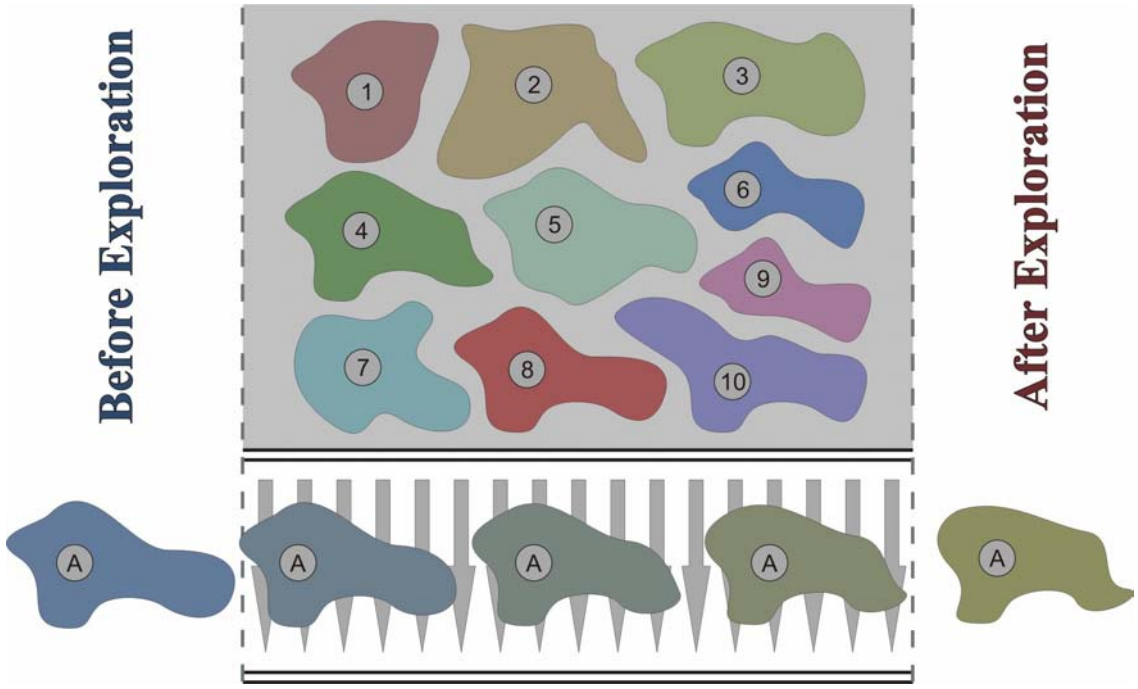

Figure 3.18: Illustrates the transformation of the actual request under the influence of old requests

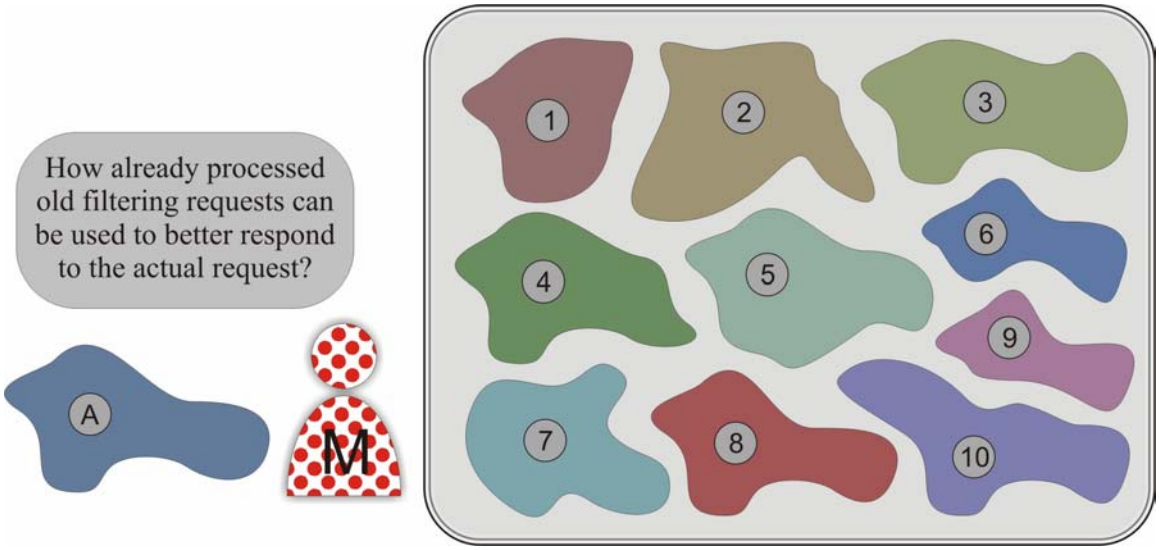

Figure 3.19: Manager M can hopefully better process the actual filtering request A, by using the knowledge that is encapsulated in ten available old requests.

The playing ground for the scenario is made from ten old filtering requests that one manager can use to improve a received actual request (Figure 3.19). Since real managers have processed not only ten but thousands of requests, this small scale scenario cannot be seen as an illustration of challenges that exist by reason of a huge abundance of available requests. In spite of that, it will be possible to give almost all other exploration problems, being concerned with the necessity to compare requests and afterwards to mine important aspects from the most similar ones.

3.2.1. Searching for Usable Old Requests

After a filtering request has been received, instead of responding in a straight forward way by proving the best matching results, the filtering engine will first do searching for successful old requests that are similar to the actual one (Figure 3.20). Reasoning behind this idea is based on an assumption that the important aspects in the similar old requests probably made them successful and represent knowledge that can be used to improve the actual request.

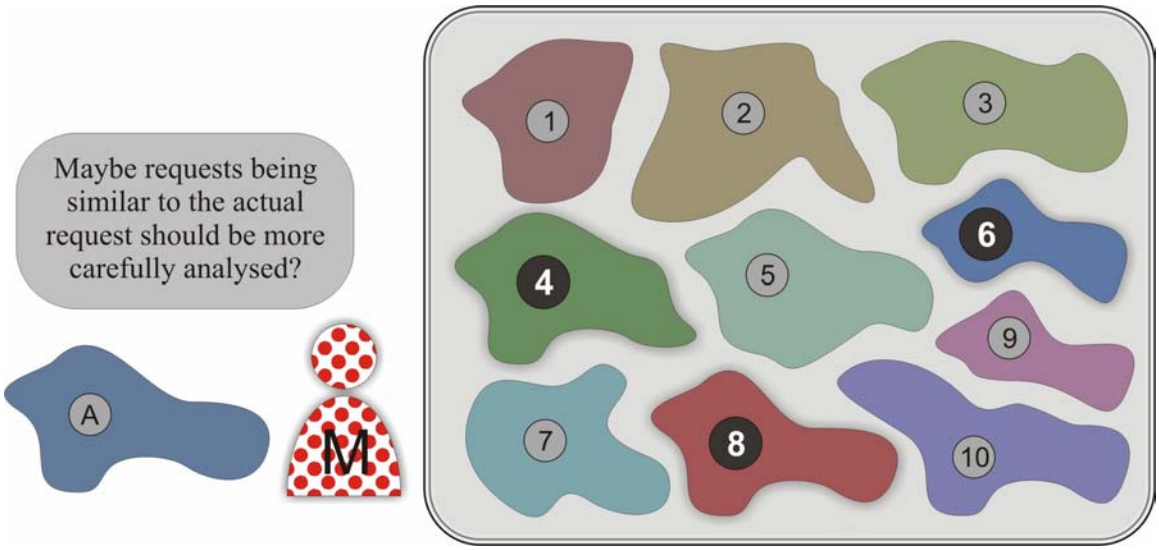

Figure 3.20: The found most similar old requests are marked with darker cell filling around notations

The found most similar and usable old requests are on Figure 3.20 requests with ids 4, 6 and 8. While requests 4 and 8 are selected mostly because of their similar shape with the actual request, request 6 is chosen due to almost identical filling colour irrespective to its smaller size. These two criteria for finding usable old requests should point out on the unavoidable necessity to somehow logically combine them. A great dilemma in this small scale example is concerned whether is more important to take into consideration shape or filling colour while deciding about the similarity of requests. The shape and filling colour are nothing else that two attributes that should be somehow combined while selecting old requests. In reality there are unfortunately much more than two attributes behind every request, and the combination of them becomes even more important. The great challenge is therefore contained in designing a suitable distance function, which is going to be able to identify the subtle differences between the compared requests.

Example 3.2: When somebody is searching for an active holiday with the great riding facilities in Spain, a system should find successful old requests that are either concerned with active holidays or directly with riding. Or for the request for papers in the area of agent technologies, a system is searching for similar requests in order to determine not only how to better focus searching to particular topics, but also to find where the relevant information is located and what else can be interesting.

3.2.2. Using Similar Requests

The found similar requests should serve as a basis for the application of data mining algorithms, which are applied with an ultimate goal to identify the main aspects, being responsible for their successfulness (Figure 3.21). By restricting the application of these algorithms only on the chosen most similar and most successful requests, the expensive mining process is bounded and made more efficient.

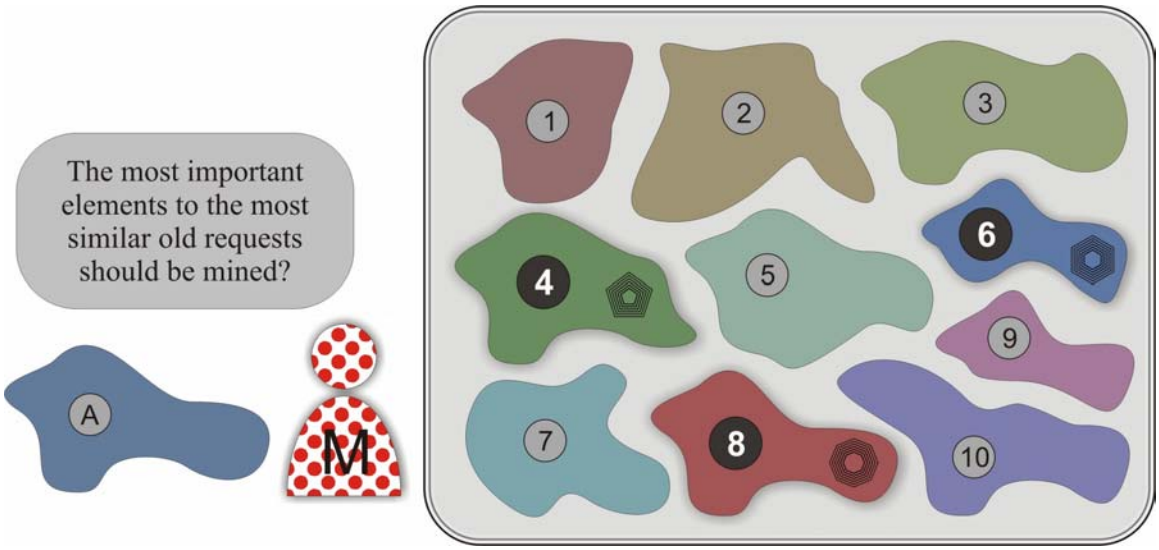

Figure 3.21: The found most similar old requests are analysed and their most important elements are identified in order to be potentially used while improving the actual request A.

The results of mining chosen old requests 4, 6 and 8 are on Figure 3.21 marked by the identification of their elements that can be useful for the adaptation of the actual request. Since the efficiency of data mining algorithms almost always critically depends from the number of pre-selected requests that will be considered, a real problem becomes deciding how many old requests to further carefully analyse. The advantages and disadvantages of selecting either only one or many old requests are briefly summarised on Figure 3.22. On the one hand (Figure 3.22a), the greatest problem with the usage of only one single old request for mining is both in its large sensitivity to features that are present in the chosen request, as well as in the high probability that not everything useful will be in that single selected request. It is highly possible that many aspects will not be present in the chosen request, and that will be therefore lost in spite of being potentially valuable. On the other hand (Figure 3.22b), while simultaneously working with many old filtering requests introduces a significant computation load, it has much better chances to identify really useful features that will improve the actual request. An optimal reasoning is probably

somewhere in the middle, where more than one, but less that 50% of all available old requests are pre-selected to be carefully examined.

Example 3.3: In the mentioned example about an active holiday with great riding facilities, a system should find from the successful old requests not only which places in Spain are suitable for riding, but also which additional activities are good alternative for riding. Similarly, from requests being similar with the request for papers about agent technologies, a filtering system is both finding that UMBC and Agent Link web pages are the best known sources, as well as additionally determining that information about actual agent conferences might be interesting.

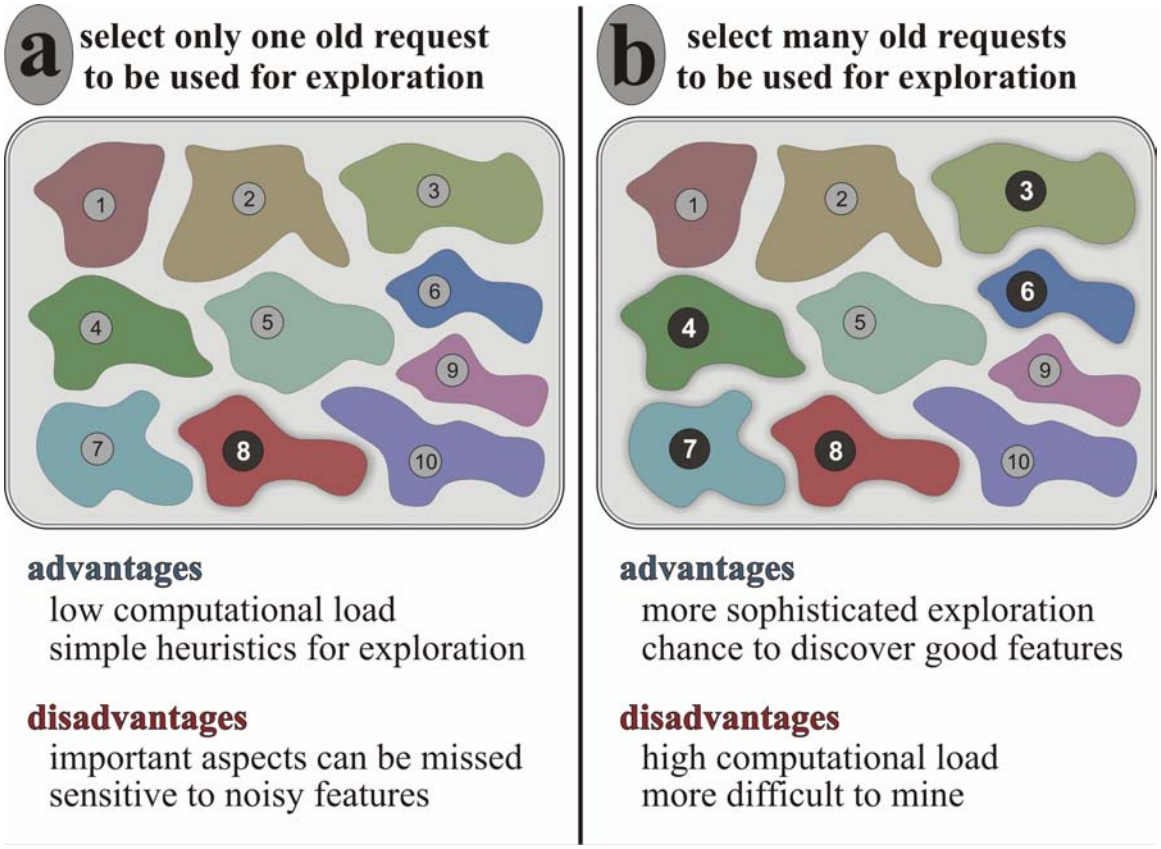

Figure 3.22: The advantages and the disadvantages of selecting the different number of old requests that have the influence while the adaptation of the actual request is performed

3.2.3. Adapting Actual Request

The results of mining activities on the set of chosen similar requests should be seen as the main reasons of their successfulness, and they should be intelligently incorporated in order to make the actual request better and consequently to increase its chances to satisfy the imposed information need (Figure 3.23). In the case where in the actual request some aspects, being very important in the found old requests, are either missing completely or having different values, these aspects are added in order to make the actual request closer to the successful old ones. The expected effect of moving the actual request towards the successful similar old ones is the collaborative intelligent exploration of new information areas based on the impression that important aspects in the liked similar requests can be also important for the actual one.

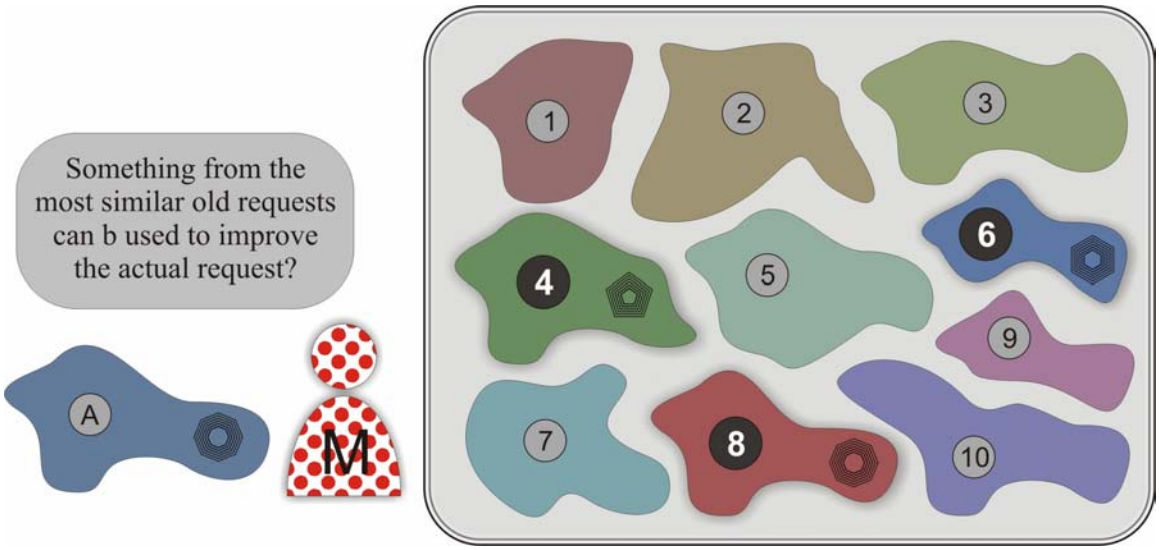

Figure 3.23: Actual filtering request is improved by adding some of the found most important elements from the most similar old requests.

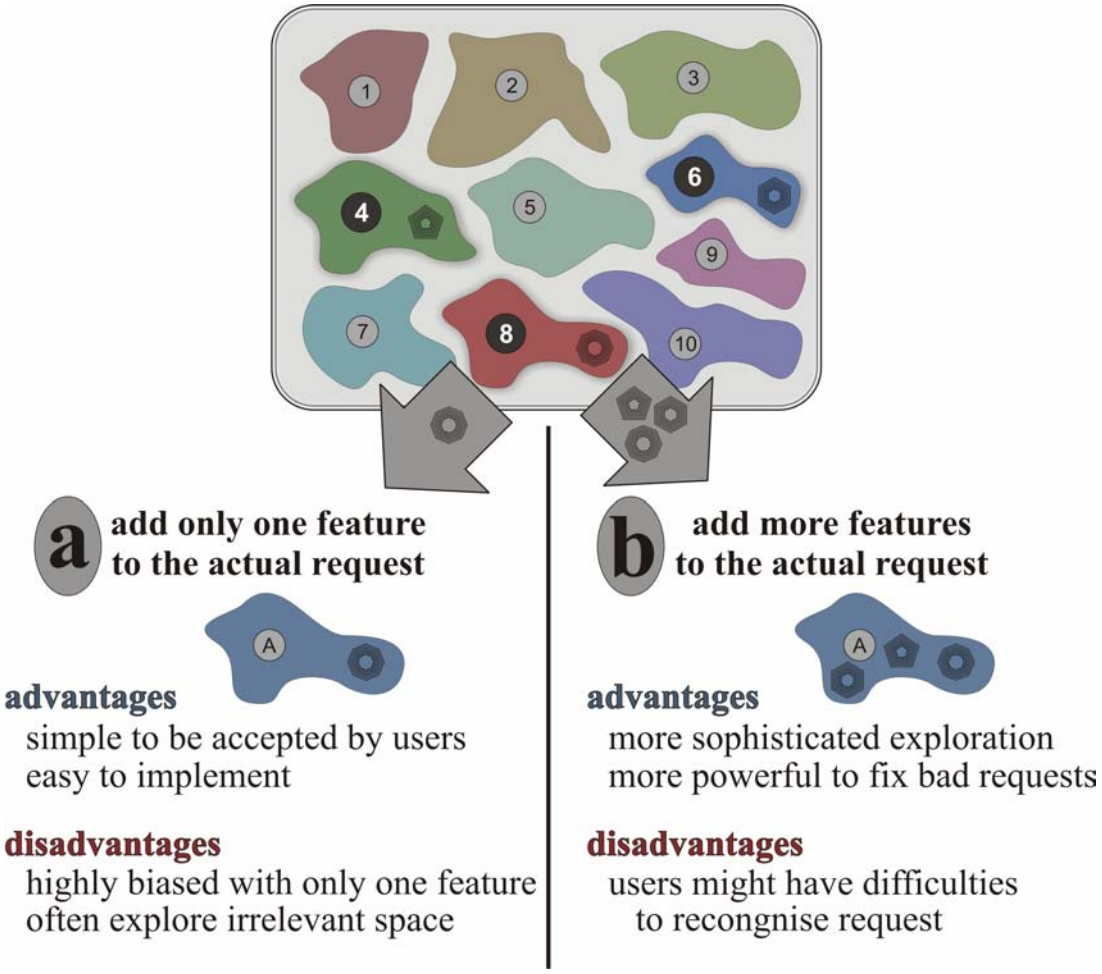

Figure 3.24: The selection of the different number of the found important features, which should be included in the actual request

Figure 3.23 illustrates that even though one most important element is identified in every of three pre-selected old requests, only the element from the request 8 is found of being worth enough to be integrated into an actual request. Such a decision is naturally concerned with a desired level of exploration, which clearly defines how much the actual request can be changed. On the one hand, an extremely small level of exploration will not change the actual request enough, and the potentially interesting information areas will remain undiscovered. Such a situation is on Figure 3.24a illustrated by adding only one novel feature in the actual request. On the other hand, the addition of really too many new features might have the undesired effect of moving the request far away from preferences that have been obtained from a user. The incorporation of 3 new features on Figure 3.24b in the actual request is maybe too ambitious, and a user consequently becomes suspicious regarding obtained results.

Example 3.4: In the holiday example, the found places being good for riding in Spain and to riding alternative activities are added in the actual request in order both to better focus the search and to explore new opportunities for an active holiday in Spain. In the request for papers about agent technologies a system adds the most promising locations for finding the relevant papers and additionally searches for data about the actual agent conferences.

3.3. Coordination Challenges

The fruitful history of information filtering technologies has produced many more or less successful filtering strategies, which make every given filtering problem solvable on many different ways. These already invented strategies can be intelligently combined or coordinated to hopefully obtain a sophisticated solution, having a superior behaviour that surely outperforms any single searching algorithm. In order to develop such coordination solution, being able to choose in a long run for as many filtering jobs as possible the most applicable algorithm, one obviously needs an advanced reasoning that tries to learn about strengths and weaknesses of every available filtering strategy. In nowadays real-time and dynamic environments one should additionally estimate the applicability of each strategy by taking into consideration both its behaviour in similar past situations and the degree of the current availability of relevant system resources.

The remainder of this sub-section will first shortly present one high level example that illustrates the usual coordination workflow without uncovering underlying details. After defining the novel symbols that will be used, these hidden coordination challenges, which are typically dealing with the robust evaluation of the applicability of each strategy while simultaneously learning about available filtering potentials, will be given by one scenario, separated into resource evaluation, job analysis, strategy selection, self healing recovery, and qualification adaptation steps.

Figure 3.25 presents one very simple example of the framework, having only one relatively small community with manager and filtering agents. In spite of its simplicity, the given example is able to illustrate the main coordination role, being the selection of a filtering agent that will find the requested results. It additionally becomes obvious that coordination takes place inside a single community, where one manager maintains the population of different filtering agents. The flow of the information request, named also

inside the community as the filtering job, is through two steps. The first step involves a user, who is responsible to encapsulate its preferences inside the information request, and to send that request to the reachable manager M_1. The subsequent second step is actually the coordination activity, where manager M_1 somehow decides that filtering agent F_{13} is the most suitable, and thus a request will be forwarded to F_{13}. A way how such a decision is made is out of the scope of this high level example, and will be the cornerstone of the remaining sub-sections.

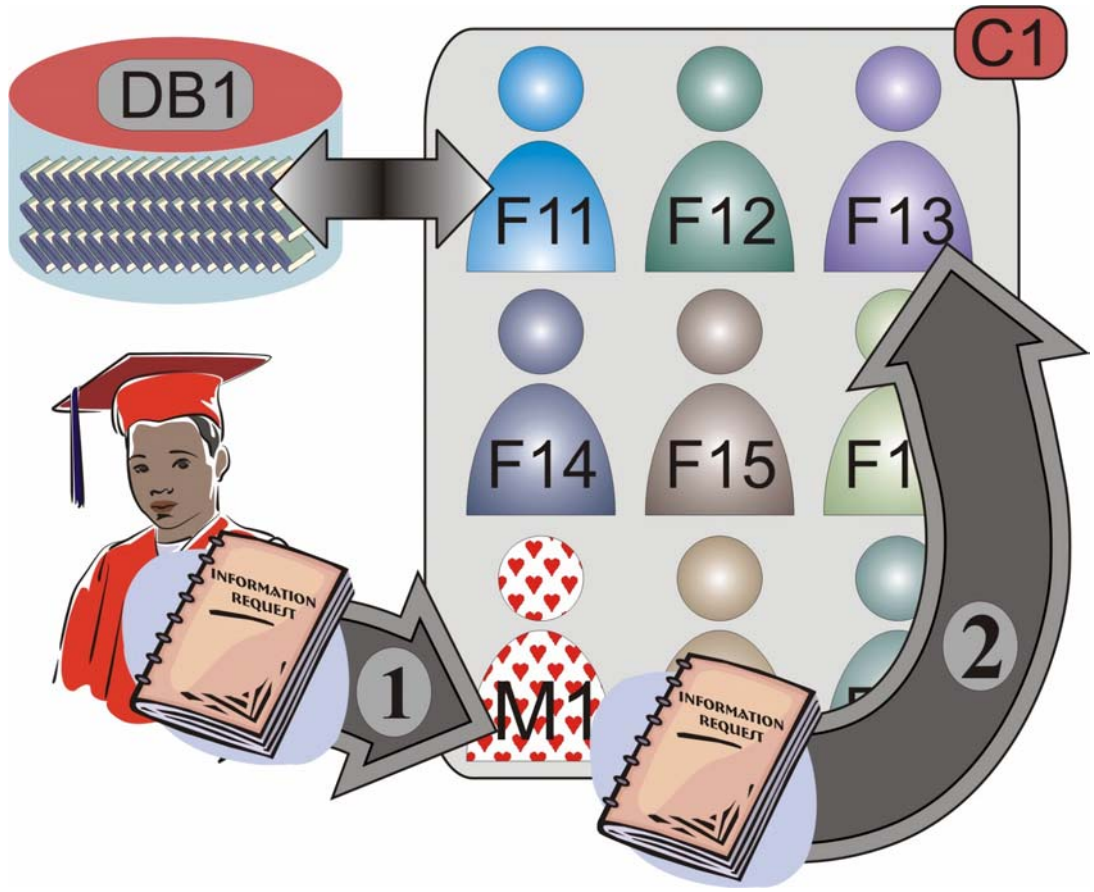

Figure 3.25: After receiving information request from a user, manager M1 decides through coordination that filtering agent F13 is the most suitable and thus should be responsible for a request

The second part of this simple example assumes that filtering has been successful, and that the responsible F_{13} has managed to find results. As being illustrated on Figure 3.26, the found results are first returned to the manager M_1 that has initially received the information request. The last step is returning results to the user, being done by M_1 that is exclusively responsible for the quality of a provided filtering service. Even though manager M_1 can perform the check of the obtained results to be sure what will be returned, this activity is performed in the cooperation, and will not be therefore analysed in this section, where the coordination issues are solely taken into consideration.

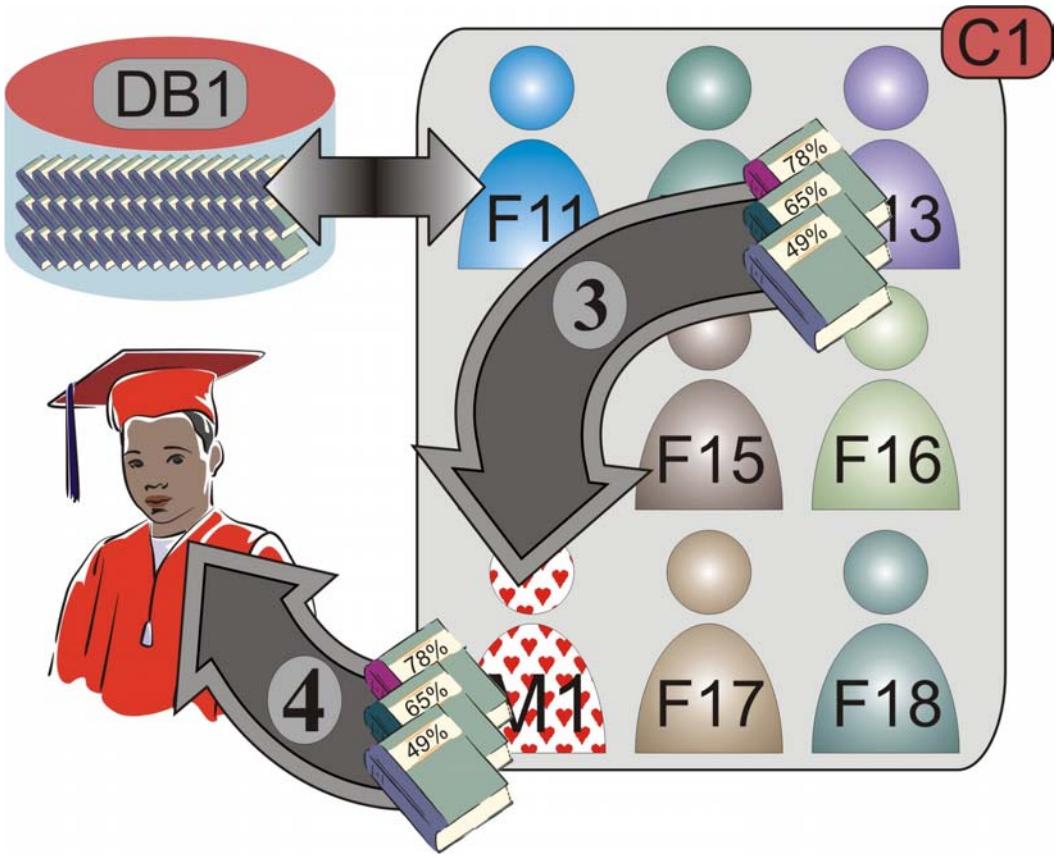

Figure 3.26: Filtering agent F13 has found results that are returned via responsible M1 to user

To make easy understandable the following scenario, Figure 3.27 introduces the novel symbols, which are mainly used for marking the different states of a particular filtering agent. Basically the four novel such symbols will be used for marking that only resource evaluation has been performed, that additionally job analysis has been accomplished, that a particular filtering agent has failed to deliver results, and that knowledge about a given agent has been adapted. Figure 3.27 also gives the definition for a filtering job in the coordination context, being nothing else than information request, as well as the utilised symbols for representing memory, CPU, and database load values.

A coordination scenario will take place inside a community that has one manager (M) and four filtering agents (Figure 3.28). Since a proposed coordination approach assumes the estimation of the load values of the relevant system resources, the used symbols for memory, CPU, and database will be present on every figure that covers the steps of the scenario. The question marks are initially assigned to these icons to illustrate that the resource estimation is still not fulfilled. The analysis of the received information request can be possible while performing the coordination in a domain of text documents, and therefore two question marks symbolise the currently unknown values for the two job properties, being the actual number of words and phrases, respectively. The developed coordination approach is universal by reason of being able simply to omit the job analysis for domains where items are not text documents, in which case the question marks, being assigned to the icon for a filtering job, should be ignored.

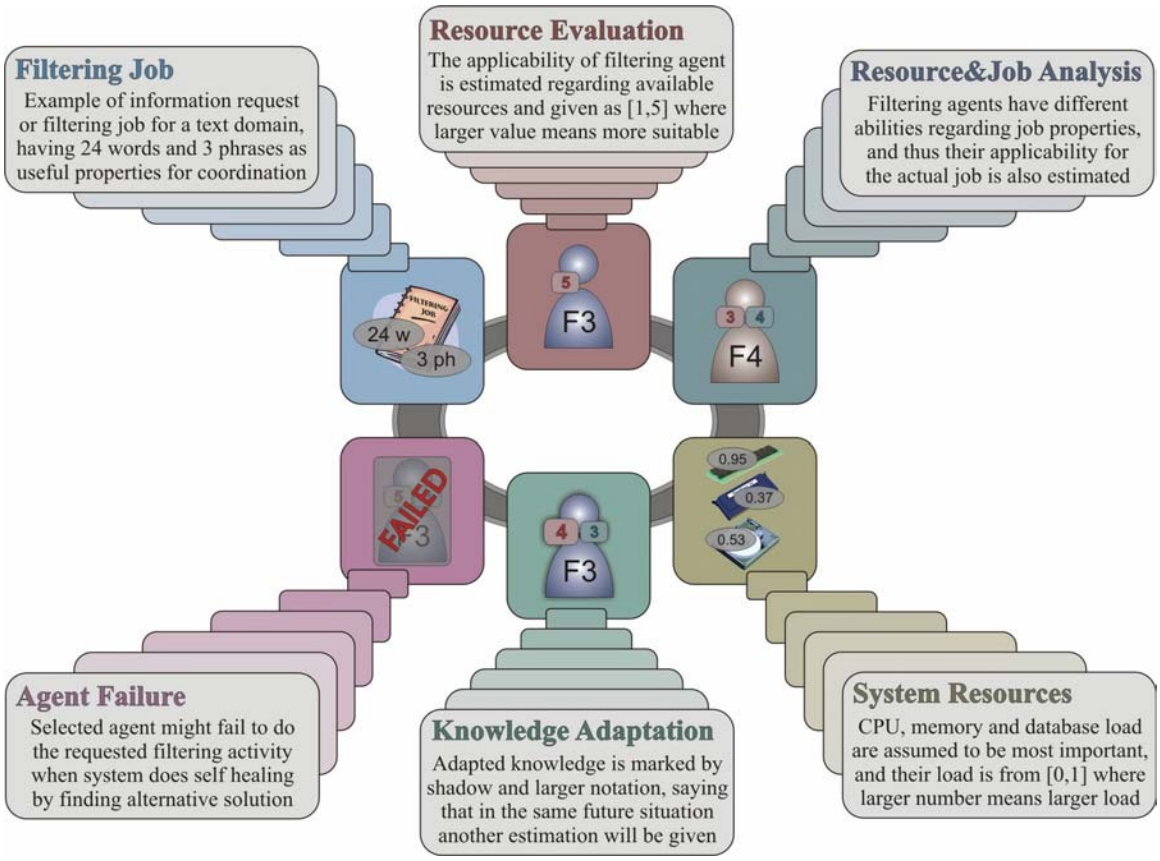

Figure 3.27: Meaning of novel symbols that will be used through coordination scenario

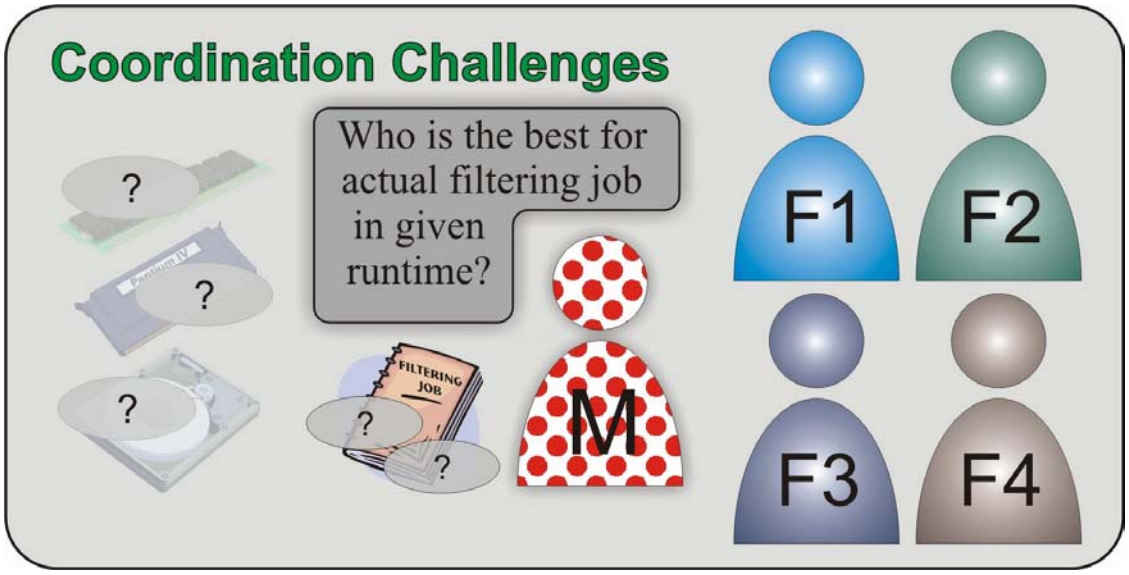

Figure 3.28: Playing ground for the coordination where manager M should decide which filtering agent in the most suitable to be selected for the received filtering job in a current runtime situation

3.3.1. Resource Evaluation

After receiving the filtering job and before making any decision about which filtering agent is going to be selected, the current runtime situation in a system will be determined (Figure 3.29). In the particular case, that actually might mean measuring parameters, such as the system load of CPU and database server, and determining the amount of available free memory. The obtained load values are afterwards somehow combined with the experience that a responsible manager has had with any available filtering agent in past coordination activities (Figure 3.30). That past experience will optimistically provide the knowledge, which clearly specifies for each and every filtering strategy, how much computation power it is necessary, in which extent it loads a database, and the amount of the needed free memory.

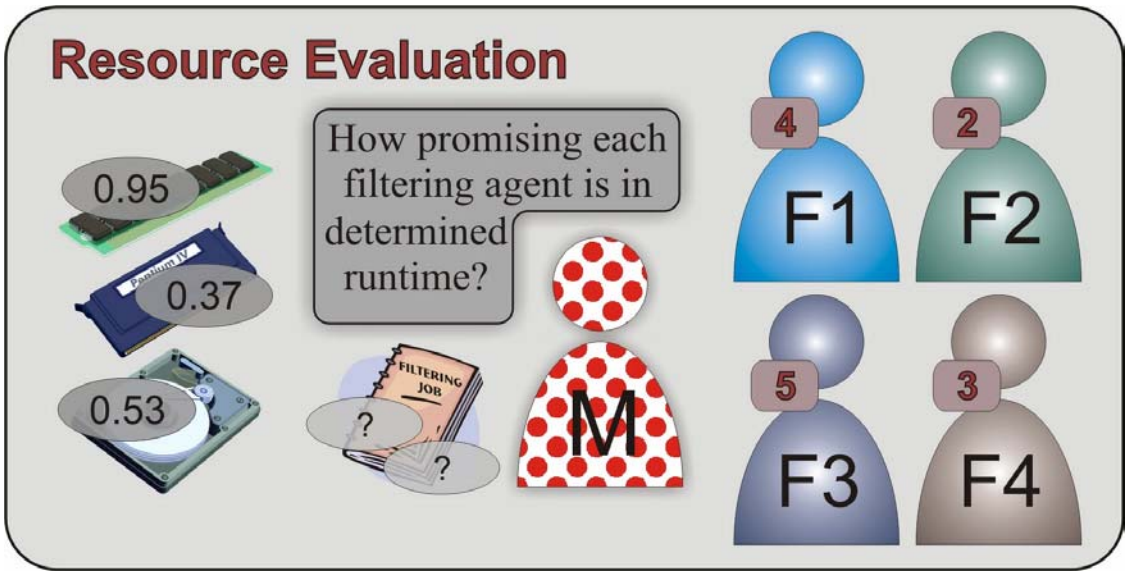

Figure 3.29: The load of relevant system resources is estimated to provide support data for coordination that is consequently managing to optimally utilise available computational power

On Figure 3.29 the load values of memory, CPU, and database server are computed and represented as numbers from $[0,1]$, where a higher value means that a corresponding resource is more loaded. On the one hand, it can be seen from the assigned numbers that memory system is quite loaded as its load is represented by 0.95. On the other hand, CPU is the least loaded resource, having the assigned load value of only 0.37. By using found load values and the experience about the capabilities of filtering agents, the applicability of every known agent is evaluated, and represented as a number from $[1,5]$, where larger number means better corresponding applicability. While filtering agent F_3 happens to be most successfully used in the similar past runtime situations (evaluation has assigned 5 to F_3), F_2 seems to be the least promising agent when only resource load values are taken into account (F_2 applicability is rated only with 2).

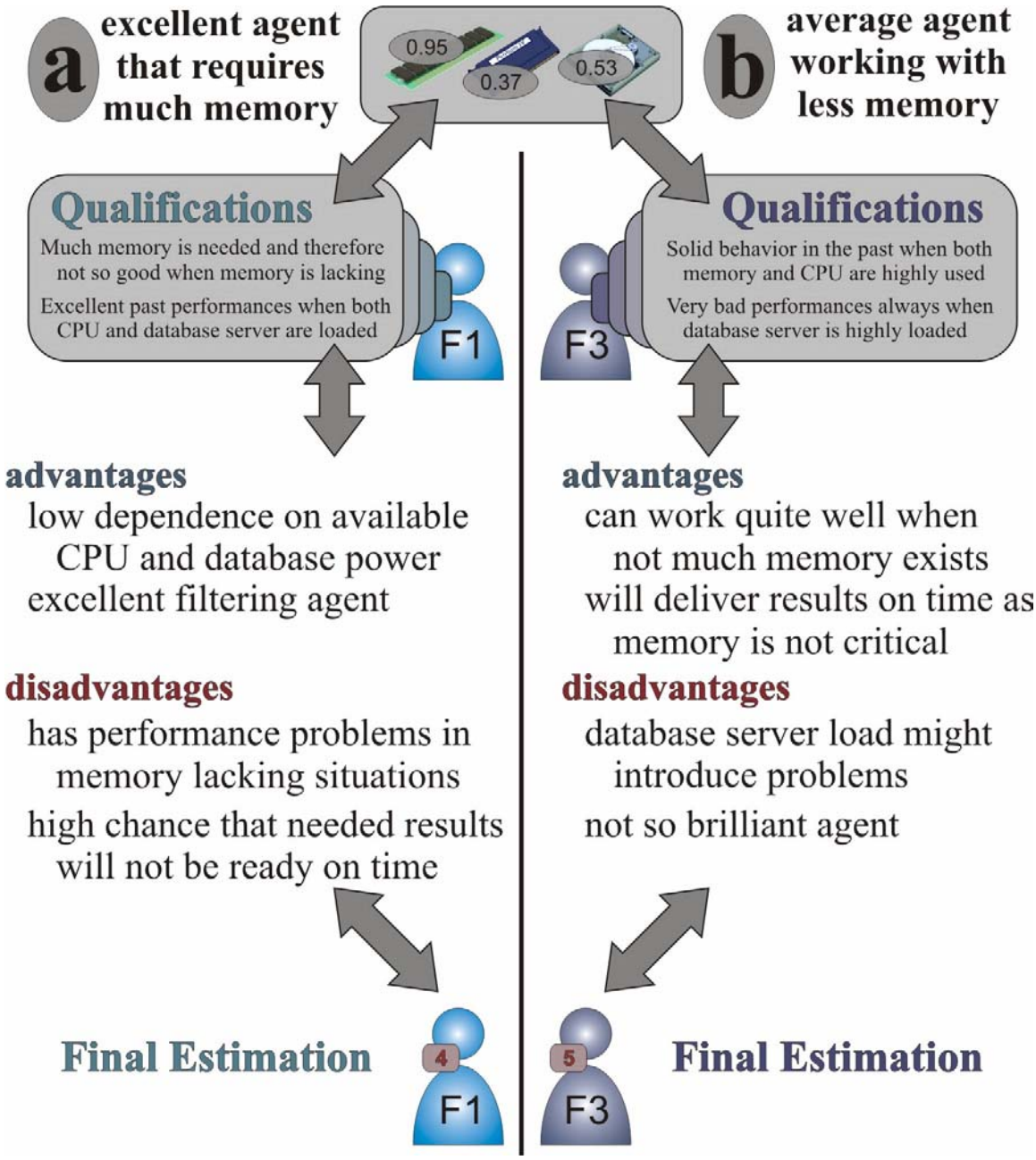

Figure 3.30: Challenge of estimating how every filtering agent can be promising in current runtime

There are very many challenging situations when the known past experience should be somehow combined with the measured load values in order to obtained the estimations about every filtering agent. These tasks become even more difficult since some agents are maybe never used in similar runtime situations, and they are thus still waiting to be tried to prove themselves. In spite of that, an uncountable number of different situations, where the available agents have been already tried, will open many opportunities for speculating which agent might be currently more promising. But, even in the case where enough past experience is available about analysed agents, it may be a quite big challenge to combine their different capabilities into a single estimation value. Only one of possible scenarios

about deciding which agent is better is given on Figure 3.30. The agent F_1, while having the excellent behaviour irrespective to the high load of CPU and database server, might show serious response time problems when there is not enough available free memory. The yet another agent F_3 possesses almost completely reverse capabilities. While F_3 is known as an agent with average performances that can have serious problems when the database server is loaded, F_3 can successfully work even when there are only a few bytes of a free memory. Although the general comparison between F_1 and F_3 will surely give the advantage to a much better F_1, a current memory lacking runtime situation brings many penalties to F_1. A final decision is therefore giving a slight advantage to F_3 by assigning 5 as estimation, mostly because of its strong guaranties to be able to work even when there is not enough memory. F_3 will produce needed recommendations on time, which is better than probably a long waiting for excellent results that F_1 can offer, being consequently estimated by 4.

3.3.2. Job Analysis

A brilliant human capability to produce different types of filtering jobs opens many challenges, being mainly concerned with somehow assigning the right types of jobs to the filtering agents with the necessary skills (Figure 3.31). The main objective of job analysis is therefore the estimation how successfully every agent can process the received job based on the available previous experience. Such an analysis is unfortunately not treating jobs as black boxes, and consequently makes the whole underlying coordination approach to be potentially domain dependant. As the retrieval of text documents is the one of most typically occurring tasks, this sub-section will assume that the filtering jobs are actually queries, being built out of words and phrases. The generic applicability of a coordination scheme is still preserved by simply omitting job analysis for all other unsupported types of jobs.

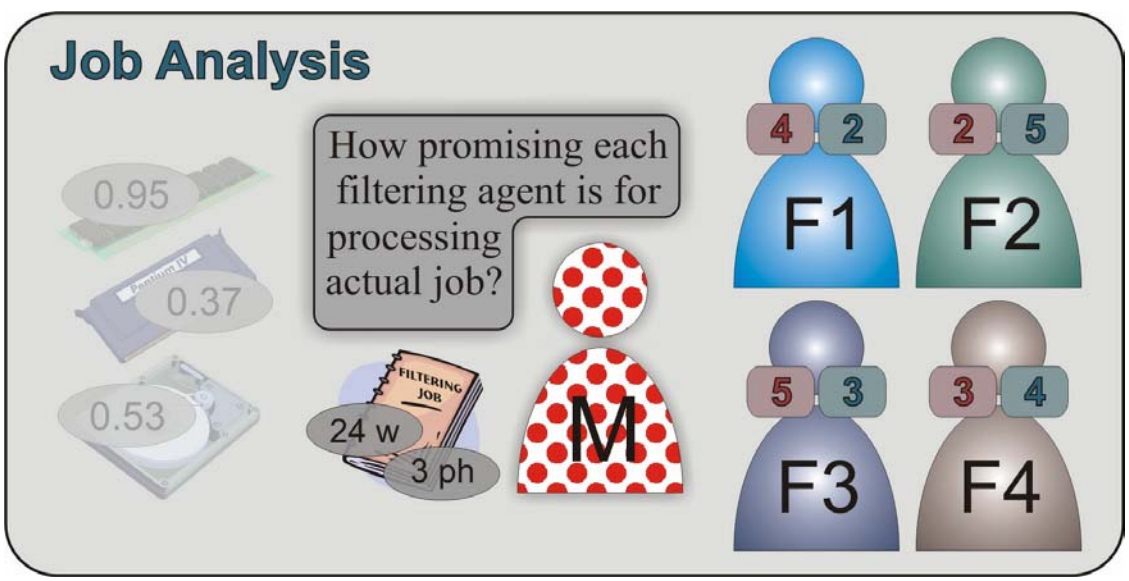

Figure 3.31: The main properties of received filtering job, such as number of words and phrases, are determined, and the suitability of filtering agents is also estimated regarding these job properties

Only some typical examples of filtering jobs in the broad domain of text documents, which require the application of different strategies, are given as follows. The typical on-line application of any search engine is usually concerned with short queries, having only few words [260], where response time has to be extremely short. In order to successfully handle many short and imprecise jobs, one has to additionally think about using strategies for the intelligent query expansion [83]. The personalised information delivery, which is specialised, adaptive and exploratory [235], has also to somehow cope with the profiles, having hundreds of words. Because these huge profiles are at the end used for searching, one needs either to perform dimensionality reduction [127] or to seamlessly split a profile [29].

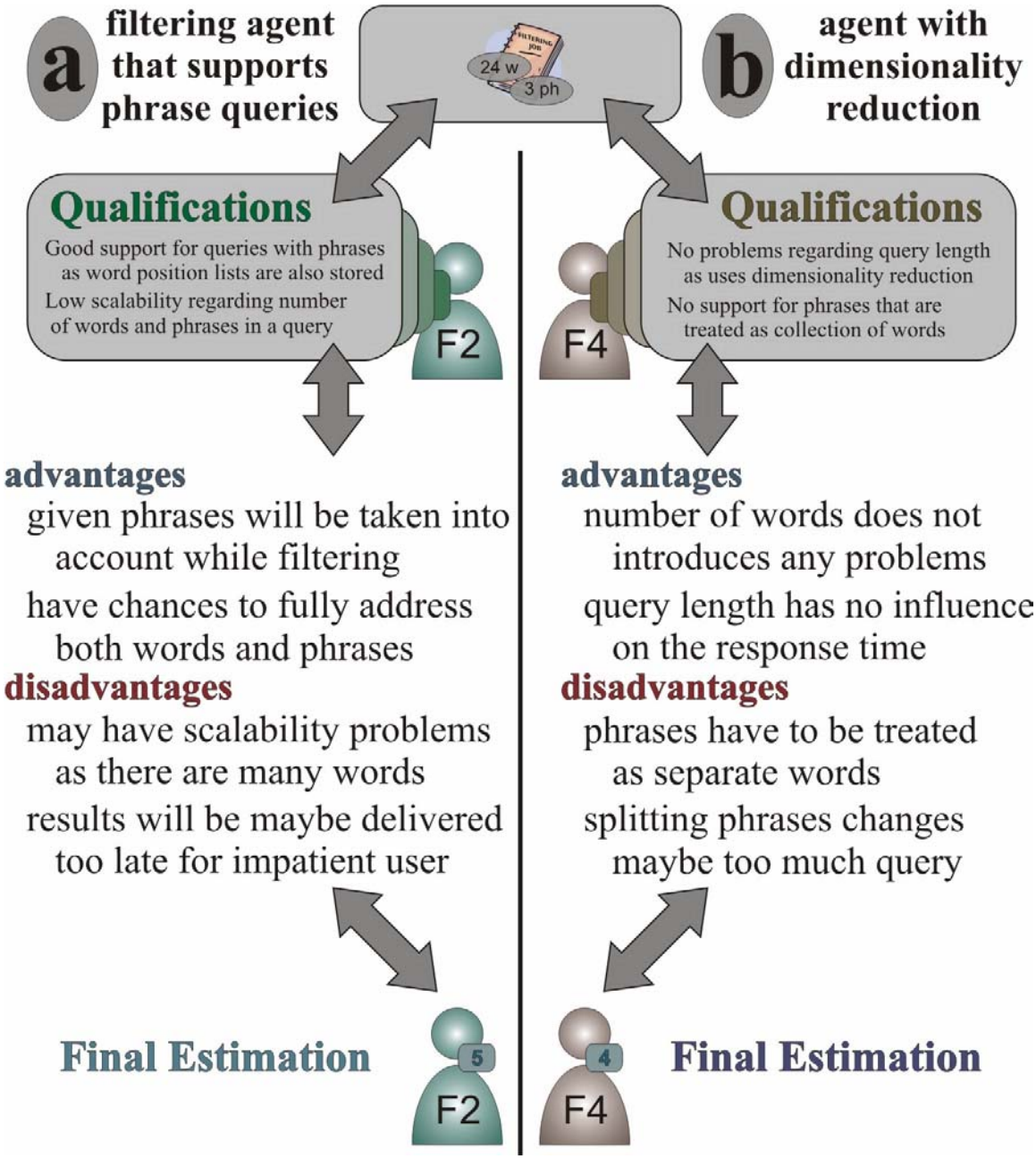

Figure 3.32: Challenge of figuring out how good each filtering agent can be for the received job

On Figure 3.31 manager M has to estimate how a usage of any from the four installed filtering agents is appealing for processing the job, having 24 words and 3 phrases. Every filtering agent usually utilises very different strategies that might have the limited support towards the particular types of jobs. On the one had, the document retrieval engines based on inverted lists [58][59] store for each and every word its positions, and consequently can support queries with phrases. Such a filtering solution unfortunately has to separately process each word from a query, and thus might have scalability problems for longer requests. On the other hand, strategies that use some sort of dimensionality reduction, such as Principal Component Analysis (PCA) [127] or Self Organising Map (SOM) [148], can easily support jobs with really many words. At the same time, these strategies do not have position lists, and consequently cannot handle phrases. These two and many other strategies clearly illustrate the necessity to take care of the type of a job, while making a decidion which one will do filtering. On Figure 3.31 while the most promising agent happens to be F_2, whose applicability for a particular job is estimated with 5, the least applicable is F_1, having only 2 as the corresponding suitability for the given job.

The advantages and disadvantages of a potential usage of only two filtering agents for a given job with 24 words and 3 phrases are deeply analysed on Figure 3.32. While agent F_2 plays the role of a filtering engine with an inverted list that can support phrases, F_4 uses the dimensionality reduction and thus ignores phrases. Even though F_4 can support significantly larger jobs than F_2, the little better estimation is given to F_2 (F_2 is rated with 5 and F_4 with 4), probably because the query with 24 words is not so big. The used reasoning assumes that for a given job both F_2 and F_4 can work acceptably fast, and thus the advantages is given to F_2 that can provide more accurate results.

3.3.3. Strategy Selection

By using the determined environmental properties, as well as the found features of the actual filtering job, together with the characteristics of different strategies, the appealing idea about which filtering strategy is the most suitable in the current situation should be obtained. In order to better explore the characteristics of all available strategies, instead of giving the job always to the most promising strategy, sometimes another strategy is going to be chosen, being the way of giving the chance to that strategy to improve its qualifications. A challenge behind is therefore contained in deciding either to exploit the already learnt properties of well known strategies or to explore the unknown, potentially promising, capabilities of novel ones.

On Figure 3.33 a manager has decided to forward the job to the filtering agent F_3 that has the best possible resource estimation (the applicability of F_3 in the current runtime situation is rated as 5). As being given on Figure 3.34, where the two best candidates are directly compared, F_3 is at the same time a quite novel filtering agent, and its selection therefore stimulates the excellent exploration of its capabilities, where really much might be learnt. The closest concurrent to F_3 happens to be the filtering agent F_4 mostly by reason of its solid support for the actual job (the level of supporting the current job by F_4 is estimated by 4). It is shown on Figure 3.34 that F_4 is the well-known filtering agent,

about whom the solid experience is available, being also its important strength. Figure 3.33 finally shows that F_2, offering the best possible support for a given job (job analysis has assigned 5 to F_2) is not so good candidate to be selected, and its capabilities are not analysed in detail on Figure 3.34. The reason behind is in its very bad resource estimation, which is marked only with 2, and which specifies that possible serious resource problems might occur in the case of its selection.

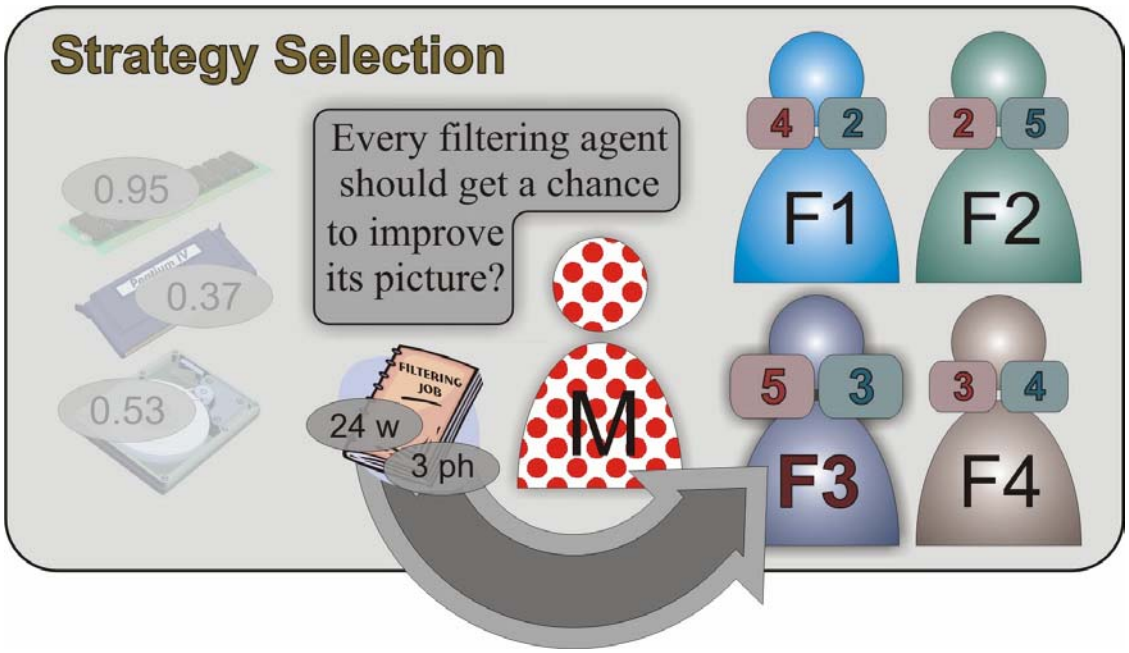

Figure 3.33: Based on the performed estimation of resources and job analysis filtering agent F3 is selected to be responsible for doing filtering and delivering expected results

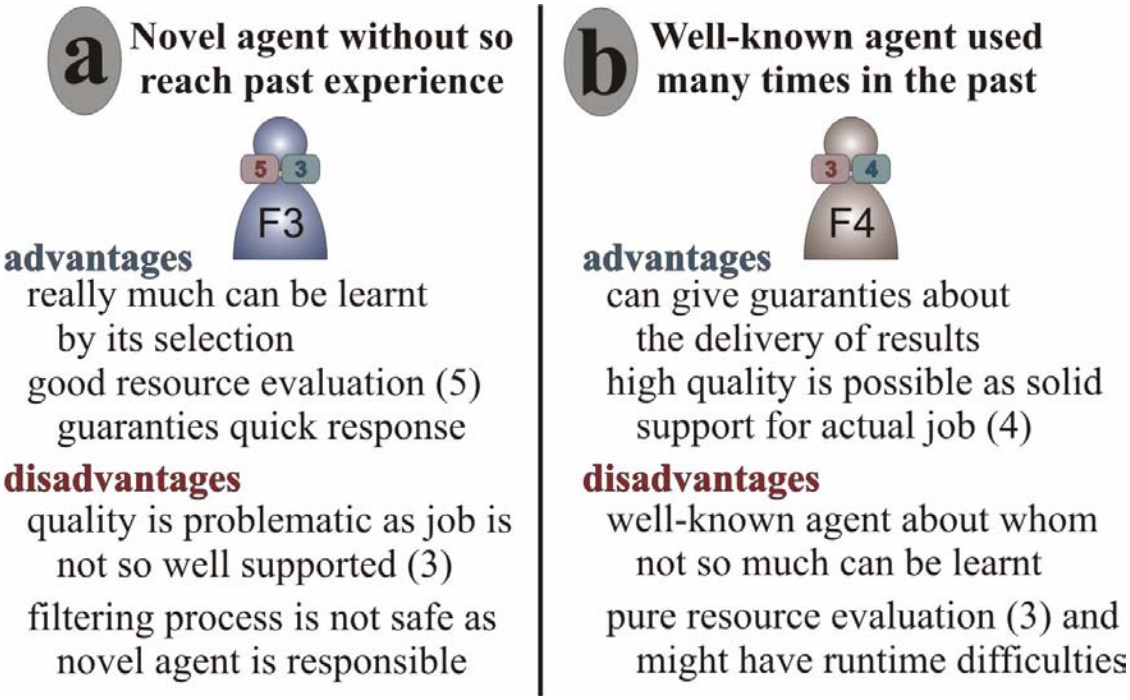

Figure 3.34: The advantages and disadvantages of a selection between two filtering agents that have the best qualifications, and when there is the different level of past experience about them

3.3.4. Self Healing Recovery

The challenge of exploiting the well known filtering agents, as well as the potentially promising novel ones, becomes even more severe by additionally requesting that some guaranties, concerning the delivery of results, have to be given. The reputation of a whole community will be considerably demolished if it is often happening that a user is getting either bad or not results at all. A community is always seen as a black box for any sender of an information request, which means that it does not matter which filtering agent is not working well. A final result set is the only thing that is counting, and the manager, being generally responsible for the quality of the provided filtering service, has to ensure that as often as possible a user is satisfied with the obtained recommendations. The exploration of the unknown capabilities of novel filtering agents evidently has somehow to be packed into self healing recovery mechanisms, which will guarantee that in the case of a failure of one agent, the alternative one will be asked to perform filtering.

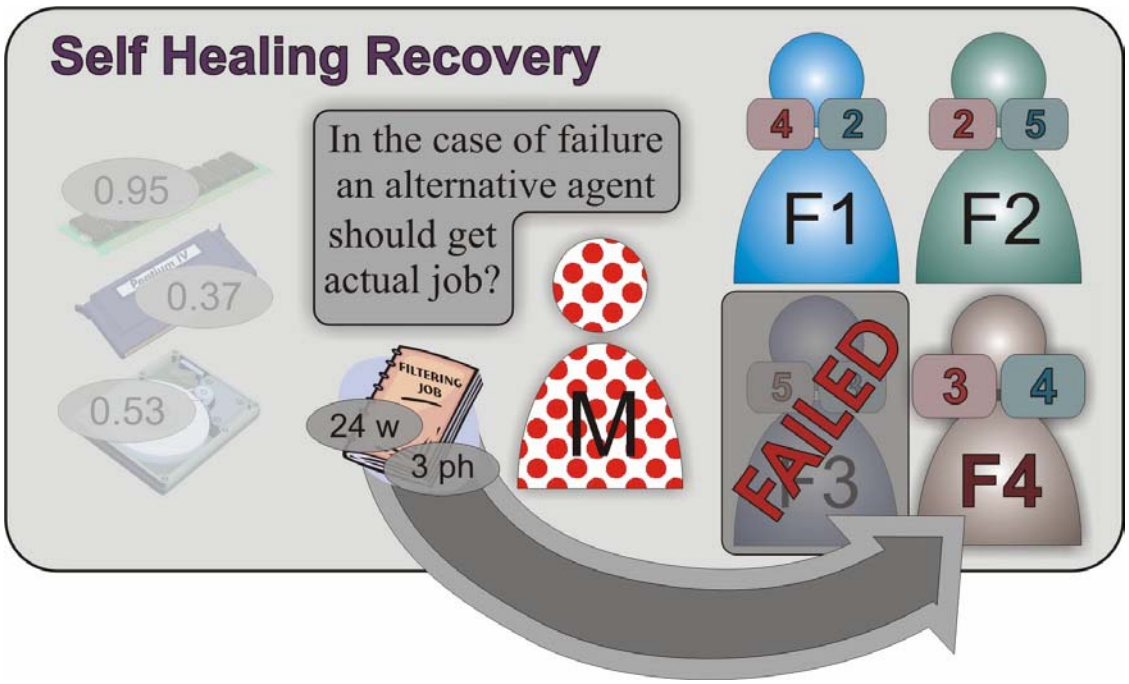

Figure 3.35: After the failure of the first selected agent F3, the alternative F4 is chosen as a recovery

On Figure 3.35 the first selected filtering agent F_3 has unfortunately failed to deliver requested results. Such an outcome is maybe not so surprising since F_3 happens to be a novel agent, as being described on Figure 3.34, whose real capabilities are still waiting to be discovered. After noticing the failure of F_3, which can either not delivering the results in the permitted amount of time or explicitly declaring that the requested filtering cannot be performed, the manager is figuring out that F_4 can be the solid recovery option. The greatest motivation, behind selecting F_4 to reliably repair the critical actual situation, is in its excellent and solid past experience (Figure 3.34). Since F_4 is a well-known filtering agent, manager plays right now more on the exploitation of what is already known, and hopes that the expected results will be quickly found. These results are now really needed

because some time has been already lost due to the unexpected failure of F_3, and a user might become impatient.

3.3.5. Qualification Adaptation

While the runtime characteristics of a strategy can be adapted as soon as the filtering results become available or the failure happens, the measure of successfulness in finding accurate results for a particular type of a job cannot be changed before a user feedback is not received. An ultimate goal of these adaptations is establishing a more realistic picture about the potentials of the available strategies. The expected desired final effect is that a system is hopefully going to learn to even better coordinate activities for future jobs.

After the failure of F_3 manager can learn for example that F_3 cannot work so reliably in the particular runtime situation. On Figure 3.36 the adaptation of knowledge about F_3 is symbolically represented by decreasing resource estimation for F_3 from 5 to 4 in order to illustrate that in the same future runtime situations F_3 will not anymore be evaluated with 5 but with 4. Under an assumption that a user has been happy with the results that F_4 has found, the already good qualifications of F_4 to well support the given job can be further boosted. On Figure 3.36 the estimation of an applicability of F_4 for a given job is increased from 4 to 5, which means that in the future F_4 will be better estimated for the same type of jobs.

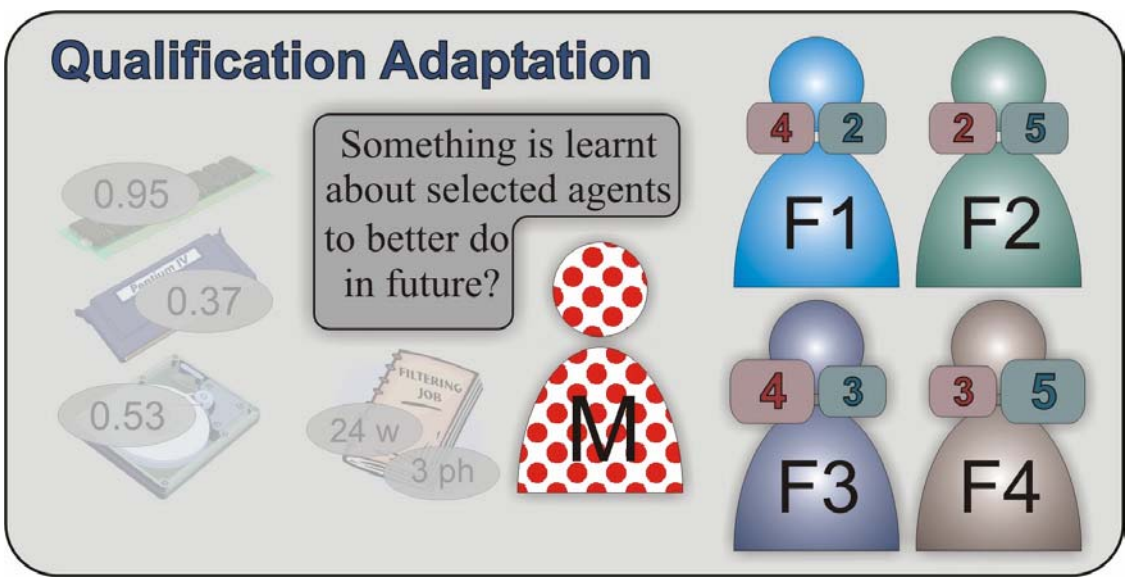

Figure 3.36: The knowledge about the triggered filtering agents F3 and F4 is adapted to better represent their capabilities and thus to enable more successful coordination activities in the future

3.3.6. Formal Example of Filtering Job

The evaluation, selection, recovery and adaptation steps are essential building blocks for generic coordination mechanisms, which are applicable on each and every filtering job from any application domain. In spite of this universal coordination applicability, the best way to understand that each filtering job can be resolvable on many ways, which can

be quite different for example as far as the resource requirements are concerned, is to formally present one specific job, let's say from the domain of document retrieval. This formal example will help in understanding mostly by illustrating the possible ways of utilising many different job properties or constrains while deciding which text documents are the most promising recommendations.

Example 3.5: Let's imagine that one research assistant in a company dealing with the exploitation of renewable energy should find five closely related project proposals either in English or in German that are about wind power plants and that are not older than one month.

The information request that encapsulates all the mentioned constrains can be formally presented as it is given in Table 3.1, where each and every constrain is represented as the corresponding filtering attribute. On the one hand attributes such as *sources*, *languages*, *domains*, *expected quality* and *type of results*, have strictly predefined permitted values. For example the legitimate values for *type of results* attribute are scientific paper, news article, conference call, project proposal, trade publication and business report. On the other hand *preference key words*, *actuality* and *number of results* attributes take values without so strict limitations, i.e. any word or phrase from a given language can be given as a *preference key word*.

This variety of filtering attributes naturally imposes the many potential ways of using them while searching for results. One can first search for all project proposals that belong to a renewable energy domain and then check other constrains, whereas the other can perform the initial reduction of a search space through the examination of the document actuality. Obviously many more or less successful filtering strategies can be designed, and the arising problem is naturally concerned with making a decision about how each particular strategy is good in a given situation.

Table 3.1: Formal description of one filtering job in document retrieval domain

Filtering attribute name	Filtering attribute value
Preference key words	Wind power plant, wind speed sensors, wind turbine, wind farm
Actuality	One month
Sources	Internet, intranet, mail box
Languages	English, German
Domains	Renewable energy
Expected quality	High
Number of results	5
Type of results	Project report

The challenges, which unfortunately exist when trying to do successful large scale coordination between many different filtering strategies, are only illustrated in a given coordination scenario, being accompanied with a formal example of a filtering job from the information retrieval domain. The possible ways of addressing evaluation, selection, recovery and adaptation problems will be the main topic in the Sections 6 of this thesis.

3.4. Conclusion

The problems of finding the right information at the right time have been analysed from the point of view of one multi-agent filtering framework. Although the application of intelligent agents has released the user from the burden of thinking where the relevant data are located and how they can be retrieved, the filtering agents have introduced many novel challenges. The fundamental architecture of the proposed filtering framework has been given on Figure 3.3 in order to make possible the precise specification of the main underlying challenges, being shortly named as cooperation, exploration and coordination on Figure 3.4.

The cooperation challenges have been in detail discussed in Section 3.1. They have been basically concerned with automatically finding communities that are most suitable for processing the actual information request. The underlying problems of first obtaining the up-to-date knowledge about the potentials of other known communities, then deciding to which the request should be dispatched, and finally composing the final result set with learning from a user feedback, have been illustrated through one scenario.

The scenario has been also utilised for presenting the main exploration challenges in Section 3.2. These challenges have been mainly concerned with exploiting the knowledge, being encapsulated into numerous already processed requests, to improve the actual one, and to increase its chances to find results that will lead towards the solid user satisfaction.

The chapter has been finished with the most important Section 3.3, where coordination challenges have been addressed. Section 3.3 has actually presented main problems while deciding which filtering agent, each encapsulating its own searching strategy, is the most promising to be applied for resolving the actual job in the particular runtime resource situation. Problems with both resource estimation and job analysis, being the activities that provide the necessary support data for coordination, have been discussed. The discussion about coordination challenges has been concluded with the analysis how the underlying selection mechanisms can recover after the failures of filtering agents, as well as efficiently learn from both from the available user relevance feedback and the response time measurements.

Chapter 4
Cooperation Approach

As being already shown in the Chapter 3 where the major cooperation problems have been discussed, the needed information is usually scattered around many different, more or less dynamic, distributed sources. Two cornerstone challenges therefore become both finding which sources should be consulted for resolving the particular request, as well as putting the found results together. While the challenge of searching for sources is known in the literature as database selection problem, the composing of a final result set is often simply referred as the information fusion [273][274]. The known specialised approaches for handling these challenges, which will be briefly presented in the Section 4.2, seem to be unfortunately too heavy for efficiently utilising the highly dynamic distributed sources. Much less heavy cooperation approach, which is already published in [9][10], and which is based on the application of the intelligent cooperative agents, is going consequently to be profoundly presented in this Chapter.

The fundamental cooperation idea is based on the installation of at least one filtering community around each database, and on setting up sophisticated mechanisms, which enable that these communities can efficiently help each other while processing incoming requests. Although the filtering request can be sent to any community, the most suitable communities will be autonomously found, and the request will be then dispatched to them. The found results will be finally collected, and only the best ones will be returned to the sender of the request. The most appealing property behind this cooperative processing is that everything is done transparently for a user, being not any more forced to manually think where the request should be sent, and which obtained filtering results are really the best ones.

The remainder of this Chapter will be organised as follows. After giving the necessary definitions and useful preliminaries in order to make possible the precise presentation of the cooperation approach, the major drawbacks of other attempts to provide solutions for the distributed information retrieval will be critically presented. The cornerstone part of this Chapter is then contained in the section, which gives principles being in the basis of the developed cooperation approach that is naturally separated into updating, estimating, dispatching, composing, and adapting steps. This chapter is afterwards finished with the short resume which additionally gives comments about the potential future improvements of a given approach.

4.1. Definitions and Preliminaries

To clearly specify the roles of implemented agents, as well as to formally define terms that will be used while concisely presenting realised cooperation approach, the definitions of Filtering Agent, Manager Agent, Filtering Community, Cooperation, Filtering Request, Community Reliability, Community Content Description, and Community Description will be given as follows:

Definition 4.1: *Filtering Agent* F encapsulates one particular searching strategy that is solely used for finding recommendations from the underlying database. In the domain of text documents, the searched recommendations are articles, and the utilised strategy is always able to receive requests with keywords. Depending on the additional capabilities of the encapsulated filtering strategy, sometimes the extra preferences, such as preferred actuality of results, desired data sources that will be consulted while searching, minimal quality of recommendations, and so on, might be supported. Since the design of filtering agents is out of the scope of this thesis, it is enough to think about them as being wizards, whose inputs are filtering requests, and outputs are results that are no matter how found. Such an abstraction of a filtering agent, together with a manager M that sends requests and receives found results, is presented on Figure 4.1, where F has managed to deliver three results, with relevance values 78%, 65% and 49%, respectively. The way how these relevance values are computed by F will be also ignored, and will not be important for a manager that solely utilise the services of filtering agents.

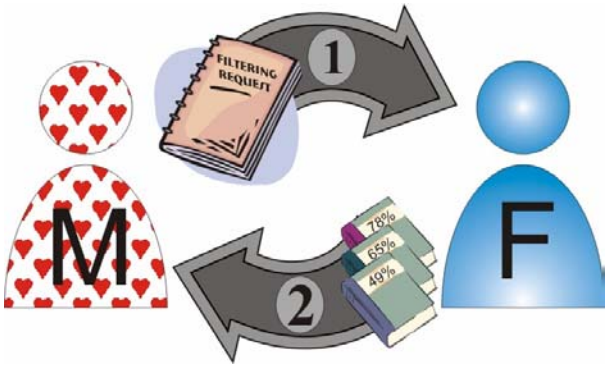

Figure 4.1: Gives a high level idea about filtering agent that somehow delivers results for the request, being received from a manager. The way how filtering agent works is out of the scope of this thesis.

Definition 4.2: *Manager Agent* M receives the filtering requests either from users or from other managers, and autonomously decides what to do with them. From the view point of cooperation, the currently responsible manager, being the one that has initially received the request, is looking for other managers that maybe have better qualifications to process the actual filtering job. The way how manager agents fulfil these cooperation activities will be the main topic of this Chapter. Other duties of a manager are concerned with exploration and coordination tasks, and their concise definitions, as well as practical realisations will be offered in Chapters 5 and 6, respectively.

Definition 4.3: *Filtering Community* FC or shortly *Community* C is the collection of many different filtering agents that are tailored to efficiently do searching depending on the current availability of system resources and the properties of the actual request on the

underling collection of objects. Instead of having only filtering agents, each and every community has also one manager agent M that is solely responsible for ensuring the high quality of the provided filtering service. The example of one filtering community is given on Figure 4.2, where a community C_2 is installed around distributed database DB_2, and contains one manager agent M_2, and five different filtering agents F_{21}, F_{22}, F_{23}, F_{24} and F_{25}.

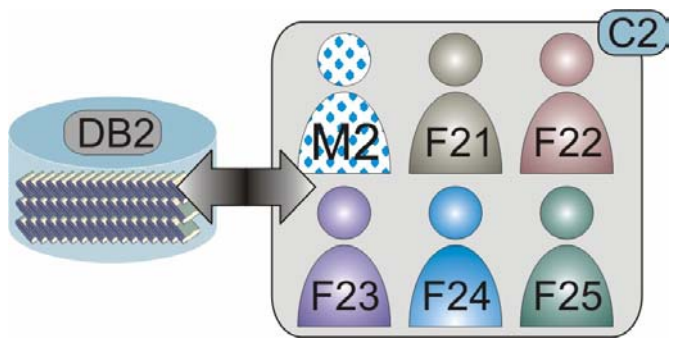

Figure 4.2: Filtering community C2 having one manager M2 and five filtering agents

Definition 4.4: *Cooperation* is performed between manager agents in order to find the most competent communities for processing the received request. This Chapter is focused on presenting the concrete realisation of a cooperation approach that is able to efficiently fulfil these requirements.

Definition 4.5: *Filtering Request FR* describes user preferences towards documents that will satisfy the imposed information needs in the best manner. Each filtering request can be formally represented as the collection of pairs $\{t_i, \omega_i^{(r)}\}$, where t_i corresponds to term in a request, and $\omega_i^{(r)}$ represents the importance of the corresponding term t_i. While larger positive value of $\omega_i^{(r)}$ means that it is more important that term t_i is found, larger negative value of $\omega_i^{(r)}$ assumes that it is better that term t_i is not present in the analysed document. In a course of this thesis the terms information request, filtering job and query are also sometimes utilised as synonyms.

Each and every filtering request FR can have many extra parameters, specifying more precisely users' preferences. One of such additional attributes, which will be utilised for cooperation purposes, is the expected quality of filtering results q_e. The intention behind q_e is that whenever the user specifies high q_e, it is quietly also accepted that the response time might be longer. The yet another extra parameter is n_r, being the requested number of filtering results.

Definition 4.6: *Community Reliability* r_C models the successfulness of the particular community while processing the past filtering requests. Filtering communities that have more often provided the results, being liked by the users, will have larger reliability r_C values, and reverse.

Definition 4.7: *Community Content Description CCD* is the short description of the underlying collection of documents, and CCD can be seen as the collection of the given number of the most important pairs $\{t_i, \omega_i^{(d)}\}$. Element t_i corresponds to a particular term in the document collection to which the given community has access, whereas $\omega_i^{(d)}$ is the number of documents from the underlying collection that have term t_i. The number of pairs should be carefully chosen since it mainly defines the size of a corresponding CCD, and consequently represents the main bottleneck whenever the transmission of CCD is necessary.

Definition 4.8: *Community Description CD* represents corresponding community in concise and descriptive way both to ensure an efficient exchange of descriptions between communities and at the same time to provide enough information to other communities that want to cooperate with it. Formally, community description CD is $\{CCD, RD, t_C, r_C\}$, where element CCD is the community content description, RD is the retrieval date when CCD is obtained, t_C represents the period of asking for new CCD, and r_C is community reliability in processing the filtering tasks in the past. The illustration of a relationship between the whole community description CD on one side, and CCD, RD, t_C and r_C on the other side is given on Figure 4.3. The example values for RD, t_C and r_C, as well as 4 different $\{t_i, \omega_i^{(d)}\}$ pairs inside CCD, having only 18 different such pairs, are there presented.

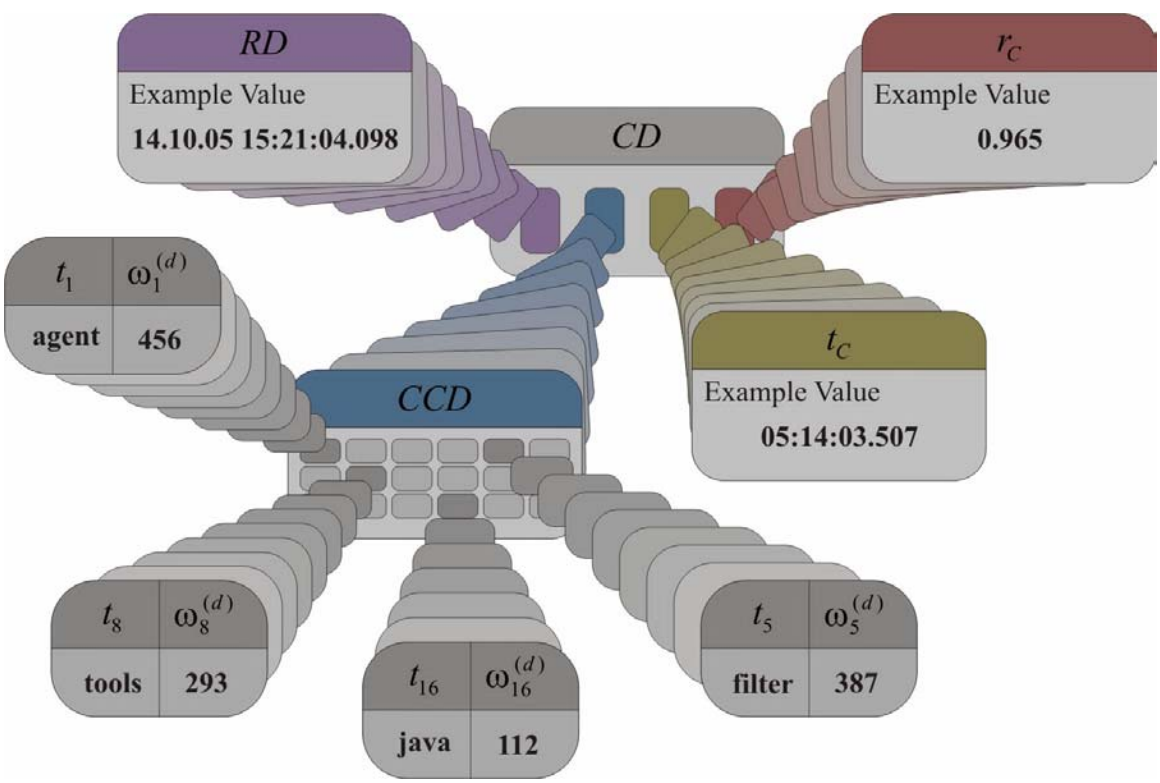

Figure 4.3: Relationship between CD, CCD, RD, t_C and r_C.

4.2. Background

With a quite rapid proliferation of the Internet in recent years, the problem of helping ordinary users discover the desired information in such an environment also continues to escalate. According to the author's best knowledge, the distributed text databases and the peer-to-peer systems try to provide solutions to the arising challenges in the information retrieval. The strengths and weaknesses of these approaches will be critically analysed in the following paragraphs and the ideas of combining them will be noted.

A task of both efficiently and accurately determining the most relevant databases with respect to a given user query is one of the main challenges for a distributed text database community [273][274][276]. The major objective of a database selection is to identify the potentially good databases that are likely to contain useful documents for the given query. That objective is in [273][274] addressed by a representative which is in advance created for each distributed database and which indicates approximately its content. In order to achieve very accurate selection, that representative contains, for each distinct term in the database, the term itself together with the reduced representation of a document that is the most important for a particular term. By reason of easily having few millions of distinct terms in each distributed database, the representative becomes very big with the size that is sometimes even larger than 100 MB [273]. That is probably the main reason in [273] [274] of completely ignoring the problem of keeping a representative to be up-to-date with the corresponding database. The solution that will be proposed in this Chapter is based on much smaller representative, having only few hundreds terms that are the most important for the corresponding database. The associated costs of sending such a small representative are quite reasonable, which makes quite possible the realisation of efficient update strategies.

As far as taking care of the dynamics of the underlying databases are concerned, the peer-to-peer (P2P) community has defined many specialised update policies [102][103]. These update policies usually assume the propagation of a change from the peer where the underlying database has been changed to all other peers [102][173]. A great work has been done in the direction of developing algorithms that can guarantee that all peers can be optimally found [102]. A used broadcast of a database representative still seems to be somewhat unpractical even for small representatives, because of too frequent updates in a highly dynamic Internet world. One step further in optimising the performances can be to send a representative only when it is needed, which will dramatically reduce the traffic in the network.

The database selection from [273][274] assumes the existence of a single point, called meta search engine, which knows everything about all other databases and which is the only one that is capable of answering queries. In a peer-to-peer world, all parties, called peers, have equivalent capabilities in providing other parties with data or services [114], and no peer needs to know everything about all other peers in order to successfully work [115]. But, although each and every peer can respond to all queries, P2P queries are usually resolved by simply propagating them to all neighbours [120], and the idea from [273][274] of selecting the neighbours is lost. The well-known Piazza system [123][124] being the system for peer data management, completely ignores optimisation issues in its small scale implementation, and almost utilises the brutal force in resolving a query. The

peerDB [194] also propagates query to all neighbours, but it additionally tries to keep in the neighbourhood only the best peers. In [194] no clear strategy is unfortunately given for maintaining such a neighbourhood, having only the best peers.

In coDB [103] hopefully so-called coordination rules [115] are used for figuring out to which neighbouring peers a query should be propagated. Composing of these rules, being similar to the semantic mappings [167] or the semantic glues among peers [124], is one of the core problems that is in the heart of several optimisation methods in data sharing networks [173]. It is consequently very often stressed that it is very hard to maintain and propagate coordination rules [219] and that their definition is application domain specific task [114], which is usually only manually done [281]. But, without solid coordination rules, the location of a needed data is very hard [122] and sharing of data becomes almost unmanageable [121]. The cornerstone idea in this Chapter is to show how update policy, being known in the P2P world as the coordination rule, can be completely automatically realised through the exchange of small database representatives in the specialised domain of a document retrieval.

4.3. Cooperation Algorithm

The fundamental cooperation property of finding for every filtering request the right communities is essentially supported by maintaining appropriate community descriptions. Every manager agent simply has the descriptions of all other communities with whom it tends to establish future cooperation activities. The important simplification is introduced by assuming that every community knows all others, which further means that the peer-to-peer propagation of the request is not necessary, and will be out of the scope of the cooperation algorithm that is going to be introduced. It will be simply assumed that the manager agents try to cooperate only for the requests that are received from a user. In the case where other manager sends the request, the selected manager knows that cooperation has already taken place, and that the request has to be locally processed.

Two cornerstone challenges, being database selection problem and information fusion are efficiently addressed by the corresponding cooperation activities. On the one hand, updating descriptions, estimating communities, and dispatching request steps serve the purpose of resolving the database selection problem. Their essential property is both the usage of quite small community content descriptions, having at most few hundreds terms, as well as learning how often every community description should be updated with new representative words, in order to optimally use the available network bandwidth. Not only that community content descriptions are much smaller that it is the case with distributed text database solution, but also a dynamic of the underlying document collection defines when the description will be again requested. On the other hand, composing results and reliability adaptation steps are responsible for providing a solution for information fusion. The major problem of the information fusion of somehow combining results, being found by different parties that utilise their unique methods for predicting the relevance values, is overcome simply by trusting to the responsible communities relatively to their reliability. Instead of retrieving the results and recalculating their relevance score by using the same relevance algorithm, the results are ranked based on their locally predicted relevance and the reliability of the responsible community. The desired final effect is the replacement of

very expensive re-examination of results, being the most often utilised way of addressing information fusion, with much more efficient result re-ranking on the fly. The adaptation of reliability values based on a provided user feedback should help find right reliabilities, and thus facilitate the better composition of a final result set.

Before presenting the previously mentioned main building blocks of the cooperation algorithm, system architecture, illustrating the communication between manager agents, is introduced on Figure 4.4. *User agent* (U) is responsible for the generation of filtering requests of jobs by collecting the user preferences. It also knows how a user feedback can be obtained and forwarded to the manager agent. *Manager agent* (M) is the cornerstone that fulfils all cooperation activities and ensures the satisfied quality of filtering services. It should be seen as the entity that first updates obsolete community descriptions and then estimates how each known community can be promising for processing a current filtering request, being shortly marked on Figure 4.4 as job. After community estimation, manager is capable to dispatch the received filtering job to the right communities. As soon as the activated communities have produced results, manager will then compose the final result set that will be returned back to the user agent. In the case of receiving any feedback from a user agent about the relevance of results, manager agent will perform the adaptation of knowledge that it has about the responsible communities.

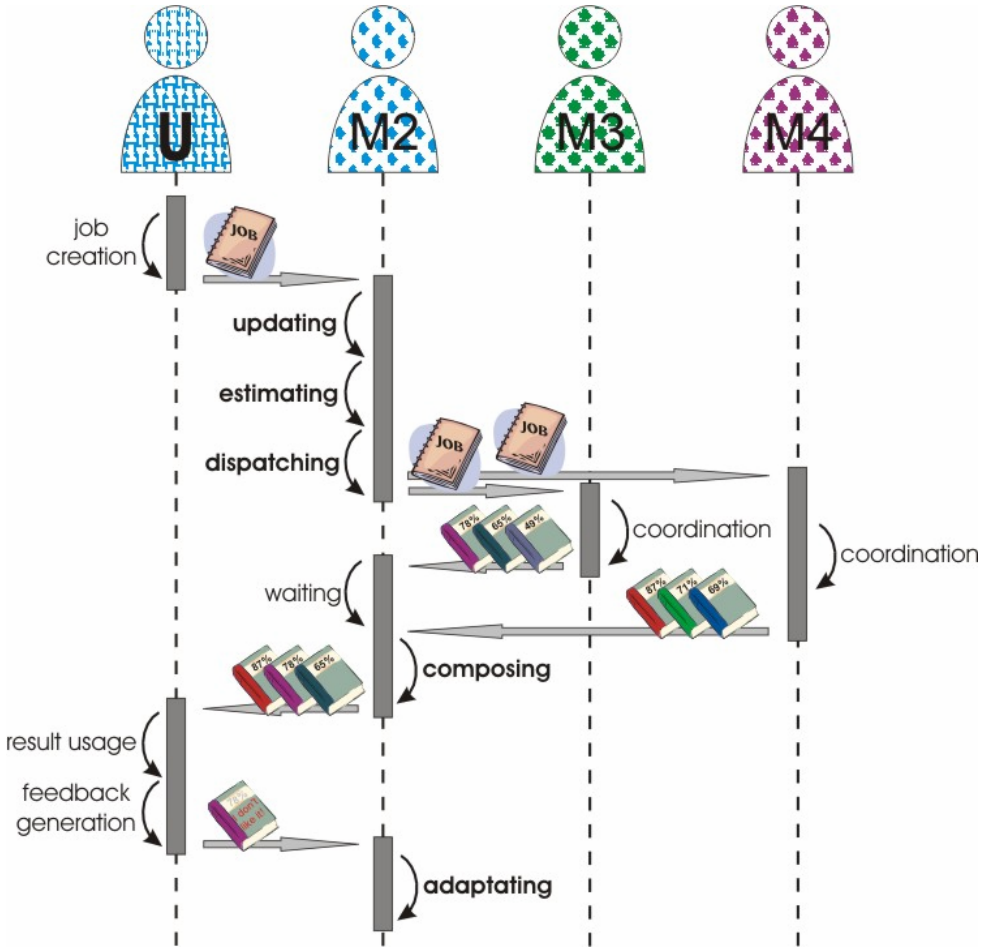

Figure 4.4: Updating, estimating, dispatching, composing and adapting cooperation steps

The ultimate goal of the deployed cooperation mechanisms is always to efficiently find which communities are the most promising for professionally providing results to the received request. This objective can be hopefully achieved through updating descriptions, estimating communities, dispatching request, composing results, and adapting reliability steps, as it is going to be shown in the following sub-sections.

4.3.1. Updating Descriptions

Each and every filtering community has the community descriptions CDs of all other communities with whom it tends to cooperate, and that are utilised for figuring out which parties are the best candidates for accepting the current request. But, before cooperation can take place, the out of date community content descriptions should be replaced with the new ones, being unavoidable since the underlying document collections are dynamic, and therefore each description is always more or less consistent with the corresponding database. This activity is shortly named as updating descriptions step, and it is separated on Figure 4.5 into obsolete check and updating single description parts. While obsolete check is solely responsible for discovering which community content descriptions are out of date, the updating description part is mainly concerned with retrieving the new content descriptions, and afterwards learning to establish the optimal trade-off between having always perfect descriptions and not loading too much the used network. Figure 4.5 shows that a community content description $CCD^{(i)}$ is out of date when its retrieval date $RD^{(i)}$ is more than $t_C^{(i)}$ away from the actual date AD (obsolete check part). It might be further clearly noticed from Figure 4.5 that the new $CCD^{(i)}$ is always obtained by simply asking the corresponding filtering community C_i to send its new content description (updating description part).

To optimise a retrieval of a community content description, filtering communities that frequently change their content descriptions, are more often asked to send their new descriptions. In other words, such communities will have smaller t_C in their CDs. It is assumed that t_C should be adapted during runtime, with a goal to learn which communities frequently change their descriptions and which do not. The used adaptation rule for t_C can be expressed as follows:

$$t_C^{(i,new)} = t_C^{(i,old)} + l(t)\left(d_J(CCD^{(i,old)}, CCD^{(i,new)}) - d_{\max}\right)^{2a+1} \tag{4.1}$$

where $t_C^{(i,new)}$ and $t_C^{(i,old)}$ are new and old period of asking for the new community content description that corresponds to a filtering community C_i, $l(t)$ is the decreasing learning rate, $d_{\max}$ gives a maximal tolerable distance between old $CCD^{(i,old)}$ and new $CCD^{(i,new)}$ content descriptions which correspond to a filtering community C_i, d_J is Jaccard index, and a is a tuning parameter that regulates a level or a speed of the underlying adaptation process.

Learning rate $l(t) = l_0 e^{-\gamma_l t}$ ensures that t_C will not be changed dramatically for the well known communities, i.e. community, being known for example as the one that does

not change its content description very often, will not change its t_C very much even in the case of providing the completely new description that differs a lot from the old one. The reasoning behind is that a fluctuation of a content description usually depends on the underlying domain, in the way that for example news domain will certainly change its description more frequently than the domain of scientific papers. Because communities usually stay bind to the same domains, author's belief is that such a learning rate will contribute in efficiently finding more realistic t_C values, which will reduce the costs of updating community content descriptions.

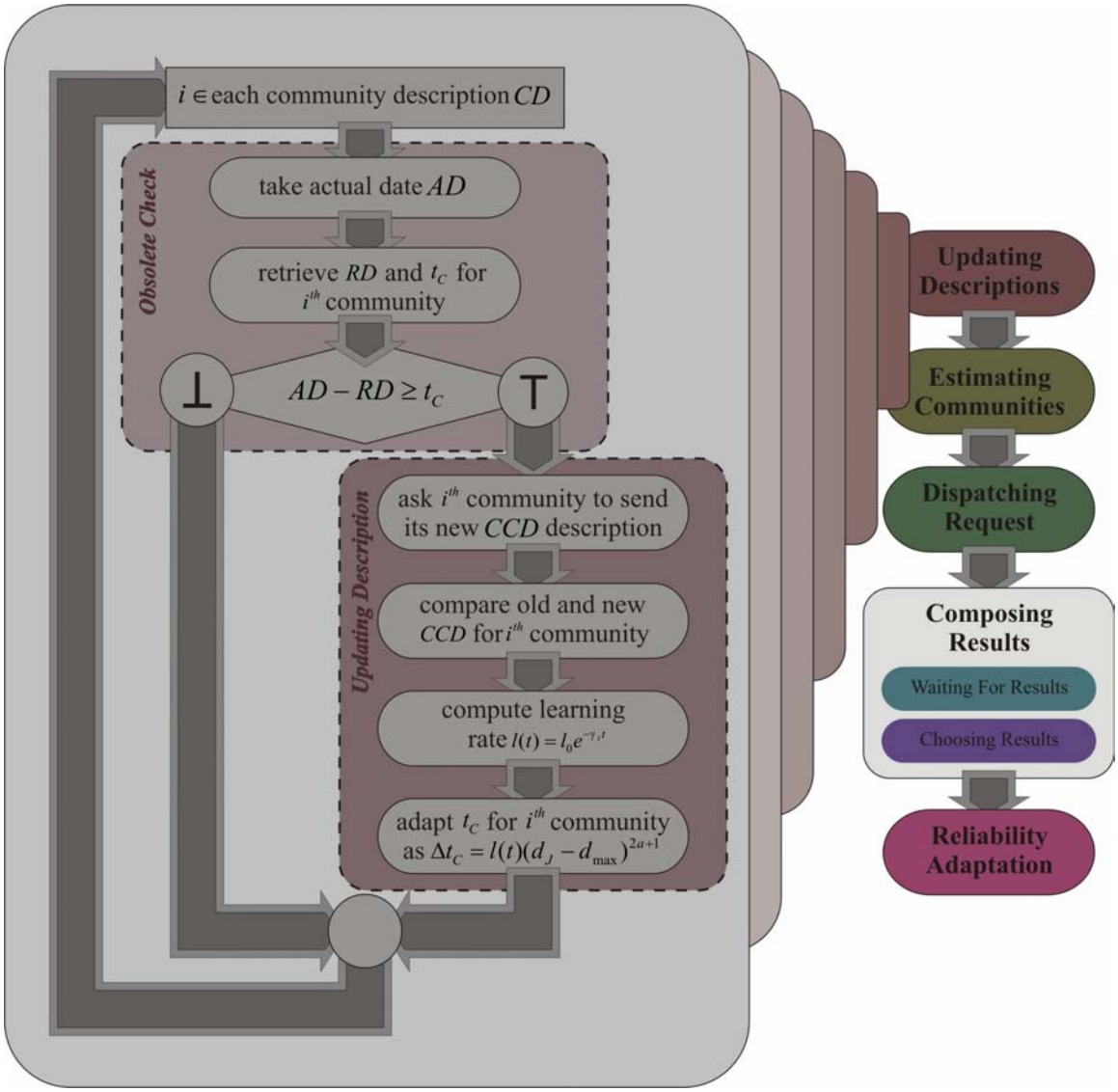

Figure 4.5. Checking which community content description should be updated because it is obsolete

Two community content descriptions $CCD^{(i,old)}$ and $CCD^{(i,new)}$ are compared by using the Jaccard index, being known as a very good metric in sparse spaces [127] and being defined as:

$$d_J(CCD^{(i,old)}, CCD^{(i,new)}) = \frac{\sum_{i \in terms\ in\ both\ CCDs} 1}{\sum_{i \in terms\ at\ least\ in\ one\ CCD} 1} \tag{4.2}$$

The computation of d_J through the given expression is very efficient because only the number of overlapping terms is computed, and at the same time, the changes in their weights are ignored. That is important because t_C has to be adapted always when new content description is retrieved, and that holds for all communities. From d_J expression, it is obvious that $d_J \in [0,1]$ and that a larger d_J means more similar objects. A maximal tolerable distance $d_{\max}$ defines how at least old and new content descriptions have to be close in order to increase t_C value. In general, very similar CCDs will impose d_J that is larger than $d_{\max}$, and consequently t_C will increase. At the other hand, quite different CCDs will have $d_J < d_{\max}$ and t_C will decrease.

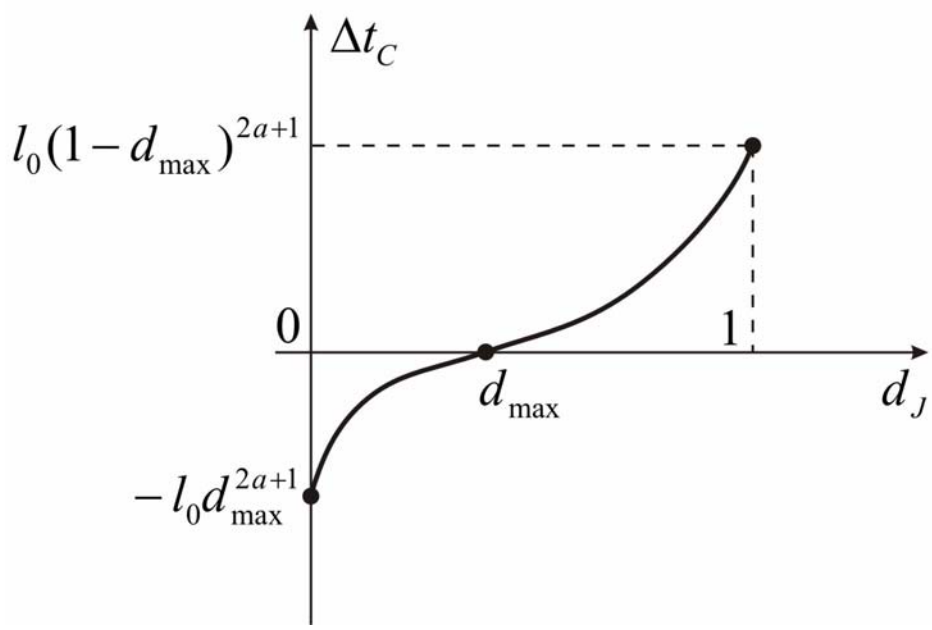

Figure 4.6: A dependence of the adaptation of a period t_C from a computed Jaccard index d_J between old and new content descriptions of a particular community.

Figure 4.6 illustrates a dependence of $\Delta t_C^{(i)} = t_C^{(i,new)} - t_C^{(i,old)}$ from d_J index, regarding a selected $d_{\max}$. As $d_J \in [0,1]$, it is necessary to choose $d_{\max}$ also from [0,1] in order for the adaptation process to work correctly. On the one hand, small $d_{\max}$ will result that Δt_C becomes negative only for the very small d_J values, which are even smaller than already tiny $d_{\max}$. Since negative Δt_C actually means decreasing the interval t_C of asking for the new content description CCD, only large differences between old and new CCD, having small resulting d_J values, will influence that a corresponding content description is asked

more often in the future. On the other hand, large $d_{\max}$ results that Δt_C becomes positive for greater d_J, corresponding to more similar CCDs. While only very similar CCDs that correspond to extremely small changes in underlying content descriptions will increase the period t_C of asking for new CCD, even not so different CCD, having not very small d_J that should be only smaller that large $d_{\max}$, will have negative Δt_C.

A final conclusion regarding $d_{\max}$ influence is twofold. On the one hand, setting small values for $d_{\max}$ leads towards designing the system, which will decrease the period t_C of asking for the new content description only when a quite big difference is detected in the obtained new CCD, regarding the already possessed old one. Such a system obviously gives a larger importance on not loading too much network than on having as accurate content descriptions as possible. On the other hand, large $d_{\max}$ results that Δt_C is more often negative and consequently t_C values, which can be expected to be obtained after many iterations, will be smaller. Since smaller t_C means more frequently asking for new CCD, setting large $d_{\max}$ tends to produce the system that will converge towards giving the advantage to having accurate CCD than to optimising the network usage.

After updating all obsolete community content descriptions, cooperation can proceed with the step, being shortly named as estimating communities, and being a topic of the next sub-section.

4.3.2. Estimating Communities

The usage of the appropriate distance function is the critical point in estimating how successful each filtering community can be in processing the actual request. Because both filtering request and community content description are defined in a highly sparse space, the appropriate modification of the weighted Jaccard index is expected to give a distance function with the desired underlying behaviour. Formally expressed, the filtering request $FR(\{t_i, \omega_i^{(r)}\})$ and the community description $CD(CCD(\{t_i, \omega_i^{(d)}\}), RD, t_C, r_C)$ might be compared as:

$$e_C(FR, CD) = \frac{\sum_{i \in both\ FR\ and\ CCD} \frac{2}{\pi} arctg(\beta \omega_i^{(d)}) \omega_i^{(r)}}{\sum_{i \in FR} \left| \omega_i^{(r)} \right|} e^{-\gamma_d \frac{AD-RD}{t_C}} \tag{4.3}$$

Since the estimation e_C should be computed for each community, the computation of e_C might be optimised, as being presented on Figure 4.7. The main idea is generally very simple, and it is contained in performing the step of analysing request only once, and in afterwards utilising the found sum value of the weights of its terms, while estimating all communities. Although the expensive computation of $arctg$ and $\exp$ functions still have to be done as many times as there are communities, the filtering request is analysed only once, being unfortunately not so obvious when only the expression for the computation of e_C values is considered.

While weights $\omega_i^{(d)}$, being present in a community description, always take only the positive values, weights $\omega_i^{(r)}$ from the request might be negative. Such negative $\omega_i^{(r)}$ are bound to unwanted terms, and always when unwanted term from a request is present in a community description, that community pays a penalty. The penalty is larger when unwanted term is more important in a community description, i.e. e_C will be reduced more when $\omega_i^{(d)}$ is larger and at the same time $\omega_i^{(r)} < 0$.

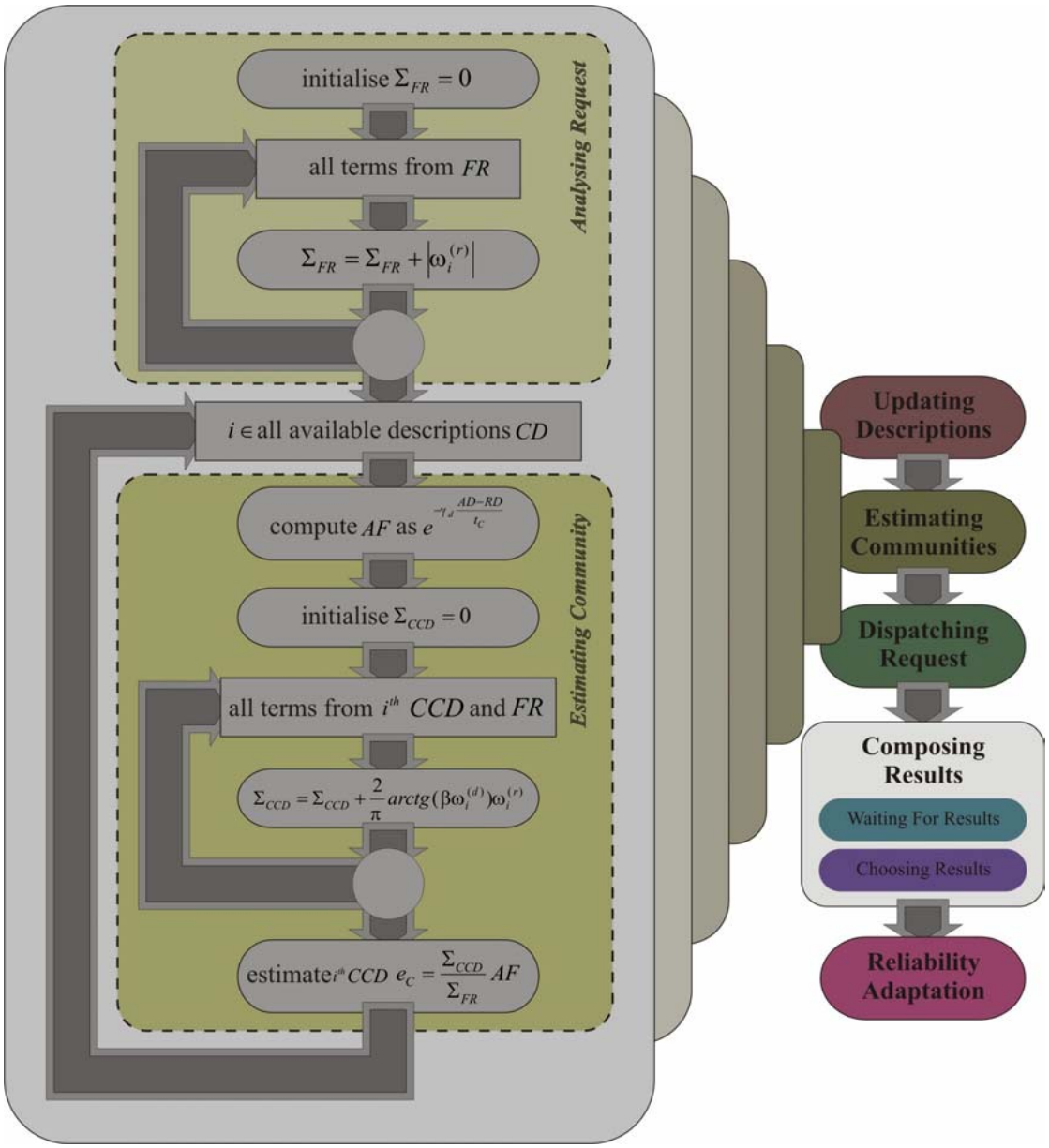

Figure 4.7: The estimation of communities also depends from the weights encapsulated in request

The introduced penalties should facilitate the successful selection of more specialised communities, always when such communities are available. In the case where unwanted terms are present in a particular community content description, the probability that good

recommendations will be found by such community is small. This holds because of the assumption that community description contains the most representative terms from the underlying document collection. Example 4.1 illustrates the influence of wanted and unwanted terms, relative to their importance both in filtering request and a community content description.

The *arctg* function is used to bound values for $\omega_i^{(d)}$ always to $[0, \pi/2]$, where a tuning parameter β controls the level of translation. Larger β means, that value $\pi/2$ is reached faster, and reverse. Such bounding is necessary because of a $\omega_i^{(d)}$ nature of representing the number of documents that have corresponding term t_i, that can be the ordinary big number. The influence of big communities, having extremely large underlying document collections, and thus owning large $\omega_i^{(d)}$ in their descriptions, is consequently controlled. Smaller communities, being usually much specialised and consequently more important, will therefore get their chances to be selected.

Denominator in e_C expression sums the absolute values of the weights of all terms in a filtering request, and consequently ensures that $e_C \in [-1,1]$, where larger e_C means that a particular community has more promising content for processing a particular request. On the one hand, it is expected that communities, having in their content descriptions a lot of wanted terms and omitting unwanted ones, will have larger e_C values. On the other hand, communities where unwanted terms dominate, and that have only few wanted ones, will have e_C values that are close to -1.

A novelty of e_C expression, mainly regarding to [10], is in taking care of the actuality of *CCD* through the inclusion of actuality factor (*AF*) as $e^{-\gamma_d \frac{AD-RD}{t_C}}$, where *AD* is actual or current system date and γ_d, $\gamma_d > 0$, defines for how much the estimation value will be decreased for communities, having *CCD* that is not completely up-to-date. In the case where *CCD* has just been obtained, *RD* is the same as *AD*, which gives $e^{-\gamma_d \frac{AD-RD}{t_C}} = 1$ and e_C is not additionally reduced. As *CCD* becomes older, the difference $AD - RD$ grows, and $e^{-\gamma_d \frac{AD-RD}{t_C}}$ drops. Finally, when *CCD* is almost ready to be replaced, the difference $AD - RD$ is very close to t_C, and $e^{-\gamma_d \frac{AD-RD}{t_C}}$ is approximately $e^{-\gamma_d}$, which is its lowest value and consequently the greatest possible reduction for e_C.

Figure 4.8 shows that maximal penalties, which not up-to-date community description is paying, exponentially drops starting from 1. The presented dependence $e^{-\gamma_d}$ actually gives which γ_d values will result in how large maximal penalty. On the one hand, small γ_d of let's say 0.25 has $e^{-\gamma_d} = 0.78$, which further means that a final estimation for one community might be at most reduced to 78% of its original value due to its not so good

actuality. On the other hand, large γ_d values, such as 1 or even 2, have $e^{-\gamma_d}\Big|_{\gamma_d=1} = 0.36$ and $e^{-\gamma_d}\Big|_{\gamma_d=2} = 0.13$, and consequently a final estimation can be in the worst case reduced to only 36% and 13% of its initial value, respectively. Setting too high values for γ_d is obviously not really advisable because the actuality factor AF starts to dominate. Since its main purpose is to perform fine tuning of a final estimation for a particular community based on the actuality of its description, the recommended values for γ_d are those that are usually smaller than 0.5 for which $e^{-\gamma_d}\Big|_{\gamma_d=0.5} = 0.6$.

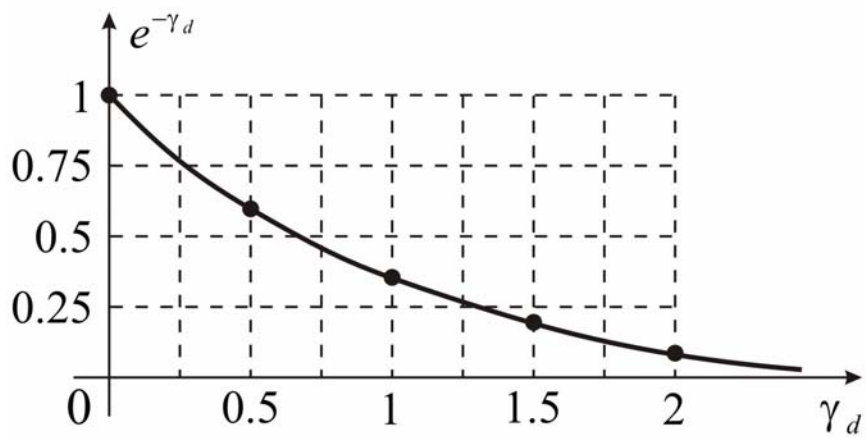

Figure 4.8: A guideline for selecting the right γ_d value that will result in the desired maximal value $e^{-\gamma_d}$ of the actuality factor $e^{-\gamma_d \frac{AD-RD}{t_C}}$, which defines penalties by reason of being not up-to-date.

4.3.3. Dispatching Request

The main cooperation objective is to dispatch the actual request only to the potentially good filtering communities, being the ones that both have the access to the most relevant needed information and at the same time have the acceptable reliability in processing past filtering tasks. While the solid idea about which communities have the best underlying documents can be assessed through the usage of in the previous step found e_C values, each community description CD additionally has the reliability r_C, showing how good the corresponding community has performed in the past.

The used dispatching principle is to send the current request to k communities, which have the best combination of e_C and r_C values, being a straightforward way of escaping the unnecessary loading of not promising enough communities, as well as not wasting the network bandwidth for sending the request to parties that have insignificant qualifications. The solution of combining e_C and r_C values from [10], is to define so-called community promise-ness p_C as:

$$p_C = \frac{\beta_e e_C + \beta_r r_C}{\beta_e + \beta_r} \tag{4.4}$$

where tuning parameters β_e and β_r control the influence of e_C and r_C in making a final judgement about how promising is a particular community. According to the fact that e_C and r_C take values from $[-1,1]$ and $[0,1]$, respectively, it follows that $p_C \in [-1,1]$, where bigger p_C means that a particular community is more promising.

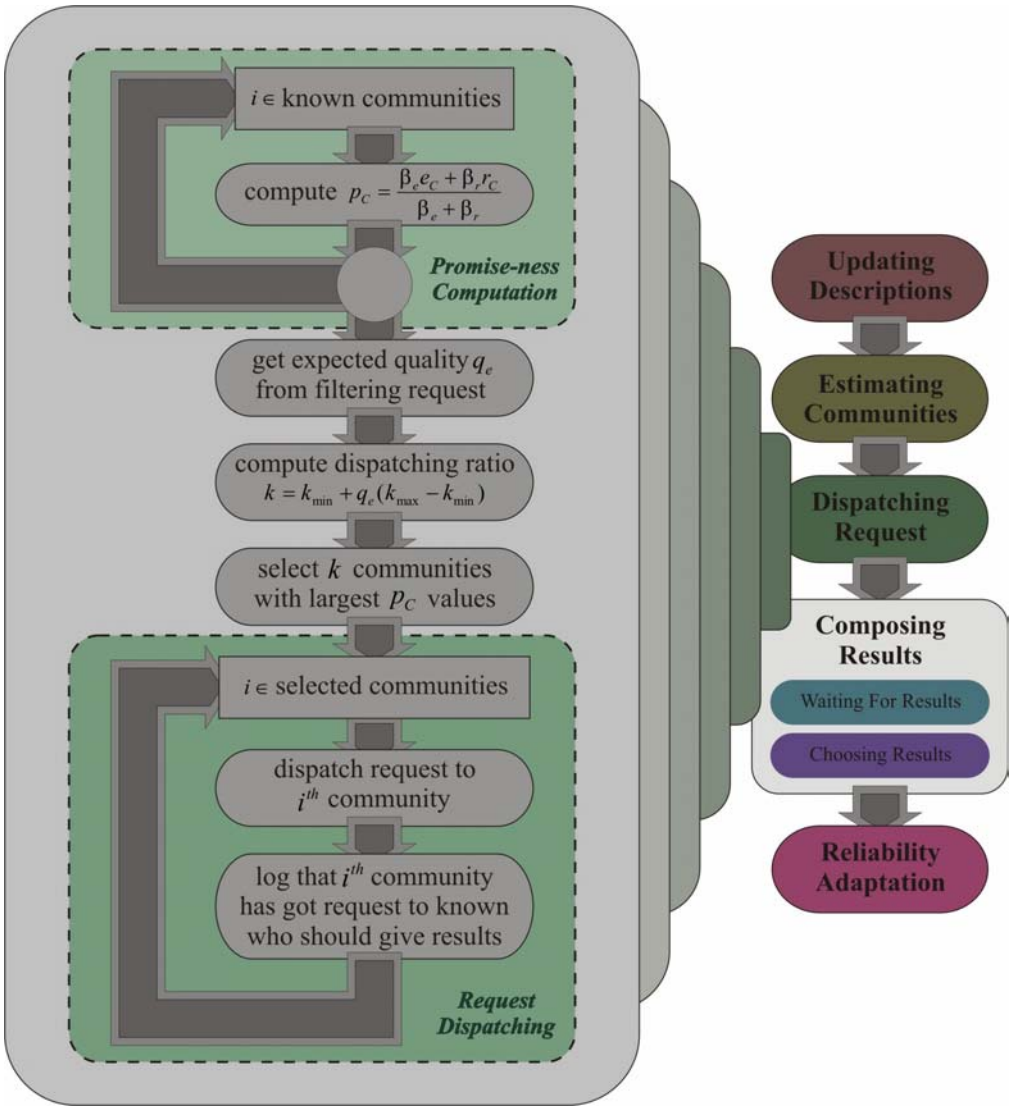

Figure 4.9: Deciding to how many communities to dispatch request depends on the expected quality that a user set in its request, where higher expected quality means that more communities will work

The novel idea, respecting to [10] is to take care of the quality of results q_e, $q_e \in [0,1]$, being expected by a user through making k to be the growing function of q_e, i.e.

$k = k_{\min} + q_e(k_{\max} - k_{\min})$. While in the case where the high quality is expected, q_e has value 1 and consequently $k_{\max}$ communities are activated, whereas in the opposite case $q_e = 0$ and $k_{\min}$ communities ensure that at least somebody tries to find recommendations for a request where the high quality is not an ultimatum. Such dispatching principle will obviously load many communities only when very high quality of results q_e is expected with a heuristic behind that more communities can at the end find better results.

The detail workflow of the deployed dispatching activity is algorithmically presented on Figure 4.9. The first loop is concerned with computing the promise-ness for each and every community for which the estimation value is already available. The number of the communities k that will be activated is afterwards computed by retrieving the expected quality q_e from the received request. The second loop finally dispatches the request to k communities with the largest promise-ness, and logs to whom the request has been sent. This logging is necessary for knowing until when the expected results should be waited for, and being able to return them not only when the time-out occurs, but also when every queried community has responded.

Example 4.1: Illustrates computation of e_C and p_C values under assumption that each CCD is fully up-to-date, i.e. the difference $AD^{(i)} - RD^{(i)} = 0$ for $i \in \{1,...,5\}$.

Let's compare $FR(\{(clustering,0.9),(filter,0.2),(neuro,-0.7)\})$ with five community content descriptions CCD. The parts of these content descriptions which are important for e_C computation are given in Table 4.1. As it was expected, the best e_C values are obtained for communities C_1 and C_4 that do not have in its descriptions unwanted $(neuro,-0.7)$ term. But, the decision to which communities request should be dispatched is also based on community reliability, and because reliability for C_3 is much higher than it is for C_1, the chosen communities are C_3 and C_4, and they are marked in Table 4.1. with a different cell filling. While the former has remarkable successfulness in satisfying user needs in the past, the latter has a content description that is very similar with the request.

Table 4.1: Comparison between $FR(\{(clustering,0.9),(filter,0.2),(neuro,-0.7)\})$ and 5 given community descriptions, and computation of e_C and p_C values, in the case where $\beta_e = \beta_r = \frac{1}{2}$.

C_i	CCD	e_C	r_C	p_C
1	{...(*clustering*,189)...}	0.49	0.32	0.405
2	{...(*neuro*,87)...}	-0.38	0.76	0.19
3	{...(*clustering*,57)...(*filter*,278)...(*neuro*,123)...}	0.21	0.89	0.55
4	{...(*clustering*,38)...(*filter*,198)...}	0.60	0.57	0.585
5	{...(*filter*,123)...(*neuro*,18)...}	-0.26	0.56	0.15

4.3.4. Composing Results

The fact, that multiple communities work together, comes to the point when the results, being produced by different communities, should be put together in order to create the unique set of recommendations. A community, having received the original request and having initiated a cooperation process, will collect all the results, being found by different communities, and will decide which filtering results are valuable enough to be returned as recommendations. In that way the performed cooperation is completely transparent for the sender of a filtering request, i.e. a user does not think about where the retrieved data were originally deployed.

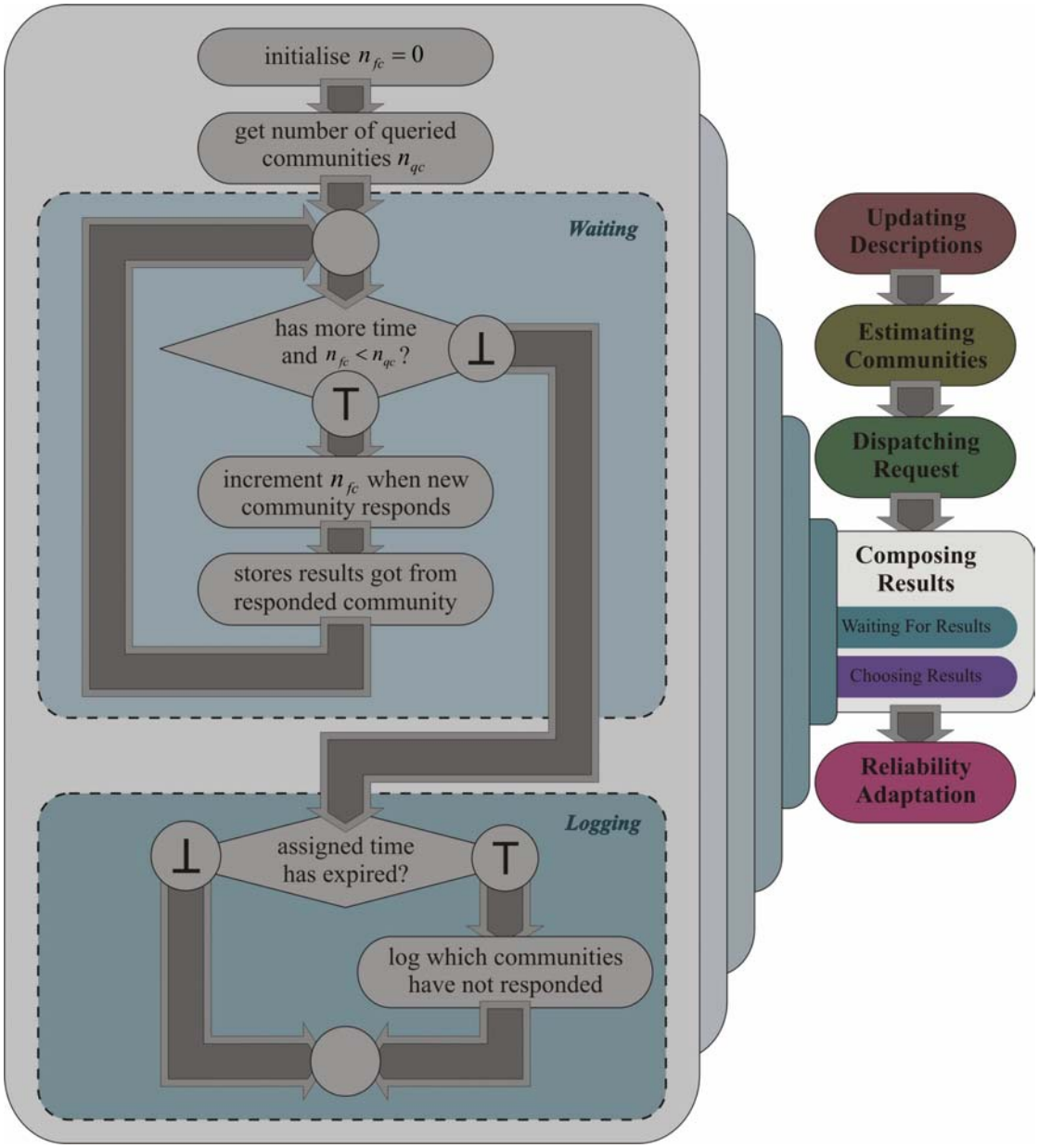

Figure 4.10: Waiting for the queried communities to respond and logging in the case when somebody has not managed to provide the requested results in the permitted amount of time

The overall activity of composing results is naturally separated into two sub-activities, named waiting for results and choosing results. The first one in presented on Figure 4.10, which gives the underlying logic for waiting, and also specifies that it should be logged which communities have not responded in an assigned amount of time. Instead of waiting forever for every queried community to respond, the installed logic limits the maximal waiting time. The reasoning behind lies in the natural assumption that it is probably much better to exploit the available results from already responded communities and to deliver something to the user after the reasonable response time than to even generate better final result set, but after an unacceptable delay. Users are impatient beings, and leaving them to wait too long might mean that they become quite suspicious regarding the deployed filtering engine.

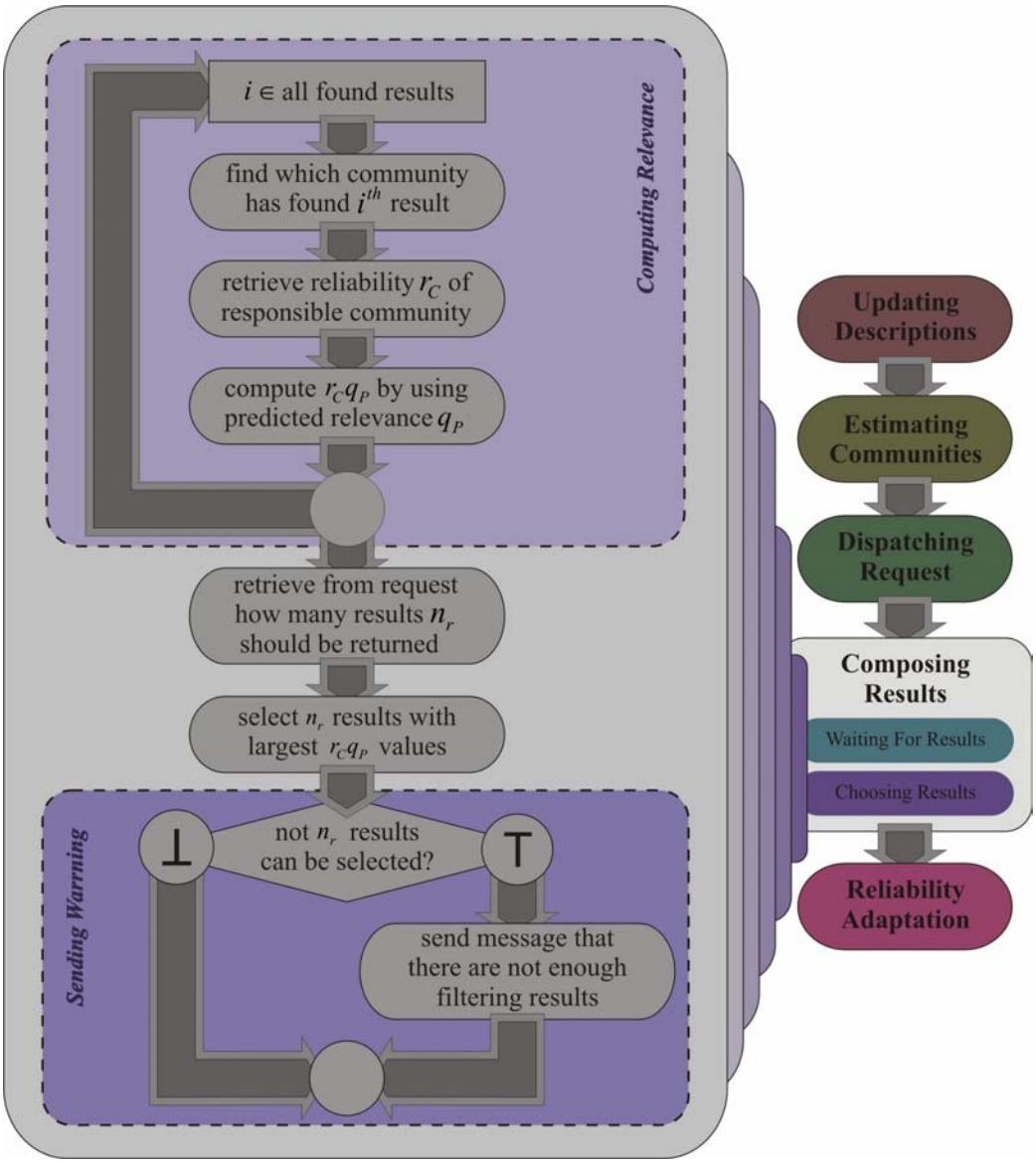

Figure 4.11: Selecting the given number of filtering results based on the locally predicted relevance of a result and the reliability of the community that has found that filtering result

It is assumed that each result comes together with the predicted quality q_p, showing how good is a particular result, and being the number from $[0,100]\%$. These qualities are set by the communities that have found the corresponding results, and consequently the community that is putting results together does not have any influence on them. In order to protect from the malicious communities, saying that their results are always the perfect ones, the community reliability r_C is also taken into account when composing results. Instead of ranking results based only on their quality q_p, a product $r_C q_p$ is used to better assess the real quality, and the asked number of results with the largest $r_C q_p$ values will be chosen. Filtering results, being found by the communities with the low reliability, will actually pay penalties, and thus reduce their chances to be finally included in the closing recommendation set.

Choosing results, being the second sub-activity of composing, is given on Figure 4.11. The most important part is contained in the first loop, which computes the product of the predicted result relevance q_P, being found by the queried party, and the reliability r_C of that responsible community. The actual filtering request is afterwards used to figure out how many results n_r are expected by a user. The n_r results with the largest $r_C q_P$ values are finally selected, and in the case where fewer results are available, a warning message is sent. A typical message may for example specify that even though 20 results have been expected, only 14 will be finally delivered by reason of not having the unreasonably long response time.

Example 4.2: Illustrates the process of deciding which results, found by communities chosen in Example 4.1, are good enough to be returned to a user.

Table 4.2: Composing of final result set that will be returned to a user, based on locally found relevance of results and the reliability of the responsible filtering community

C_i	r_C	i	$q_p[\%]$	$r_C q_p$
3	0.89	1	78	69.42
		2	65	57.85
		3	49	43.61
4	0.57	4	87	49.59
		5	71	40.47
		6	69	39.33

Let's suppose that communities C_3 and C_4, being chosen as the most promising communities in Example 4.1, have found all together six results and have predicted how relevant these results can be to a user. The predicted relevance q_p values for these six results are given in Table 4.2 in a column marked as $q_p[\%]$. In the case where three results should be returned to a user, the best results, having the largest $r_C q_p$ values, are the ones with $i = 1,2,4$, and the corresponding rows in Table 4.2 are filled with different cell filling. One should remark that results with $i = 5,6$ are not chosen, even though they

have bigger predicted quality q_p than a chosen result with $i = 2$, because they have been found by a community C_4, having lower reliability r_C value.

4.3.5. Reliability Adaptation

After completing a recommendation set and receiving a user feedback about the actual relevance of found results, learning through reliability value adaptation takes place in order to ensure that the assigned reliability value r_C reflects as accurate as possible the corresponding ability of a filtering community to satisfy the imposed information needs. The adaptation of r_C is based on the comparison between by the filtering community predicted result relevance q_p and the actual relevance q_a, generated from the obtained user feedback (Figure 4.12). The used adaptation rule can be expressed as:

$$\Delta r_C = \gamma_C l(t)(\varepsilon - |q_a - q_p|)^{2k+1} \tag{4.5}$$

In Δr_C expression ε is a tolerance, which defines how close the predicted relevance of results should be to the actual relevance in order to reward the responsible filtering community, k $(k > 0)$ increases the influence of large q_a deviations from q_p, γ_C is a tuning parameter, and $l(t) = l_0 e^{-\gamma_t t}$ is a decreasing learning rate which insures that already learnt reliability value will not be easily destroyed. The filtering communities with the solid history will, according to such a learning rate, change their reliability values in a smaller extent than the novel ones.

Figure 4.12 gives the procedure that is applied on each received feedback value. The first task to be done for each and every user feedback, bringing the actual relevance q_a, is to find out which community has been responsible for that result. It is afterwards possible to get the predicted relevance q_P of that result, as well as to adapt the reliability value r_C of that community. The procedure for processing the received feedback values ends by propagating user ratings to other known communities always when such configuration setting is used. Such a propagation of feedback values is mainly necessary to support the learning inside every single community, where the responsible manager tries to select as often as possible the best available filtering agent. These learning efforts make the basis for coordination activities, which are the cornerstone of this Thesis, and which will be in detail analysed in Chapter 6.

While a reward is limited to $\gamma_C l_0 \varepsilon^{2k+1}$ and corresponds to the case where q_a is exactly the same as q_P, a penalty for really bad estimations of q_p, being quite different from q_a, is theoretically unlimited. In reality, penalties are also limited because both q_a and q_p take values from $[0,100]\%$, which gives $|q_a - q_p| \leq 100\%$, and therefore the largest possible penalty will be at most $\gamma_d l_0 (\varepsilon - 100)^{2k+1}$. The limitation concerning the maximal $|q_a - q_P|$ value also influence the logical range for a tolerance ε, which has always to be from $[0,100]\%$.

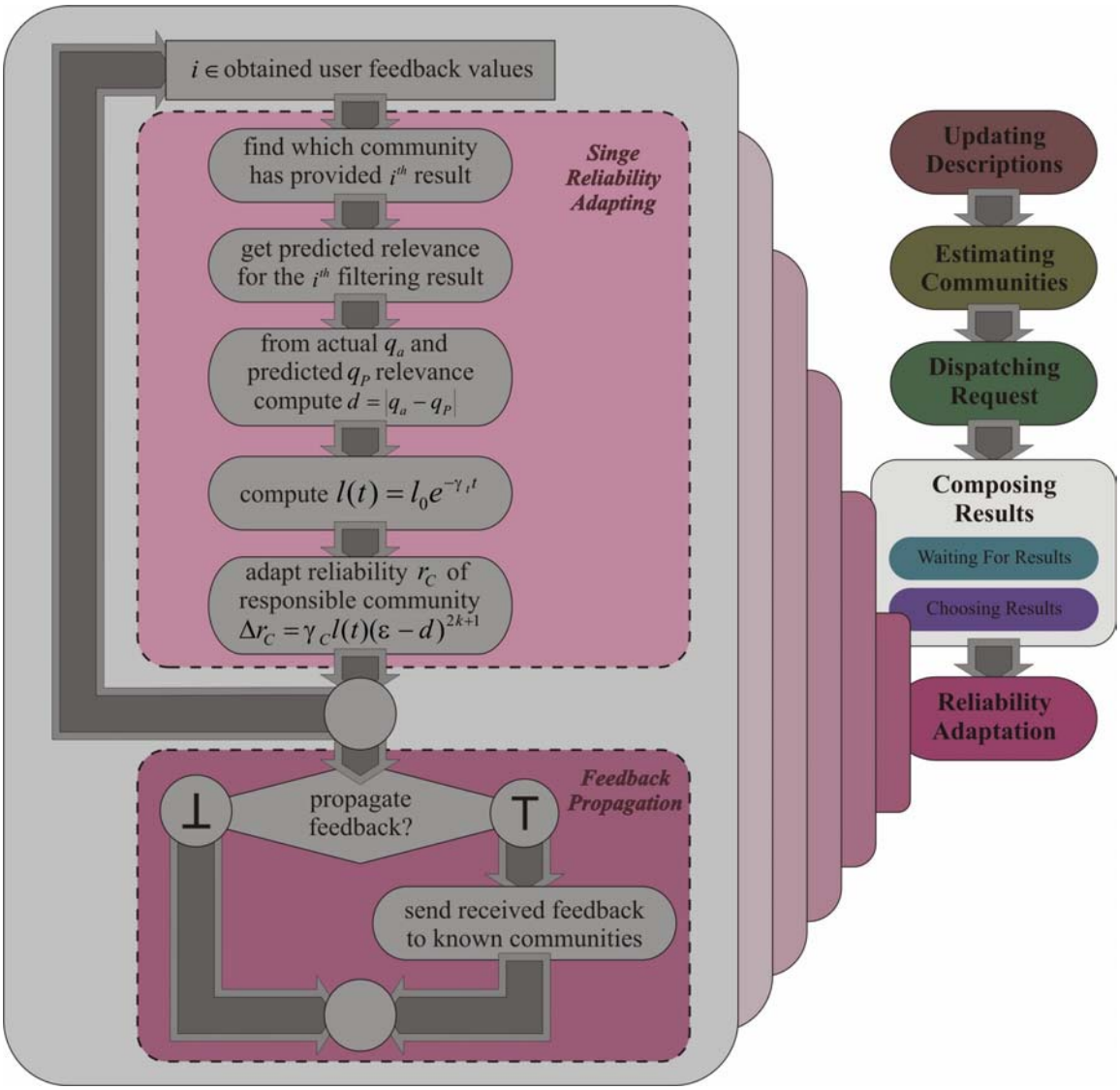

Figure 4.12. Adaptation of community reliability r_C value and propagation of feedback values

The influence of the value, being used as a tolerance ε, can be best assessed from the Figure 4.13, which illustrates a dependence of Δr_C from $|q_a - q_P|$. On the one hand, small tolerance ε has as a consequence that only communities, whose predicted relevance q_P deviates only a little from the actual relevance q_a, will be rewarded by increasing their reliability r_C. This holds because Δr_C is going to be positive only for $|q_a - q_P|$ being even smaller that than already small ε. Such a small setting for ε, will obviously lead towards an adaptation, which is very restrictive when giving rewards, and which prefers punishing communities. On the other hand, large ε results that most of the time Δr_C is positive, even in the cases where q_a differs a lot from q_P. A system with a large tolerance ε will therefore prefer rewards, being the opposite behaviour regarding the case with the small ε value.

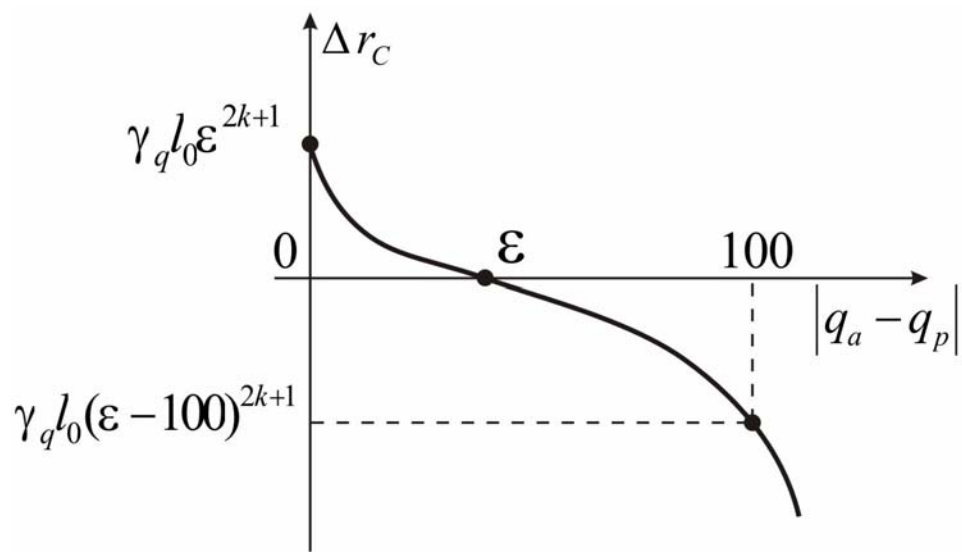

Figure 4.13. A dependence of community reliability adaptation from a deviation of q_a from q_p

4.4. Conclusion

The efficient cooperative solution for well-known database selection and information fusion problems has been introduced in this Chapter. The presented approach is based on the assumption that communities know each other, and that the peer-to-peer propagation of a request is not necessary. Even though such an assumption is logical for architectures having five to ten filtering communities, the natural extension of the presented approach might be towards assigning the well-known time-to-live parameter to each request. The simple underlying application says that always after the request is dispatched, its time-to-live value will be decremented, and any future cooperation will be possible only in a case where the time-to-live parameter has a value, being larger than 0.

The unique characteristic of the proposed cooperation approach has been in the usage of small community content descriptions for addressing the database selection problem. These small descriptions have been mainly introduced to minimise the needed network bandwidth for their transmission. As an efficient usage of the available network resources becomes critical for many modern distributed systems, the transmission of these content descriptions has been further optimised by learning how often every description should be retrieved. A final attempt of not loading too much underlying network has been made by establishing the advanced estimation and dispatching principles, which will forward the request only to promising communities, and not to all nodes, as being typical in peer-to-peer world.

The usage of resources has been finally optimised by proposing a very cheep solution for overcoming the information fusion problem. Instead of performing the very expensive re-examination of results in order to create the uniform relevance values at the same way, the locally predicted relevance has been combined with the reliability of the community, which has computed that relevance. The reliability adaptation step has been designed to learn the reliability values, which represent as close as possible the available capabilities of the corresponding communities.

Chapter 5
Exploration Approach

For each and every request a filtering engine has already received hundreds of slightly different requests, being concerned with satisfying more or less similar information needs. The cornerstone idea is to utilise the strengths of these similar requests to adapt the actual filtering request, and to force it to intelligently explore only relevant information areas. While users traditionally always expect to get the best possible filtering results that could be found in the underlying collection, they unfortunately have a highly different expertise, being fundamental for the successful specification of the information needs. Only the tiny fraction of users belongs to a group of domain experts that know how to query to be well understood, and therefore really many requests, originating from the novice users, will be poorly formed. Such requests will unfortunately most probably lead towards the results with at best the average quality, and really many inexperienced users will have to search somewhere else, where they will be better understood.

To keep the huge majority of users, who desperately need any help in satisfying their information needs, the multi-domain applicable exploration approach will be presented in this Chapter. The offered help is going to be in the frame of intelligent query processing techniques, which generally aim at returning not only what has been explicitly asked, but also at trying to efficiently discover the information, which might be potentially useful and interesting [127]. The simple straightforward way of performing the intelligent query processing is by adapting the actual request in the frame of the explicit communication with a user, who is accepting or rejecting the advices about the possible query refinement. Although such an approach might produce the excellent results, it requires sometimes the heavy assistance of a user. In the nowadays highly dynamic information societies, a user attention has definitely become a precious resource, and thus an explicit communication with the user will not be in the basis of the proposed exploration approach.

The yet another technique for achieving the intelligent query processing without user interventions, which is going to be the foundation for the intelligent exploration, is based on silently utilising the experience from the already successfully fulfilled past requests, without explicitly saying anything to the user. Such an approach, being already published in [6], should be able to deliver the surprisingly good and relevant filtering results, as it is going to be shown in the remainder of this Chapter, being structured as follows. The next section defines the most important terms, and gives the other necessary preliminaries, to make possible the concise presentation of the novel, multi-domain applicable, exploration

approach. After critically presenting the major drawbacks of other known strategies to provide exploration, a cornerstone of this Chapter is contained in the section, which gives main design principles, being behind the proposed intelligent exploration. Because of its complexity, this section is separated into searching for usable old requests, using found similar requests, and adapting actual requests sub-sections. The Chapter is subsequently finished with the short concluding remarks, summarising the properties of the proposed universally applicable exploration approach.

5.1. Definitions and Preliminaries

A multi-domain applicability of the exploration approach requires that filtering request has to be defined on a more general way than it has been done by the Definition 4.5 in the Chapter where cooperation approach has been given. The novel definition for the filtering request is based on attribute, and consequently attribute and filtering request terms, which will be used only in the course of this Chapter, will be introduced as follows:

Definition 5.1: *Attribute* a_i is a pair (v_i, ω_i), where v_i encapsulates its value and ω_i its weight or importance. On the one hand, in the domain of text documents, it is usually enough to known which word is represented by a particular attribute, and neither its value nor importance is necessary. A typical example is a query that is sent to search engines, and that contains only words without values and weights. The more restrictive application, such as the one being described in Chapter 4 with the cooperation approach, assumes that each word has associated importance. This additional dimension is used to more precisely define how important a particular word is for a given user. On the other hand, there are many applications, such as for example a domain of recommending holiday offers, which require the yet additional dimension for specifying each attribute. It becomes reasonable to define for each attribute, corresponding to the available holiday facility, such as riding, sauna, bowling, swimming-pool, and so on, not only the desired value like four stars, but also its importance, saying for example that it is very important for a given user that four stars are given to the riding facility.

Definition 5.2: *Filtering Request* $F_r \equiv F_r(a_{i_1}, a_{i_2}, ..., a_{i_k})$ is a collection of k attributes $\{a_{i_1}, a_{i_2}, ..., a_{i_k}\}$ where $\{a_{i_1}, ..., a_{i_k}\} \subset \{a_1, ..., a_n\}$. On the one hand, a number of all possible attributes n corresponds for example in the domain of text documents to the size of the dictionary, having millions of different words. On the other hand, requests usually do not have many attributes, or stated equally k is usually not very big. A typical query sent for example to a search engine has only few words. It is thus reasonable to state not only that $k \leq n$, but also that usually $k << n$.

In the course of this Chapter the alternative notation FR for the filtering request will be also utilised in algorithms due to visibility reasons. In the given formulas actual and expected filtering request are marked as $F_r^{(a)}$ and $F_r^{(e)}$, respectively. In algorithms, the same notation is based on stating explicitly either *actual* FR or *expected* FR, having the much better visual distinctiveness.

Figure 5.1 gives one example of the filtering request, being defined in a relatively low dimensional space with $n = 8$ attributes, corresponding to different facilities in a domain

of finding holidays. The given request has defined only $k = 4$ attributes, named as food, sauna, pool, and sport. Every attribute has associated corresponding value and importance, and for example the attribute food goes with a preferred value of 4.56 and an importance of 0.97.

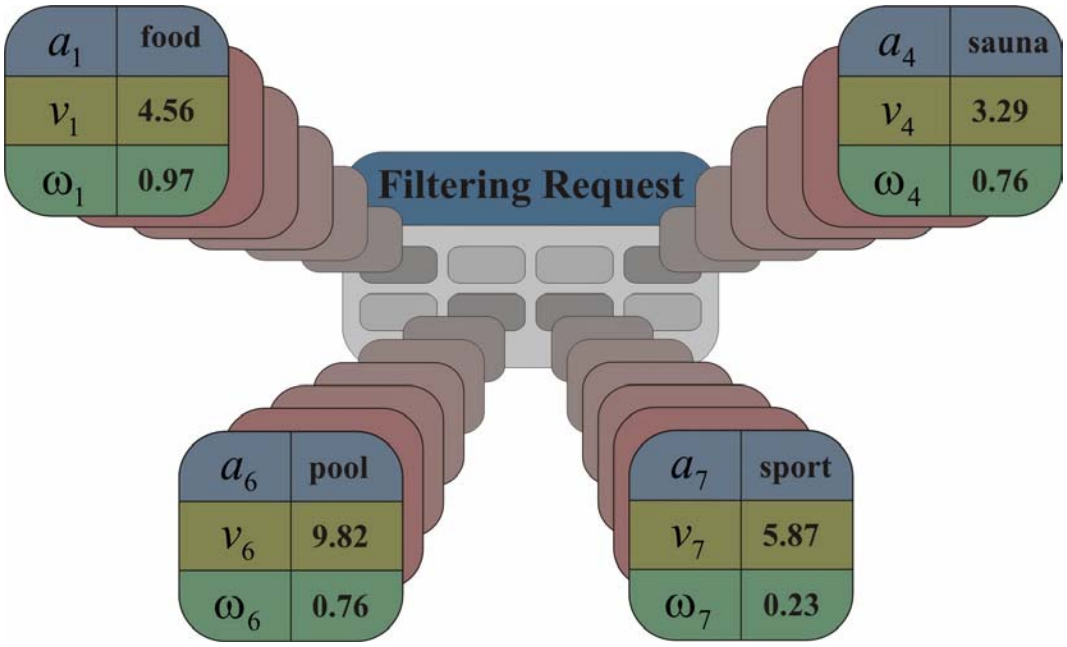

Figure 5.1: Example of a filtering request having defined 4 attributes from 8 possible

Definition 5.3: *Exploration* is an activity performed inside a single manager agent that has agreed to locally, inside its own community, process the actual filtering request. This Chapter is focused on the presentation of the exploration approach, being contained in the mining of other already processed requests that should help improve the actual one, and make it to more intelligently explore the relevant areas of users' interests.

5.2. Background

The very high importance of helping to the user to explore its information needs has resulted in many different ways of addressing these exploratory issues. Up to the author's best knowledge the four main directions for achieving exploration are going shortly to be critically presented in the following subsections.

5.2.1. Collaborative Filtering

An obvious way to achieve exploration is to use filtering strategy that is exploratory in its nature. One such filtering technique is collaborative or social one [30][76][116][213][221], which is based on an assumption that items, being liked by similar users, are good to be recommended. The well known problem with the collaborative filtering is its poor explanation power [131], being very important when more expensive items should be recommended. Somebody can hardly be satisfied with an explanation that similar users like the particular holiday destination, and that is a single reason why it is recommended.

Another problem with pure collaborative filtering lies in its great difficulty to control exploration. Even though a user shows its interests in articles from the particular domain, collaborative filtering can find recommendations in completely unrelated domains only because similar users like them. In a situation where the concurrent sites are only few mouse clicks away, such the behaviour of collaborative filtering is unacceptable, because many disappointed users can easily switch to another recommendation engine.

Because collaborative filtering is treating each item as a black box, the presented problems can be hardly solved only by using pure collaborative filtering. The way to overcome these drawbacks is to utilise collaborative filtering not as single, but only as an auxiliary exploratory engine, as it is going to be shown in this Chapter.

5.2.2. Skyline

For an efficient processing of requests where a user has not specified the importance of attributes, a skyline technique [55] can be applied. Skyline is a collection of items that are not dominated by other items, where it is said that one item is dominated when there is another item with better attribute vales. For example in the case where two analysed attributes are hotel price and distance from the beach, because obviously smaller attribute values are preferred, a particular hotel with a given price is not dominated by others only when it is the closest to the beach.

In exploration a skyline is used in the way that only items being members of a skyline should be recommended, where the various parts of a skyline correspond to the different importance of selected attributes for which a skyline is formed. In a given example after determining that a particular hotel is too expensive the next one, which is going to be recommended, is the cheaper one that belongs to skyline, being analogously to increasing the importance of a price attribute.

Although skyline sounds very promising for finding the best importance for attributes in a received request, its authors have pointed out that in spite of utilising the advanced spatial indexing techniques, such as the different variations of R trees [108], skyline is practically applicable solely for the requests with only few attributes [152]. Because of an intention to develop the exploration mechanisms, which are applicable on requests with hundreds of attributes, being something quite normal in a domain of recommending text documents, exciting skyline approach cannot be taken into serious consideration.

5.2.3. Similarity versus Diversity

Under the assumption that the received request perfectly describes the user wishes, very recently few techniques have been developed that take care not only of the similarity between chosen items and the request, but also of the diversity of items that are going to be recommended [185]. One such technique always tries to maximise the product of the similarity between a candidate item and a request, and dissimilarity between a particular candidate item and already chosen items that are going to be recommended. Obviously, increasing a diversity decreases similarity which imposes that similarity-diversity trade-off is unavoidable.

A similarity preserving technique [243] is developed that addresses the problem of the mentioned trade-off. This technique will increase diversity of chosen items only if their similarity to the request is preserved. In reality insisting that similarity is fully preserved

unfortunately usually means that achieved diversity cannot be increased, which limits its application. This is the main reason together with the initial assumption that a request always perfectly corresponds to user preferences, why exploration challenges are going in this Chapter to be addressed on a different way.

5.2.4. Mutation & Crossover

In evolution strategies [181][269], the recombination techniques are used to change a genetic material of individuals by either mutation or crossover in order to ensure a solid diversity of a population, and to protect a system of being trapped in only locally optimal solutions. While the mutation is contained in randomly changing a single individual, the crossover operator tries to exchange the genetic material among good individuals, who possess high fitness values by reason of having shown fine behaviour in the past. There are numerous existing systems, such as Amalthaea [188], NewT [235] and PEA [269], which use mutation and crossover recombination operators to achieve the exploration in a domain of information filtering, and adapt to the constant changes of user's preferences. Common to all these systems is both the modelling of the user's information interests by multiple profiles, each playing the role of one individual in a population, and having a fitness that depends on received user ratings, as well as searching for optimal profiles by applying mutation and crossover operations among them.

The usage of mutation and crossover operators for achieving the exploration through a query expansion in the case of one recommendation engine similarly means that before deploying any particular filtering strategy, a request is modified by both applying small random changes on its attribute values, as well as adding additional attributes from the old requests, which are at the same time successful by reason of having produced good results in the past. The problems of such an exploration are unfortunately twofold. Since a request is randomly changed by a mutation, it can even worse describe user preferences. Furthermore, the crossover adds attributes from successful old requests, which are maybe dealing with the completely different subject matter, and which might force the undesired exploration of completely unrelated areas. The exploration technique, based on mutation and crossover, can be consequently treated as a blind one, whereas the generic approach from this Chapter will show how attributes can be intelligently changed to increase the chances of a request to be a successful one.

5.3. Exploration Algorithm

The cornerstone property of the realised exploration approach of using the knowledge, being encapsulated into processed old filtering requests, is essentially supported by first searching for the usable old requests, then by mining the found old requests, and finally by adapting the actual request. Finding the usable old requests is actually the k-nearest search, which utilises the specially designed distance function that is the combination of Euclidean distance and Jaccard index, and which is discussed in Section 5.3.1. Using the found old requests is based on summarising the information about the different attributes from a formed similar neighbourhood, and is a main topic of Section 5.3.2. Section 5.3.3 finally gives the adaptation of the actual request, being based on either adapting or adding attributes that are important in a neighbourhood, and that might bring improvements. The

proposed controlled both the adaptation and the addition of attributes represents the main contribution of the whole this Chapter.

The shortly mentioned main exploration building blocks, which will be the main topic in the remainder of this Section, are illustrated on Figure 5.2. A scenario begins by a *User agent* (U) that creates request or job, and sends it to any known *Manager* (M). The M_2, being initially selected by a user, afterwards performs the necessary cooperation activities, which result in dispatching the job to M_3 that is better suited than M_2. The exploration steps, being shortly named on Figure 5.2 as searching for old requests, using similar old requests, and adapting new request, are subsequently executed to improve the actual job. After finishing the intelligent exploration, manager M_3 proceeds with a coordination that will find the requested filtering results with the associated relevance values of 78%, 65%, and 49%. The scenario ends by forwarding the found results to user U via M_2, deciding through the adequate cooperation activities that the result with a relevance of 49% is not valuable enough to be returned.

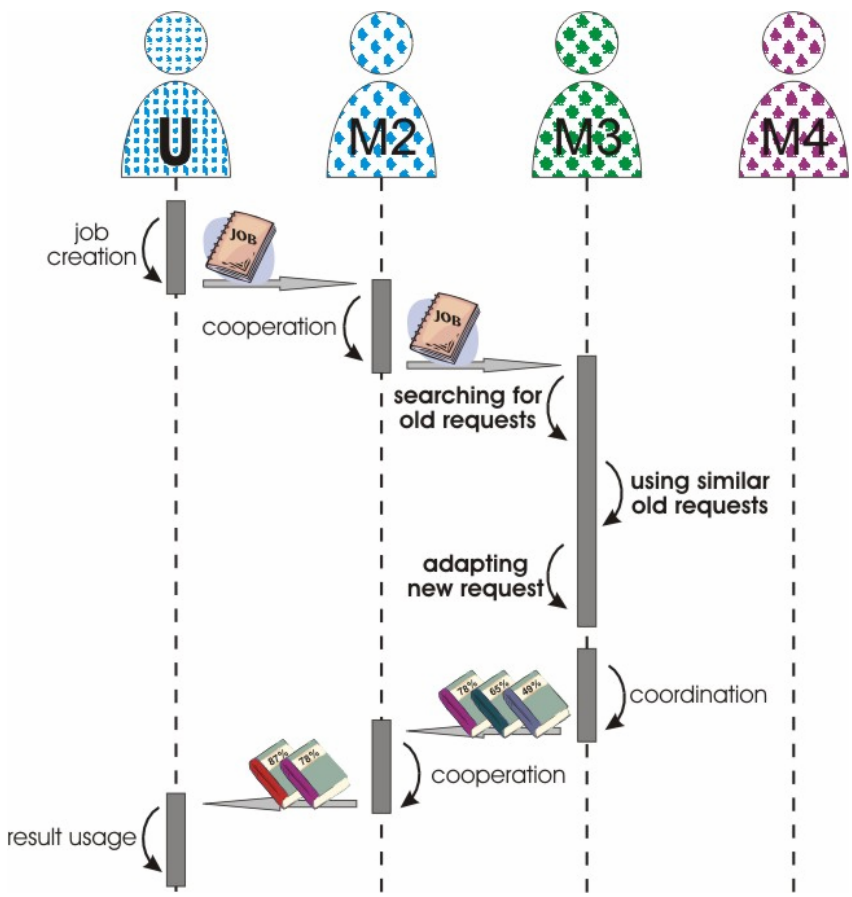

Figure 5.2: The exploration activities, being shortly named as searching for old requests, using similar old requests and adapting new request, take place between cooperation and coordination

5.3.1. Searching for Usable Old Requests

The application of the appropriate distance function is a critical point in performing an effective searching for the usable old requests, being the ones that have been processed

successfully, and at the same time that are similar enough with the actual one. Filtering requests are defined solely on the small subset of all available attributes, which makes request-attribute matrix very sparse. This sparseness is the main problem for Euclidean distance function that is defined for $F_r^{(a)}$ and $F_r^{(k)}$ as:

$$d_E(F_r^{(a)},F_r^{(k)}) = \sqrt{\sum_{i\in\{attributes\ in\ both\ F_r^{(a)}\ and\ F_r^{(k)}\}} avg(\omega_i^{(a)},\omega_i^{(k)})\left(\frac{v_i^{(a)}-v_i^{(k)}}{\sigma_i}\right)^2} \tag{5.1}$$

where $avg(x,y)=(x+y)/2$, σ_i is a standard deviation for attribute $a_i(v_i,\omega_i)$, and $v_i^{(x)}$ and $\omega_i^{(x)}$, $x\in\{a,k\}$ are value and weight for attribute a_i, respectively. The used rule for deciding about the similarity of objects specifies that for more similar objects, smaller distances should be obtained. Euclidean distance function can be unfortunately applied only on the parts of requests, which have the same attributes, with an obvious problem that the larger number of common attributes imposes probably a greater distance, being illogical because of an assumption that similar requests tend to have really many common attributes, and that requests without common attributes cannot be neighbours.

A distance function that is good for sparse spaces and that addresses the overlapping size is Jaccard index $d_J(F_r^{(a)},F_r^{(k)})$, which is defined as:

$$d_J(F_r^{(a)},F_r^{(k)}) = \frac{\sum_{i\in\{attributes\ in\ both\ F_r^{(a)}\ and\ F_r^{(k)}\}} avg(\omega_i^{(a)},\omega_i^{(k)})}{\sum_{i\in\{attributes\ at\ least\ in\ F_r^{(a)}\ or\ F_r^{(k)}\}} avg(\omega_i^{(a)},\omega_i^{(k)})} \tag{5.2}$$

where in the case of a presence of an attribute only in one request, a weighting average is returning the weight that is given, i.e. $avg(x,missing)=avg(missing,x)=x$. The greater Jaccard index means that corresponding objects are more similar.

Example 5.1: For the following pairs of requests $F_r^{(3)}\big(a_1(3{,}0.6)\big)\leftrightarrow F_r^{(4)}\big(a_1(6{,}0.4)\big)$ and $F_r^{(5)}\big(a_2(2{,}0.4),a_3(3{,}0.5),a_4(2{,}0.8)\big)\leftrightarrow F_r^{(6)}\big(a_2(3{,}0.4),a_3(3{,}0.5),a_4(2{,}0.5),a_5(4{,}0.9)\big)$, found Jaccard indexes are $d_J(F_r^{(3)},F_r^{(4)})=1$ and $d_J(F_r^{(5)},F_r^{(6)})=0.63$, or in other words the first pair is more similar than the second one, which is maybe not so reasonable in the case where similar neighbourhoods should be formed. This additionally holds because from the first pair nothing new can be learnt, but from the second pair, having much bigger overlapping, it can be deduced that attribute $a_5(4{,}0.9)$ is also maybe interesting for $F_r^{(5)}$.

The Example 5.1 illustrates a problem, which is a consequence of the fact that Jaccard index is taking care only of the number of attributes, being present in the objects that are compared, and not of the total number of the attributes in the whole attribute space. The easiest way to handle the noticed problem is by defining the absolute overlapping size $o_S(F_r^{(a)},F_r^{(k)})$, between filtering requests $F_r^{(a)}$ and $F_r^{(k)}$, as a ration between the number of overlapping and the total number of attributes, i.e.

$$o_S(F_r^{(a)}, F_r^{(k)}) = \frac{\sum_{i \in \{attributes\ in\ both\ F_r^{(a)}\ and\ F_r^{(k)}\}} 1}{\sum_{i \in \{whole\ attribute\ space\}} 1} \tag{5.3}$$

Another noticeable problem with Jaccard index is that attribute values are completely ignored, being exactly where Euclidean function is good. A natural solution is to combine introduced distance functions, and to take the best from both in the following way:

$$d_C(F_r^{(a)}, F_r^{(k)}) = \frac{\delta_E o_S(F_r^{(a)}, F_r^{(k)}) e^{-\beta d_E(F_r^{(a)}, F_r^{(k)})} + \delta_J d_J(F_r^{(a)}, F_r^{(k)})}{\delta_E + \delta_J} \tag{5.4}$$

where δ_E and δ_J controls the influence of $d_E(F_r^{(a)}, F_r^{(k)})$ and $d_J(F_r^{(a)}, F_r^{(k)})$, in the way that a larger value imposes a greater corresponding influence. Because $d_E \in [0, \infty)$ and $d_J \in [0,1]$, in order to make fair cooperation, Euclidean function is contributing in overall distance as $o_S(F_r^{(a)}, F_r^{(k)}) e^{-\beta d_E(F_r^{(a)}, F_r^{(k)})}$, where the factor $o_S(F_r^{(a)}, F_r^{(k)})$ ensures that short overlaps do not dominate.

Under the assumption that the actual filtering request $F_r^{(a)}$ is compared to any already processed request $F_r^{(i)}$ by computing the introduced combined distance $d_C(F_r^{(a)}, F_r^{(i)})$, finding k most similar filtering requests is the well known k- nearest neighbour search [127], where k filtering requests with the greatest $d_C(F_r^{(a)}, F_r^{(i)})$ will form the requested similar neighbourhood.

The loading, selecting and warning steps of the algorithm, which uncovers some of design decisions behind realizing k-nearest neighbour search, are given on Figure 5.3. While the cornerstone computation is performed in the selecting part of the algorithm, the loading and warning are important by reason of performance and administration purposes. These three parts are going to be briefly described and explained as follows.

The loading part is concerned with re-using already pre-computed standard deviation values for attributes from the actual request. As these values relates to a whole collection of requests, they do not change dramatically. Their on the fly computation might be also quite expensive for even the ordinary large databases. Performances are thus improved by only from time to time, let's say once a day or week, computing standard deviations, and by loading the last available their values always when they are needed.

The second selecting part does the heaviest task of computing first separately different distance measures between the actual request and any old available request. The single metrics that are used are Euclidean distance d_E, overlapping size o_S, and Jaccard index d_J, and they are afterwards combined to compute d_C. The similar neighbourhood S_k is created by checking whether the found d_C value, corresponding to the currently analysed old request, is at least as large as the smallest d_C for a request that is already selected in S_k. In the case of the addition of the given old request in S_k, it has to be always checked

whether the worst request from S_k should be removed. This can be simply achieved by designing neighbourhood S_k to be able to accommodate at most k requests, and always remove the request with the smallest d_C when being over sized.

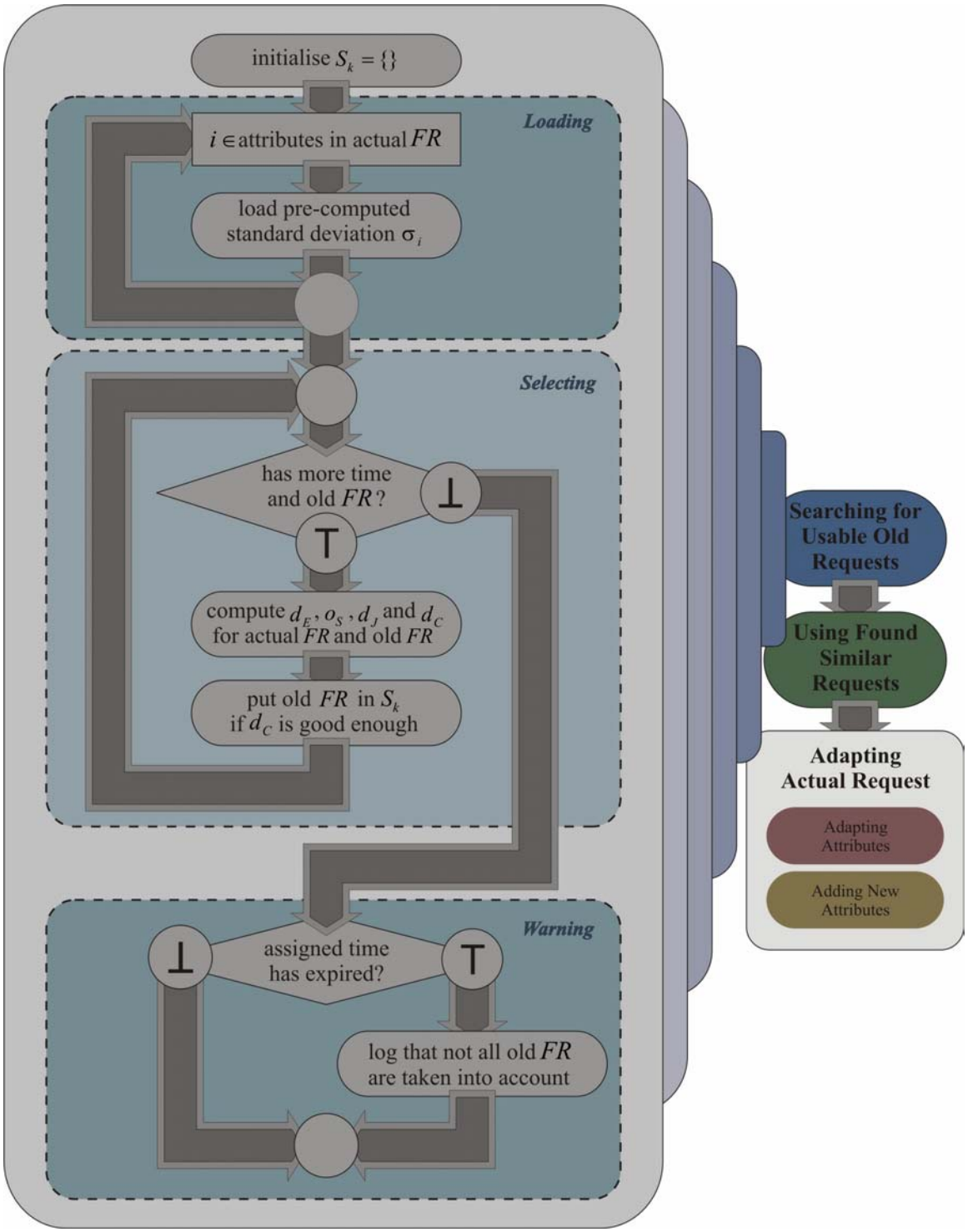

Figure 5.3: The creation of the k similar neighbourhood of old FR for the actual FR

The last warning part points out that in real situations not even the small fraction of available old requests can be taken into consideration. Similarly as modern collaborative filtering engines trade accuracy for speed by using only the user records that are currently in high speed memory, the exploration is intended to spend at most given amount of time on the creation to similar request neighbourhood. In the case where it happens that not all available old filtering requests could be processed, the warning message, specifying how many requests are taken into account because of escaping too long response time, might be generated.

Example 5.2: Example illustrates the formation of k-similar request neighbourhood in the attribute space $A_S\{a_1,...,a_8\}$.

Table 5.1: List of 10 past filtering requests, where attribute value and weight are marked as $^{\text{value}}/_{\text{weight}}$. Columns $d_E(a,i) \equiv d_E(F_r^{(a)}, F_r^{(i)})$, $d_J(a,i) \equiv d_J(F_r^{(a)}, F_r^{(i)})$, $o_S(a,i) \equiv o_S(F_r^{(a)}, F_r^{(i)})$ and $d_C(a,i) \equiv d_C(F_r^{(a)}, F_r^{(i)})$ correspond to the request $a \equiv F_r^{(a)}(a_1(3,0.5), a_7(4,0.2), a_8(3,0.4))$.

i	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	$d_E(a,i)$	$d_J(a,i)$	$o_S(a,i)$	$d_C(a,i)$
1		$^2/_{0.3}$				$^4/_{0.2}$	$^4/_{0.3}$		0	0.15	0.125	0.138
2			$^3/_{0.2}$	$^4/_{0.1}$					0	0	0	0
3	$^5/_{0.2}$	$^4/_{0.2}$			$^2/_{0.1}$			$^2/_{0.3}$	1.16	0.58	0.25	0.346
4	$^1/_{0.9}$		$^2/_{0.9}$		$^3/_{0.2}$		$^4/_{0.2}$	$^3/_{0.2}$	1.13	0.52	0.375	0.427
5	$^2/_{0.1}$			$^3/_{0.2}$	$^4/_{0.1}$	$^2/_{0.1}$	$^4/_{0.2}$		0.37	0.38	0.25	0.31
6				$^2/_{0.1}$	$^3/_{0.2}$		$^4/_{0.8}$	$^3/_{0.5}$	0	0.54	0.25	0.395
7		$^2/_{0.2}$	$^3/_{0.1}$			$^4/_{0.3}$	$^1/_{0.2}$		1.1	0.12	0.125	0.116
8	$^3/_{0.4}$		$^3/_{0.2}$	$^2/_{0.1}$				$^3/_{0.3}$	0	0.62	0.25	0.435
9		$^3/_{0.1}$		$^2/_{0.8}$	$^3/_{0.9}$		$^4/_{0.1}$	$^4/_{0.3}$	0.84	0.18	0.25	0.205
10	$^3/_{0.2}$	$^3/_{0.1}$		$^4/_{0.1}$		$^3/_{0.1}$			0	0.27	0.125	0.198

Let's assume that for actual filtering request $a \equiv F_r^{(a)}(a_1(3,0.5), a_7(4,0.2), a_8(3,0.4))$, a k-similar neighbourhood should be formed by using the history data about the ten past filtering requests. Mean and standard deviation for each attribute in the attribute space $A_S\{a_1,...,a_8\}$ are $m_1 = 2.8$, $m_2 = 2.8$, $m_3 = 2.75$, $m_4 = 2.84$, $m_5 = 3$, $m_6 = 3.25$, $m_7 = 3.5$, $m_8 = 3$, $\sigma_1 = 1.48$, $\sigma_2 = 0.83$, $\sigma_3 = 0.5$, $\sigma_4 = 0.98$, $\sigma_5 = 0.7$, $\sigma_6 = 0.95$, $\sigma_7 = 1.22$ and $\sigma_8 = 0.7$. Columns $d_E(a,i)$, $d_J(a,i)$, $o_S(a,i)$ and $d_C(a,i)$ in Table 5.1 correspond to Euclidean distance, Jaccard index, absolute overlapping size and combined distance between actual request a and each past filtering request i ($i = 1..10$). These columns clearly illustrate cases, where every separately used distance function does not performed well, and where a combined distance is a better choice.

The obvious drawback of Euclidean distance, being mentioned in a definition part of this section that a larger overlapping size probably imposes lower similarity, can be seen for $d_E(a, i = 4) = 1.13$, being the second highest value in column $d_E(a,i)$, which further

means that $F_r^{(4)}(a_1(1,0.9), a_3(2,0.9), a_5(3,0.2), a_7(4,0.2), a_8(3,0.2))$ is not really so similar to $F_r^{(a)}(a_1(3,0.5), a_7(4,0.2), a_8(3,0.4))$ and that almost all other requests are much better. Furthermore, requests $F_r^{(1)}(a_2(2,0.3), a_6(4,0.2), a_7(4,0.3))$, $F_r^{(2)}(a_3(3,0.2), a_4(4,0.1))$ and $F_r^{(10)}(a_1(3,0.2), a_2(3,0.1), a_4(4,0.1), a_6(3,0.1))$ are much more similar to request $F_r^{(a)}$ even though they have at most one common attribute.

Even the simple absolute overlapping size o_S, which both ignores attribute values and weights, is correctly concluding that $F_r^{(a)}$ is much more similar to $F_r^{(4)}$ than it is the case with $F_r^{(1)}$, $F_r^{(2)}$ and $F_r^{(10)}$, holding because corresponding absolute overlapping sizes are $o_S(a, i=1) = 0.125$, $o_S(a, i=2) = 0$, $o_S(a, i=4) = 0.375$ and $o_S(a, i=10) = 0.125$. By ignoring both attribute values and weights, o_S is not able to make a distinction between $F_r^{(a)} \leftrightarrow F_r^{(3)}$ and $F_r^{(a)} \leftrightarrow F_r^{(9)}$, even though $F_r^{(9)}$ is much better than $F_r^{(3)}$, as it will be illustrated in the following paragraph.

Jaccard index is performing better, but it also has problems because of not taking care of the overlapping quality. For the request $F_r^{(3)}(a_1(5,0.2), a_2(4,0.2), a_5(2,0.1), a_8(2,0.3))$ Jaccard index is $d_J(a, i=3) = 0.58$, being the second highest value in column $d_J(a,i)$. According to that $F_r^{(3)}$ is very similar to $F_r^{(a)}$, even though the overlapping $F_r^{(a)} \leftrightarrow F_r^{(3)}$, i.e. $(a_1(3,0.5), a_8(3,0.4)) \leftrightarrow (a_1(5,0.2), a_8(2,0.3))$ is quite bad, when attribute values are compared. The overlapping $(a_7(4,0.2), a_8(3,0.4)) \leftrightarrow (a_7(4,0.1), a_8(4,0.3))$ between $F_r^{(a)}$ and $F_r^{(9)}(a_2(3,0.1), a_4(2,0.8), a_5(3,0.9), a_7(4,0.1), a_8(4,0.3))$ is obviously much better, but Jaccard index is only $d_J(a, i=9) = 0.18$.

In Table 5.1 found $k=3$ and $k=5$ most similar past filtering requests, being requests ($4,6,8$) and ($3,4,5,6,8$), are marked by a different cell filling. These neighbourhoods are generated by applying the combined distance $d_C(a,i)$ with $\beta = 0.1$ and $\delta_E = \delta_J = 0.5$. The three most similar filtering requests are those that either have the maximal possible overlapping ($F_r^{(4)}$), or the very good little smaller overlapping ($F_r^{(6)}$ and $F_r^{(8)}$), being the logical choice.

5.3.2. Using Similar Requests

Making the concise description of the formed similar neighbourhood of requests is an ultimate task for the subsequent mining process, which will be accomplished through the computation of expected attribute value – importance $(v_i^{(e)}, \omega_i^{(e)})$ pairs. That computation is performed for each attribute, which is appearing at least once in a neighbourhood, by using the following formula:

$$a_k^{(e)} = \frac{\sum_{i \in neighbourhood} d_C(F_r^{(a)}, F_r^{(i)}) a_k^{(i)}}{\sum_{i \in neighbourhood} d_C(F_r^{(a)}, F_r^{(i)})} \tag{5.5}$$

In the expression (5.5), the summation is performed over the filtering requests from the similar neighbourhood that have attribute $a_k(v_k, \omega_k)$, and the algebraic operations are made on the vectors on a component basis with a result that both attribute value v_k and importance ω_k are computed. As summation weights a combined distance $d_C(F_r^{(a)}, F_r^{(i)})$ is used in order to ensure that more similar requests, having a larger $d_C(F_r^{(a)}, F_r^{(i)})$ value, have a greater influence on the computation of expected attribute value – importance pairs $(v_i^{(e)}, \omega_i^{(e)})$.

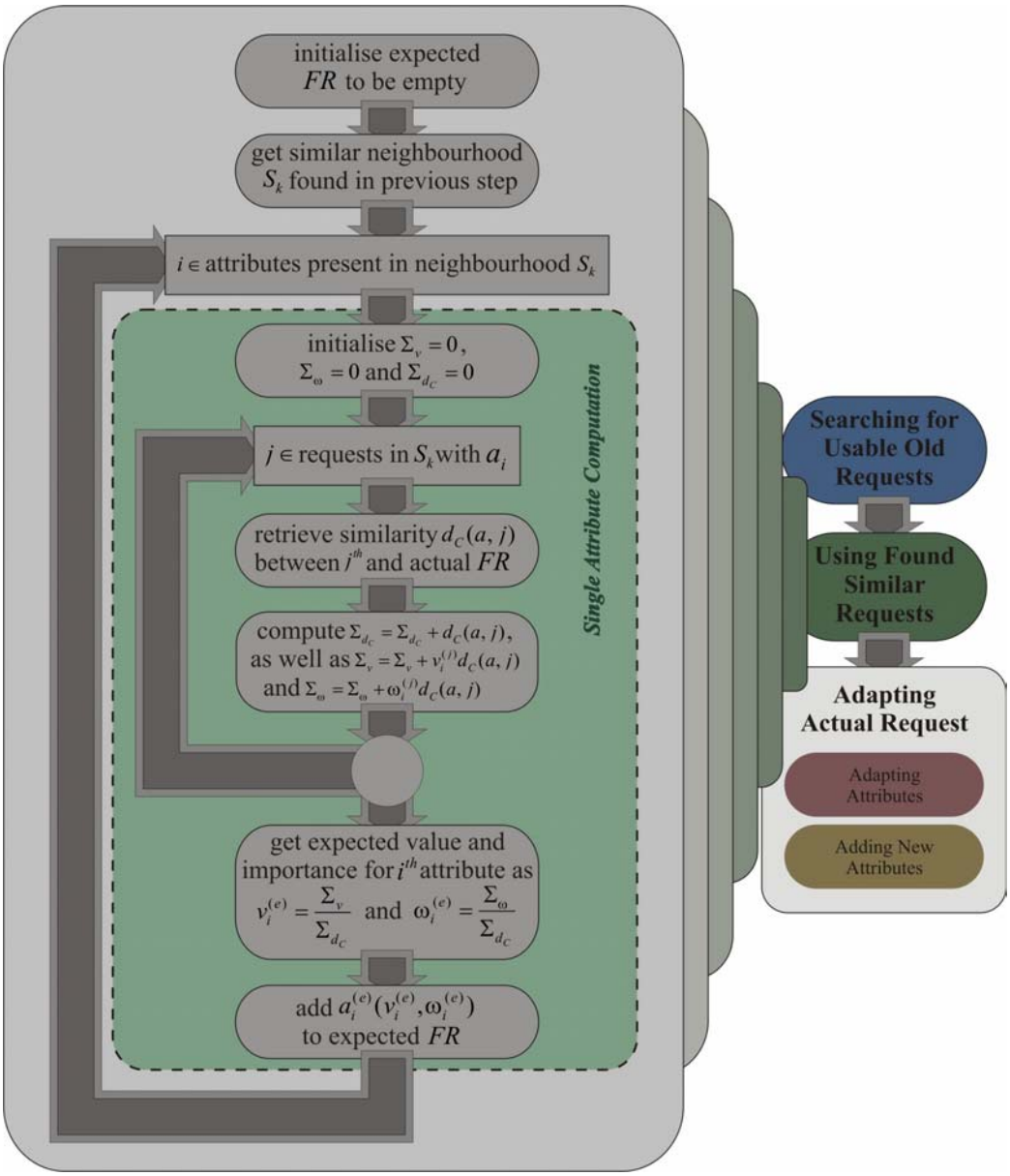

Figure 5.4: Using the created similar request neighbourhood S_k to compute expected FR

The computation of both attribute value and importance, being actually behind shortly stated that operations are made on component basis in (5.5), is algorithmically presented

on Figure 5.4. By passing through the atomic sub-steps that are inside the single attribute computation part, it becomes obvious that exactly the same operations are performed for value and importance of every attribute from the formed neighbourhood. While the first main loop iterates over all attributes a_i that are present at least in one old request from the similar neighbourhood S_k, the second inner loop, inside single attribute computation part, runs for all filtering requests in S_k that have the currently analysed attribute a_i. The algorithm for the second inner loop clearly shows that each request from S_k, having the currently analysed attribute a_i, contributes to the expected value $v_i^{(e)}$ and importance $\omega_i^{(e)}$ computation proportionally to its similarity $d_C(a,i)$ with the actual filtering request.

Example 5.3: By using k - similar request neighbourhoods, being formed in Example 5.2 for the actual filtering request $F_r^{(a)}(a_1(3,0.5), a_7(4,0.2), a_8(3,0.4))$, the expected attribute value – importance pairs $(v_i^{(e)}, \omega_i^{(e)})$ are computed and presented in Table 5.2.

Table 5.2: Expected attribute value – importance pairs corresponding to neighbourhoods in Table 5.1 for different neighbourhood size $k = 3,5,10$. Columns that correspond to attributes being present in $F_r^{(a)}(a_1(3,0.5), a_7(4,0.2), a_8(3,0.4))$ are marked with a different cell filling.

k	$a_1^{(e)}$	$a_2^{(e)}$	$a_3^{(e)}$	$a_4^{(e)}$	$a_5^{(e)}$	$a_6^{(e)}$	$a_7^{(e)}$	$a_8^{(e)}$
3	$^{2}/_{0.65}$		$^{2.5}/_{0.55}$	$^{2}/_{0.1}$	$^{3}/_{0.2}$		$^{4}/_{0.49}$	$^{3}/_{0.32}$
5	$^{2.68}/_{0.43}$	$^{4}/_{0.2}$	$^{2.5}/_{0.55}$	$^{2.27}/_{0.13}$	$^{2.97}/_{0.16}$	$^{2}/_{0.1}$	$^{4}/_{0.41}$	$^{2.78}/_{0.32}$
10	$^{2.7}/_{0.4}$	$^{3.09}/_{0.17}$	$^{2.56}/_{0.49}$	$^{2.45}/_{0.21}$	$^{2.98}/_{0.25}$	$^{2.92}/_{0.15}$	$^{3.78}/_{0.34}$	$^{2.92}/_{0.32}$

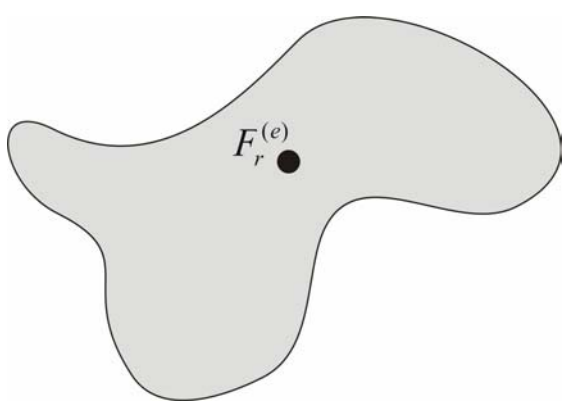

Figure 5.5: Formed similar request neighbourhood and its unique representation as expected $F_r^{(e)}$.

Because more similar filtering requests, being requests that overlap better with the actual one, have a greater influence when expected attribute value – importance pairs are computed, pairs $(v_1^{(e)}, \omega_1^{(e)})$, $(v_7^{(e)}, \omega_7^{(e)})$ and $(v_8^{(e)}, \omega_8^{(e)})$, which correspond to attributes a_1, a_7 and a_8 that control the overlapping quality, are quite close to the corresponding attribute pairs in $F_r^{(a)}$. Columns $a_1^{(e)}$, $a_7^{(e)}$ and $a_8^{(e)}$ in Table 5.2 additionally show that a

neighbourhood size does not have the large influence, especially when expected attribute values $v_1^{(e)}$, $v_7^{(e)}$ and $v_8^{(e)}$ are computed. That definitely approves the usage of the smaller neighbourhoods, being also preferred when computation resources are limited.

A collection of all computed expected attribute value – importance $(v_i^{(e)}, \omega_i^{(e)})$ pairs, marked on Figure 5.5 as $F_r^{(e)}$, can be clearly seen as the centroid of the similar request neighbourhood. Because of the applied weighting scheme, the neighbourhood centroid is formed under a guidance of the actual request, being an important difference between such a centroid and a centroid in any k - means partition clustering algorithm [127].

5.3.3. Adapting Actual Request

The utilised adaptation process should move the actual filtering request $F_r^{(a)}$ towards the computed centroid of a similar neighbourhood, marked on Figure 5.6 as the expected $F_r^{(e)}$. It is basically performed depending on presence or absence of a particular attribute in $F_r^{(a)}$, as concisely illustrated on Figure 5.7. While in the former case, attribute value – importance pair $(v_k^{(a)}, \omega_k^{(a)})$ is moved towards the expected $(v_k^{(e)}, \omega_k^{(e)})$, in the latter case, a decision to add the particular attribute a_k in $F_r^{(a)}$ is made when the missing attribute is found to be very important. Since the expected $F_r^{(e)}$ encapsulates the average knowledge of most successful old requests that are dealing with the similar matter as the actual one, the movement of the actual $F_r^{(a)}$ towards the expected $F_r^{(e)}$ by both adapting and adding attributes should hopefully result in an intelligent exploration of potentially relevant areas of interests. These two cases are going to be analytically and algorithmically described in the subsequent paragraphs.

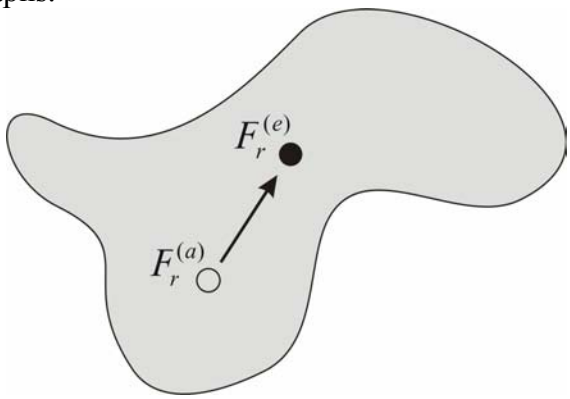

Figure 5.6: The adaptation of the actual request $F_r^{(a)}$ by moving it towards the centroid $F_r^{(e)}$.

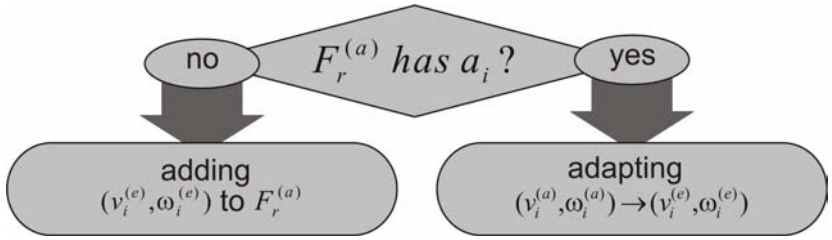

Figure 5.7: Adaptation of $F_r^{(a)}$ depends on the presence or absence of each attribute in $F_r^{(a)}$.

When actual filtering request $F_r^{(a)}$ has the particular attribute a_i, the attribute value – importance pair $(v_k^{(old)}, \omega_k^{(old)})$ is moved towards the expected one $(v_k^{(e)}, \omega_k^{(e)})$, being read from a centroid $F_r^{(e)}$, as:

$$a_k^{(new)} = a_k^{(old)} + \beta_E (a_k^{(e)} - a_k^{(old)}) \tag{5.6}$$

where $a_k^{(old)} \equiv (v_k^{(old)}, \omega_k^{(old)})$, $a_k^{(new)} \equiv (v_k^{(new)}, \omega_k^{(new)})$ and $a_k^{(e)} \equiv (v_k^{(e)}, \omega_k^{(e)})$ are old, new and expected attribute value – importance pairs, respectively. The algebraic operations in the expression (5.6) are done on vectors with the expected result that both attribute value and importance are adapted. The parameter $\beta_E \in [0,1]$ is an exploration rate, defining the level of the adaptation for both value and important of attributes in actual $F_r^{(a)}$. On the one hand, the higher values of β_E imply a greater adaptation, and consequently the larger exploration of relevant information areas. While being simply done by moving the actual $F_r^{(a)}$ towards expected $F_r^{(e)}$ in a greater extent, this can be a typical use-case for a novice user that critically needs the intelligent support in its information retrieval activities. On the other hand, the smaller β_E means that actual $F_r^{(a)}$ is not changed a lot, as well as that the exploration of new information space is less important than precisely delivering what has been asked. The advanced user behind obviously knows very well to create a request, and does not want any dramatic corrections.

Figure 5.8 illustrates the step-by-step procedure, which runs behind the adaptation of attributes that are already present in the actual $F_r^{(a)}$. While from the concise adaptation expression (5.6) is maybe not so clear what happens with every attribute, the algorithm on Figure 5.8 demonstrates that practically the same adaptation procedure is applied on both value and importance of each attribute from actual $F_r^{(a)}$. The applied transformation, making the cornerstone of the adaptation algorithm, is marked as adapting single attribute part on Figure 5.8. Although only attributes from $F_r^{(a)}$ are affected by the adaptation, the main loop iterates over every attribute from the expected $F_r^{(e)}$. This has been done only because of presentation reasons in order to make necessary the fundamental condition on the entrance to adapting single attribute part, which checks whether the currently selected attribute exist or not in the actual $F_r^{(a)}$, and which consequently distinguishes adaptation from addition. The much more efficient practical realisation naturally only iterates over the attributes from $F_r^{(a)}$, and reads the needed value and importance of attributes from the expected request $F_r^{(e)}$.

Example 5.4: By using expected $(v_i^{(e)}, \omega_i^{(e)})$ pairs for $k = 5$ from Table 5.2, attributes a_1, a_7 and a_8 from the request $F_r^{(a)}$ can be adapted as $a_1^{(a)}(3,0.5) \rightarrow a_1^{(e)}(2.68,0.43)$, $a_7^{(a)}(4,0.2) \rightarrow a_7^{(e)}(4,0.41)$ and $a_8^{(a)}(3,0.4) \rightarrow a_8^{(e)}(2.78,0.32)$. For $\beta_E = 0.5$, it follows $a_1^{(new)}(2.84,0.475)$, $a_7^{(new)}(4,0.305)$ and $a_8^{(new)}(2.89,0.36)$.

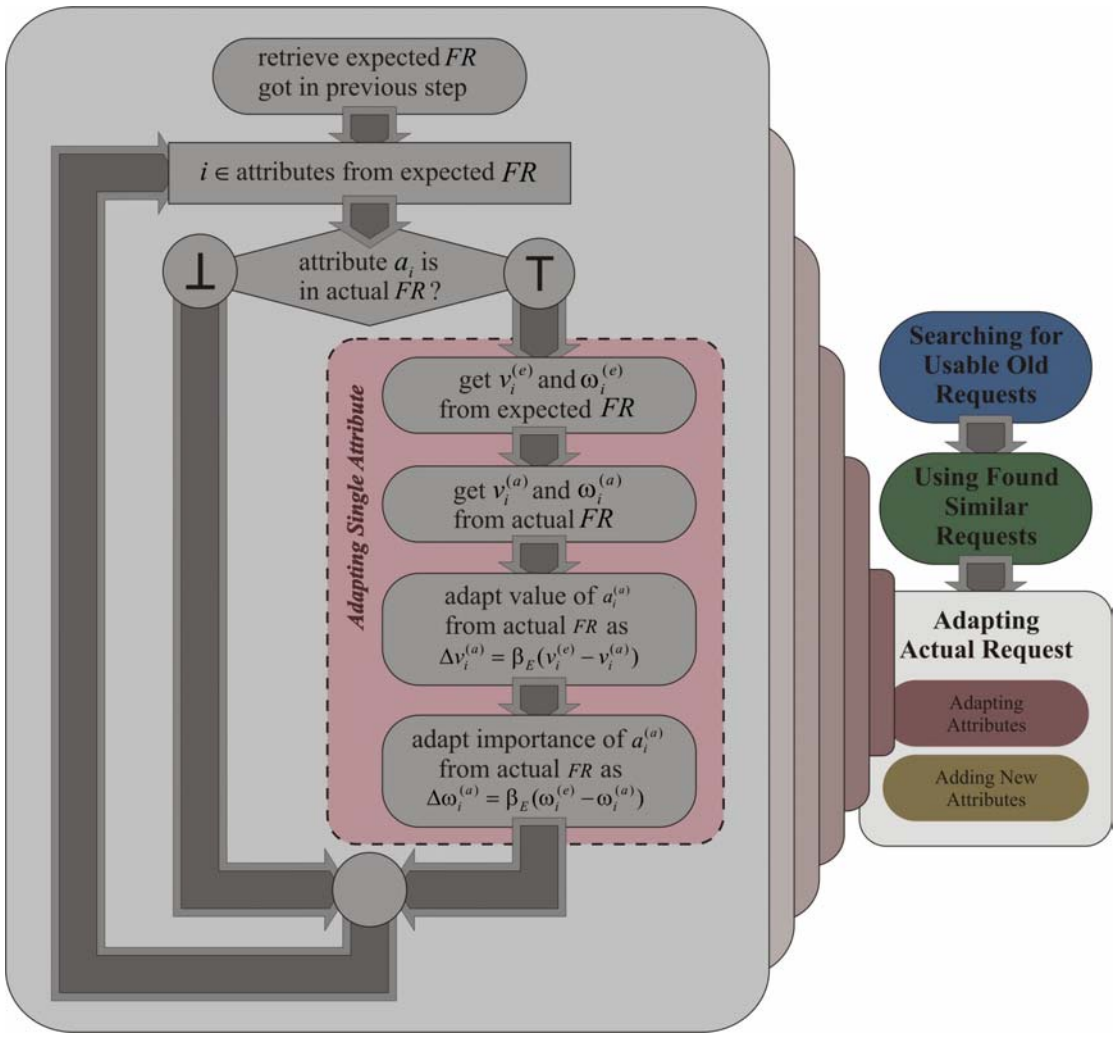

Figure 5.8: Adapting value and importance of attributes in actual FR by using expected FR

In the case where the given attribute a_k is missing in $F_r^{(a)}$, a decision to add a missing attribute depends whether the particular attribute plays the significant role in the formed neighbourhood, being formally checked as:

$$\frac{\sum_{i \in neighbourhood} \omega_k^{(i)}}{\sum_{i \in neighbourhood} \sum_{j \in all\ attributes} \omega_j^{(i)}} \geq 1 - \beta_P \tag{5.7}$$

In the expression (5.7), $\beta_P \in [0,1]$ is the prediction rate that defines how important a missing attribute a_k should be in similar request neighbourhood in order to be inserted in the actual request $F_r^{(a)}$. Larger prediction rate β_P results in a smaller $1-\beta_P$ value, and consequently more attributes will be added by reason of satisfying the insertion condition. While a numerator sums importance for a_k in similar neighbourhood, denominator sums importance for all attributes in that neighbourhood, being a way to get an impression of relative a_k importance, and to protect from the addition of too many attributes in $F_r^{(a)}$.

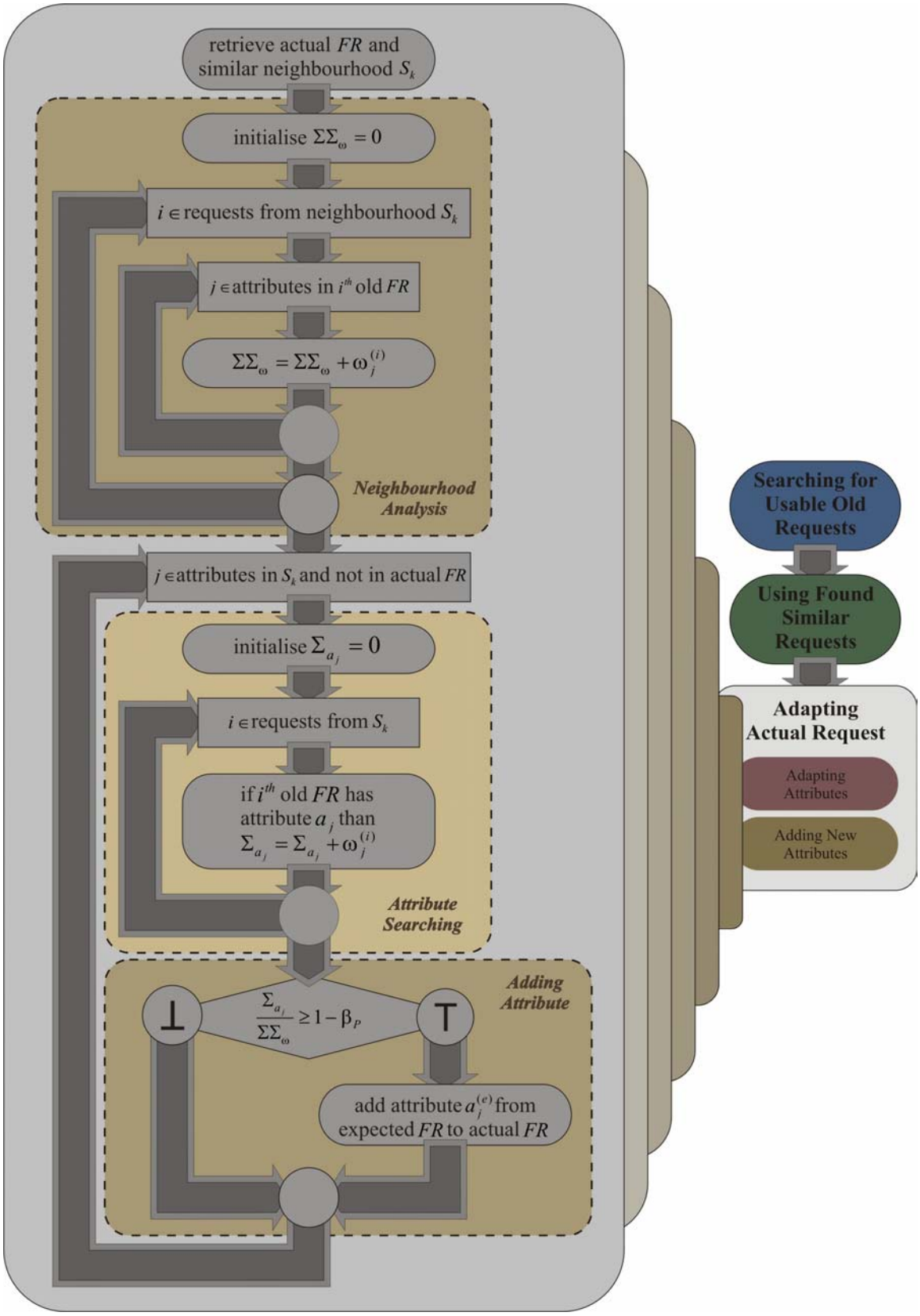

Figure 5.9: Adding missing attributes in actual FR and that are available in expected FR

The underlying computation of numerous needed sums for checking a condition (5.7) is algorithmically presented on Figure 5.9, where the activity of deciding, whether or not one attribute should be added in $F_r^{(a)}$, is separated into neighbourhood analysis, attribute searching, and adding attribute parts. The first step, being the neighbourhood analysis, is performed only once, with the goal to sum the importance values of all attributes over all requests, being nothing else than the denominator of the expression (5.7). The following steps are simply first summing important values separately for each and every attribute, and afterwards checking whether the single computed importance sum is large enough to permit the addition of the corresponding attribute.

Example 5.5: For the prediction rate $\beta_P = 0.85$, neighbourhood size $k = 5$, and data being given in Table 5.2, it follows that only attribute a_3 should be added in $F_r^{(a)}$ as $a_3^{(e)}(2.5,0.55)$. That holds as only $\sum_{i\in\{3,4,5,6,8\}} \sum_{j\in\{a_1,\ldots,a_8\}} \omega_j^{(i)} = 6.5$ and $\sum_{i\in\{3,4,5,6,8\}} \omega_3^{(i)} = 1.1$ satisfy the attribute addition condition.

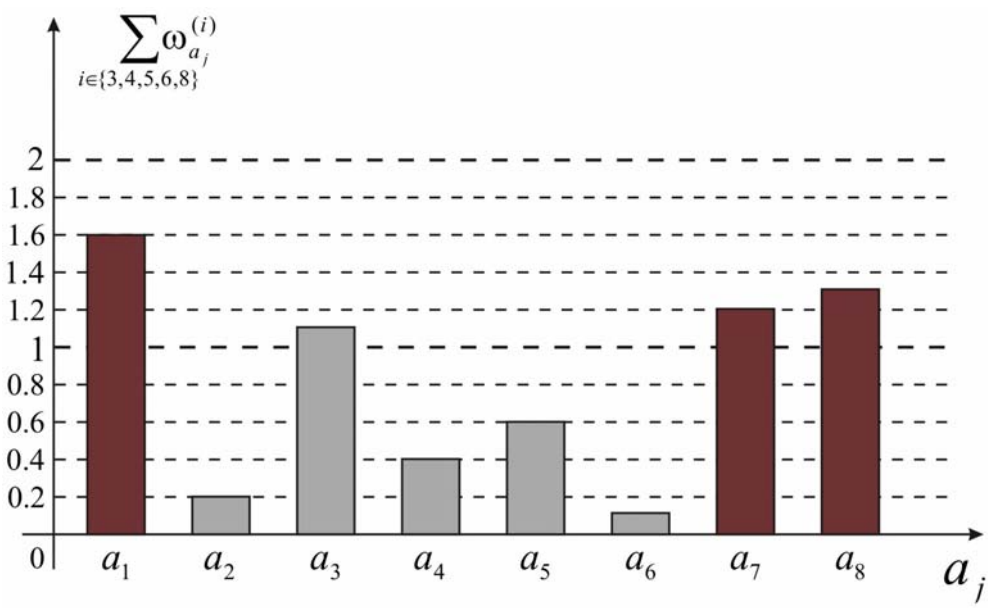

Figure 5.10: Absolute importance of every attribute in $k = 5$ neighbourhood from Example 5.2

The all separately computed sums of importance values for each of 8 attributes, being present in the neighbourhood of a size $k = 5$ from Example 5.2, are given on Figure 5.10. The three attributes, being a_1, a_7, and a_8, possess the largest resulting importance sums, mostly by reason of also being present in $F_r^{(a)}$, and thus being directly responsible for the creation of a neighbourhood. As being already presented on Figure 5.3, each old request is compared with the actual one, whose initially given attributes have the central role in the process of the selection of promising old requests. Even though a_1, a_7, and a_8 are evidently the most important attributes in the neighbourhood, they cannot be added by reason of being already in $F_r^{(a)}$, and therefore they are marked on Figure 5.10 by a red cell filling. As far as other attributes are concerned, with an exception of a_3, the resulting

importance sums are quite small, again because attributes that are not originally present in $F_r^{(a)}$ simply play no role while a similar neighbourhood is formed. The actual request can be made less dominant by utilising for example a so-called aggregate neighbourhood creation algorithm, which selects further old requests not only based on $F_r^{(a)}$, but also by consulting the already chosen requests.

Figure 5.11 gives a percentage distribution of importance sums from Figure 5.10, and therefore enables the graphical presentation of the attribute selection process. In the case where $\beta_P = 0.85$, and therefore $1-\beta_P = 0.15$, all not already included attributes, whose relative importance sums are greater than 15%, will be added into $F_r^{(a)}$. As a_1, a_7, and a_8 are already in $F_r^{(a)}$, only a_3 has its relative importance sum above 15%, and only a_3 will be added.

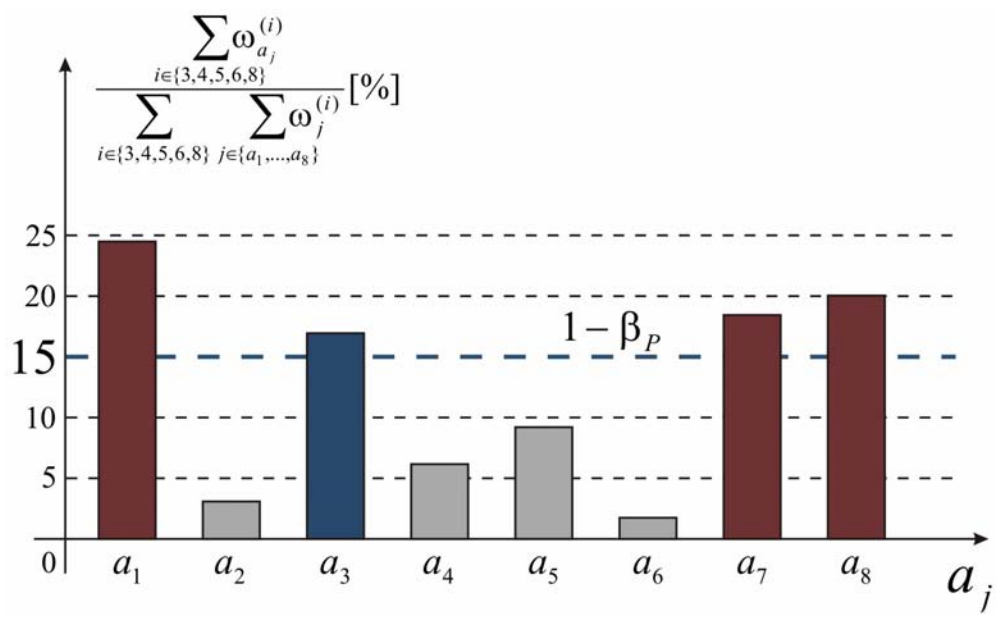

Figure 5.11: Relative importance of every attribute in $k = 5$ neighbourhood from Example 5.2, being suitable for the direct comparison with $1-\beta_P$ and deducing which attributes should be added

5.4. Conclusion

The multi-domain applicable exploration approach, being able to autonomously utilise the knowledge from the already processed requests to improve the actual one, has been presented in this Chapter. The underlying strengths of the presented exploration strategy are first concerned with the realisation of the combined distance function, which makes benefit from Euclidean distance and Jaccard index. The major advantage of the proposed approach is then contained in the controlled adaptation of the actual filtering request with the knowledge that has been mined from the found old requests.

The combination of Euclidean distance and Jaccard index has been analysed deeply in Section 5.3.1. The designed combined distance has aimed at exploiting the ability of the

Euclidean function to precisely compare the parts of requests with the same attributes. Its drawback of being poorly applicable in a highly sparse space has been complemented by Jaccard index, being known as an excellent choice when many attributes are missing in the requests that should be compared.

The controlled adaptation and addition of new attributes, being found in the formed similar neighbourhood, has been discussed in Section 5.3.3. The very simple and efficient adaptation mechanisms have been introduced in order to intelligently control in which extent the actual filtering request will be changed either by moving its attributes towards the expected values and weights or by adding the found important new attributes that are completely missing.

Chapter 6
Coordination Approach

The fundamental thoughts about efficiently coordinating available filtering strategies might be concerned with their demands towards different system resources, their abilities to produce relevant results for the given job, as well as their reliabilities to work without failures. At the first place, one should be very careful with the strategy, whose efficiency critically depends on a system resource, being currently highly loaded, since its selection will usually produce so-called long-lasting filtering jobs, having an unacceptable duration. Furthermore, the encapsulated filtering algorithm naturally provides a very different level of a support for various jobs, and treating the received requests as the black boxes during coordination might happen to be not always so promising idea. Lastly, in the time critical situations, it can be more important to select a strategy, which will as reliable as possible work without failures, than to think about providing the results with the highest quality. These three requirements, relating to optimally using available resources, carefully taking care of the properties of a received job, and wisely thinking about the self-healing aspects, are going, due to their great importance, to be separately addressed in Sections 6.3, 6.4 and 6.5, respectively.

The common property for any of the coordination schemes, which will be presented in this Chapter, is concerned with balancing the exploitation of what is already known about the available strategies with the exploration of their novel capabilities. This exploitation-exploration trade-off is fundamentally supported by using the proportional selection [181], which gives the chances even to the worst filtering strategy to be selected, and to provide the results that will be liked by a user. The background knowledge about the underlying capabilities of available strategies is always more or less problematic, and consequently the judgement, about which one is either the best or the worst candidate, might be weak. The probabilistic selection of strategies, where currently more fitted ones have the larger probability of being selected, seems to be thus a holy grail for both ensuring the satisfied quality of a provided filtering service, as well as learning about the hidden capabilities of novel searching techniques.

A natural way of incorporating a proportional selection into a core of any coordination scheme will be the main topic of this Chapter, being structured as follows. After basically introducing the main fitness values, which will be utilised for formally representing the different abilities of strategies, the most noticeable drawbacks of other solutions, trying to benefit from combining various filtering techniques, are going to be critically presented.

The main contribution of this Chapter, being already published in [11][13][14][15][17][18], is then contained in three separate sub-sections, which actually correspond to a step-by-step refinement of the basic coordination scheme. The first Section 6.3 gives the pure resource-aware coordination approach, which takes care only of the available resources, and which also becomes multi-domain applicable by reason of ignoring everything else. The second approach is named job-aware coordination algorithm as it additionally takes into account the properties of jobs, and it is presented in Section 6.4. The Section 6.5 finally gives the self-healing extension, which can upgrade either resource-aware or job-aware approach. The Chapter is finished with a section, where the concluding remarks are given.

6.1. Definitions and Preliminaries

The successful selection of a strategy, which should reliably enough process the actual filtering request in a current runtime situation, is essentially supported by the appropriate fitness values, illustrating the underlying abilities of searching algorithms. This section is going consequently after formally defining the coordination term, to introduce resource, job, and healing related fitness values.

Definition 6.1: *Coordination* is the comprehensive selection activity, being performed by a manager agent with an ultimate goal to find how reliably every filtering agent from its community can process the received request regarding a current availability of system resources. Three different coordination schemes, focussing on resource, job property, or reliability aspects, will make the foundation not only of this Chapter, but also of a whole Thesis.

6.1.1. Resource Related Fitness Values

Because CPU, DB and memory are, according to the author's assumptions, found to be the most important filtering resources, processor (F_{CPU}), database (F_{DB}) and memory (F_m) fitness values are introduced as follows:

Definition 6.2: *Central Processor Fitness* F_{CPU} describes the behaviour of a filtering strategy regarding a central processor load, where higher F_{CPU} means that less computing power is needed.

A high F_{CPU} value should be assigned to a strategy which is able to successfully work even in a case where a central processor is highly loaded. It is reasonable for example to assign a high F_{CPU} value to a strategy which either does not make the complex and CPU expensive computations or applies many useful approximations which save CPU power.

Definition 6.3: *Database Fitness* F_{DB} relates to filtering strategy behaviour regarding a database server load, where high F_{DB} is utilised for expressing the fact that a particular strategy loads a database server in a small extent.

A strategy, which for example stores in the operational memory a lot of data used for guiding a search and retrieving from a database only the really relevant data, will usually

have acceptable performances in the case of a high database server load and according to that should have a high F_{DB} value.

Definition 6.4: *Memory Fitness* F_m corresponds to the needed memory requirements of a filtering strategy, where higher F_m means that less memory is needed for performing filtering.

In the case where physical memory resources are not available in the needed quantity for a particular strategy, it is not only possible that a given strategy will have runtime difficulties, but also that maybe a whole filtering system will become un-operational. A logical guideline in such memory lacking situations is giving a priority to strategies with higher F_m values, because they are much less depended on the amount of available free memory.

6.1.2. Job Related Fitness Values

Jobs in a document retrieval domain can be seen as collections of words and phrases, and phrase (F_{ph}), big job (F_{bj}) and small job (F_{sj}) fitness values are defined as:

Definition 6.5: *Phrase Fitness* F_{ph} represents the behaviour of a strategy regarding a presence of phrases in the job, where a higher F_{ph} means the better handling of jobs with phrases.

A motivation to introduce F_{ph} is mostly concerned with the architecture of an inverted list based document retrieval engine [58][59], which has for every term to additionally store positions in order to smoothly support phrase searching, and consequently to have a large F_{ph}. Without position lists, phrase searching will be much more expensive, and that can be expressed through a smaller F_{ph}.

Definition 6.6: *Big Job Fitness* F_{bj} describes the ability of the filtering strategy to successfully handle jobs with many words, where higher F_{bj} value means a better support for big jobs.

Definition 6.7: *Small Job Fitness* F_{sj} corresponds to a strategy behaviour regarding jobs with only few words, where a higher F_{sj} value illustrates the better handling of small jobs.

Both F_{bj} and F_{sj} are defined mostly because of the existence of filtering strategies that use some mean for a dimensionality reduction, such as random mapping [127], Principal Component Analysis (PCA)[127], Self Organising Map (SOM) [148], and so on. Such strategies can successfully handle even jobs with thousands of words and consequently a high F_{bj} value should be assigned to them. Otherwise, strategies with no dimensionality reduction are usually bad concerning big filtering jobs, and therefore they should have a small F_{bj} value.

6.1.3. Healing Related Fitness Values

The properties, which are important for a successful recovery, are both related with the reliability of a strategy to work without failures and its ability to produce the necessary results. Consequently, failure (F_F) and efficiency (F_E) fitness values are defined as follows:

Definition 6.8: *Failure Fitness* F_F relates to the ability of a particular strategy to work without failures, where a high F_F means a good robustness and guarantees reliable and failure less filtering.

A strategy that has its own healing system will either never or very rarely fail, and therefore such a strategy should have a high F_F value. For example, there are very many filtering algorithms, such as hill climbing, simulated annealing, tabu search, gradient-based numerical optimisation methods and evolution algorithms [181], which might be stopped at anytime. In the case where something unexpected happened during filtering, these any time algorithms can easily return as final recommendations the results that are till then found. Obviously, it seems reasonable to assign high F_F values to such any time algorithms with self healing abilities.

Definition 6.9: *Efficiency Fitness* F_E represents the ability of a strategy to always find an asked number of results, where a higher F_E means better behaviour in unconditionally producing needed recommendations.

A motivation to introduce F_E is mostly concerned with the processing of many strictly formulated queries or jobs, which naturally impose that not enough results can be found even when very reliable strategy with high F_F is chosen. In such cases, strategies, having low F_E and high F_F, will not completely fail in filtering, but will most probably find only few results, which can really hardly satisfy the imposed information need. There are hopefully strategies that perform the automatic relaxation of query conditions [127] in the case where the original query has not produced enough results. It is therefore logical to assign high F_E values to such strategies to give them the advantage to be selected for the jobs whose straightforward treatment will hardly produce enough results.

6.2. Background

With the abundance of the available filtering strategies [82][94][127][181][196][213][221][224], finding the most appropriate one in a particular situation can amount to a real challenge. Each and every filtering strategy has its unique strengths and weaknesses and deciding how to efficiently combine them is something of an art [131] addressed by many researchers [76][77][131][262]. While the event-based and the content-based filtering are combined in Letizia [169][170] and Amalthaea [188] to better deduce the user model by tracking its behaviour during Web browsing, WebWatcher [27][253][254] and WAIR [60][278] additionally utilise reinforcement learning for adapting to constant changes in information needs. The same adaptation to dynamic users' interests is in PEA [269] and NewT [235] realised through the integration of evolution strategies, and in PIAgent [155]

by neural network classification with content-based filtering. In order to exploit aspects, such as style and point-of-view, which are invisible to pure content-based analysis, but are easily captured solely by humans [131], collaborative and content-based filtering are variously combined in FAB [30], P-Tango [76], TripMatcher [83], GroupLens [213] and many other systems.

To minimise the risk of choosing the inappropriate filtering strategy, the mentioned systems always combine only two well known strategies in a predefined static manner. A content based filtering is always one of the combined strategies, which is a reasonable conventional decision, because all these systems are concerned with information retrieval tasks. One nice trial to dynamically combine multiple strategies is reported in [77], where machine learning is used to produce a classifier that is capable to decide which strategy is the most promising to be applied for resolving a particular filtering job. The classifier is trained on a corpus of pre-selected documents with a relevance that is known for every pre-defined filtering job. All these jobs are resolved through the application of every available strategy, and in that way the classifier learns for which jobs each strategy is good for. The main limitation of this approach is contained in a requirement that the relevance of every document from a training corpus has to be manually determined for each pre-defined filtering job. It unfortunately becomes unfeasible to manually prepare enough {document, relevance} pairs for training the classifier that will be able afterwards to coordinate the broad spectrum of different jobs on collections with thousands of documents. This seriously limits the application of the coordination approach from [77] on only very small domains, having few hundreds documents, where the training process can be performed with a manageably small collection of the pre-selected documents and jobs.

The yet another attempt [262] is to apply reinforcement learning for finding the right filtering strategy in a given situation. The cornerstone idea is to reward the responsible strategy always after receiving a positive explicit user feedback value. This reinforcement learning coordination approach is unfortunately tested only with the artificial users that always both provide a feedback and do that in a deterministic way. It is far away from being truth that real users always provide the expected feedback values [37], which will certainly dramatically influence the successfulness of a reinforcement learning process. Coordination approaches, which will be presented in this Chapter, are fully aware of the instability of the explicit user feedback values, and consequently will additionally base its adaptation process on other much more stable indicators of interests, such as the response time [192].

According to the author's best knowledge, most of the mentioned systems are tested in highly protected environments, where usually the resource availability has not been taken into consideration. Even though the particular strategy can be the best suited one for the assigned job, in the case where the needed system resource is highly loaded, that strategy will most probably have weak chances to successfully produce expected results. Resource aware coordination mechanisms, which are going to be introduced in the Section 6.3, and which make a foundation for all other coordination algorithms, try to avoid the selection of a strategy for which the unfavourable resource situation exists, and to assign a job to a strategy that will produce results in the reasonable amount of time.

6.3. Resource-Aware Coordination Algorithm

The variety of filtering algorithms, having usually very different requirements towards the needed system resources, represents the cornerstone for a realisation of a coordination scheme, which tries to delegate the actual job to the strategy that can successfully work in the particular runtime situation. The necessary pre-conditions for such a resource-aware coordination approach are twofold. On the one hand, important system resources should be identified, and their current load should be as reliably measured as possible. The much more advanced estimation of resources might additionally request that some guaranties, concerning the needed amount of time for measuring the load values, should be given. On the other hand, the abilities of strategies to work in different runtime situations are nicely represented by the adequate resource related fitness values. Their successful usage surely requires the sophisticated learning procedure, which tries to adapt fitness values towards precisely representing a corresponding strategy. The efficient synergy between these two pre-conditions will be the main topic for the following discussions. The next sub-sections will as well show that the actual resource situation can be put together with the different abilities of strategies without even thinking about what is inside the received filtering job. This property finally makes the realised resource-aware coordination scheme to be multi-domain applicable, completely independent from both the types of jobs and used filtering strategies.

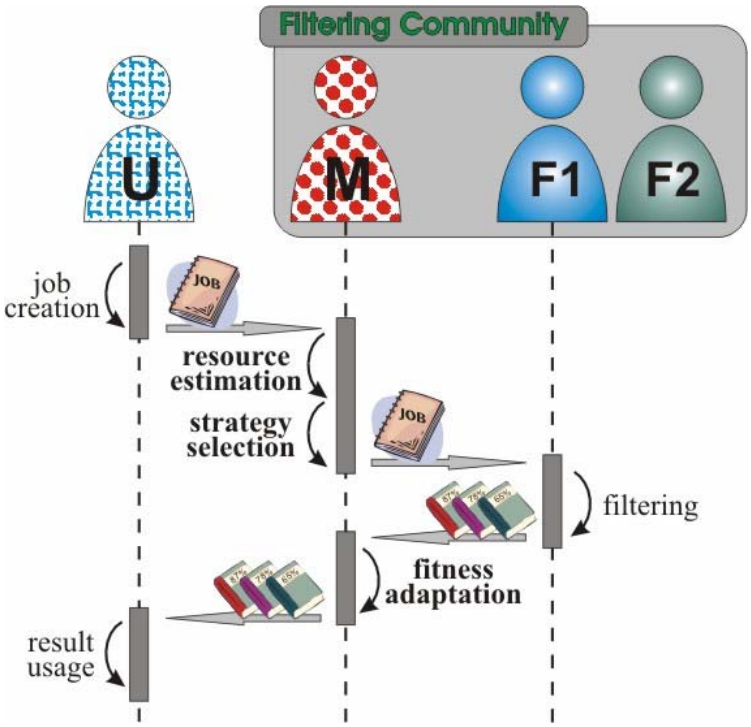

Figure 6.1: System architecture illustrating agent communication

The underlying coordination activities, being essentially responsible for the mentioned optimal exploitation of available system resources, are represented on Figure 6.1, which gives the simplest possible selection scenario. Under the assumption that everything goes perfectly, the scenario starts with a job creation and ends with a result usage, being done

by a User agent (U). The real coordination activities, being performed in a meantime by the chosen Manager (M), are first resource estimation, and afterwards strategy selection. After the selected Filtering agent (F), encapsulating a particular searching algorithm, has produced results, the manager can adapt fitness values based on the measurement of the response time. The found filtering results are finally returned back to the user agent, and the scenario ends.

The concrete solution, showing how these resource evaluation, strategy selection and fitness adaptation coordination steps can make benefit from F_{CPU}, F_{DB}, F_m fitness values, being formally introduced in Section 6.1.1, will be deeply described in the following sub-sections.

6.3.1. Resource Estimation

The ultimate evaluation goal is to estimate the applicability of each available filtering strategy for processing the current request, by first determining the actual system runtime properties and then combining them with, in the Section 6.1.1 introduced, strategy fitness values. System runtime properties, which are inside the interest scope of evaluation, are central processor (ω_{CPU}), database server (ω_{DB}) and memory system (ω_m) load. These runtime properties correspond to author's assumptions about the most important filtering resources for which the behaviour of the filtering strategy is already described by F_{CPU}, F_{DB} and F_m values. While the computation of the load of a memory system is based on determining the amount of the used memory for which integrated system functions[1] can be used, the load computation of both CPU and database server are more sophisticated, and require the execution, as well as the response time measuring of specially designed expressions.

A CPU load is simply computed as:

$$\omega_{CPU} = \beta_{CPU} t_{CPU} \tag{6.1}$$

where β_{CPU} is a tuning parameter and t_{CPU} is the time being needed for the computation of an algebraic expression, being formed only in order to estimate a current CPU load. A higher CPU load naturally implies the longer computation time t_{CPU} and consequently a larger ω_{CPU} value.

As an algebraic expression for CPU load estimation a sum $\sum_{i=1}^{n_{SL}} f(i)$ is used, where a function $f(i) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(i-m)^2/(2\sigma^2)}$ is the Gaussian variable with the standard deviation $\sigma = 1 + ran(0,1)$ and the mean $m = ran(0,1)$. Value $ran(0,1)$, being a randomly generated real number from $[0,1]$, is added on both standard deviation and mean in order to make every execution different from others, and to consequently ensure that no internal CPU optimisations can utilise something that has been already computed and to therefore make

[1] In java in Runtime class methods *freeMemory*, *maxMemory* and *totalMemory* are provided.

the unfair future executions. The summation length n_{SL} should be chosen in a way that the execution of the utilised expression does not load CPU significantly, and at the same time last long enough to provide an accurate CPU load computation. The experimentally found dependence of t_{CPU} from n_{SL} is given in the Evaluation Chapter in Section 8.4.1, where the load of CPU is controlled by lunching additional threads, which are repeatedly executing the $\sum_{i=1}^{n_{SL}} f(i)$ expression.

A whole process of first randomly generating mean m and standard deviation σ, and then measuring the execution time t_{CPU} of the requested summation is algorithmically presented on Figure 6.2. While the steps of taking the current system time, just before a summing starts $t_{CPU}^{(start)}$ and immediately after it is finished $t_{CPU}^{(stop)}$, are marked by red colour on Figure 6.2, a summation, whose execution time t_{CPU} is measured, is made distinct by being labelled as execution part.

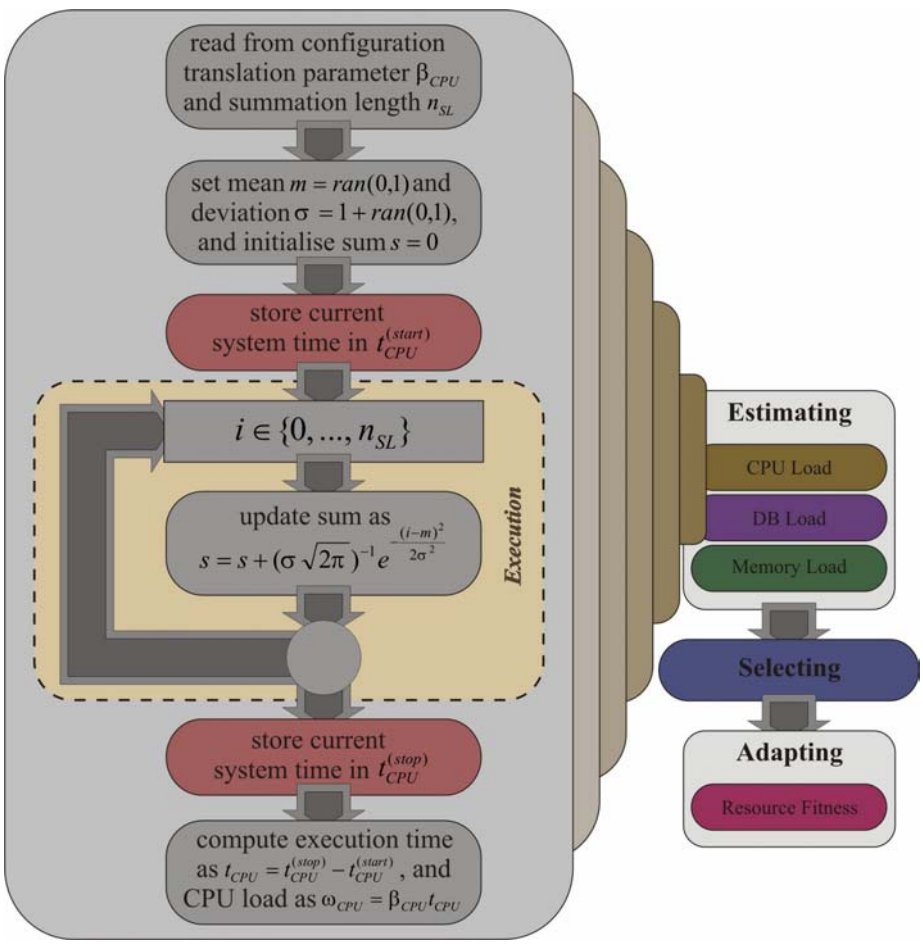

Figure 6.2: Estimating CPU Load

A database server load ω_{DB} is computed as:

$$\omega_{DB} = \beta_{DB} t_{DB} \tag{6.2}$$

where again β_{DB} is a suitable tuning parameter and t_{DB} is the time being needed for the execution of a SQL query that is specially designed to estimate a database server load. As a consequence of further loading database server, t_{DB} will increase, which finally results in a bigger ω_{DB} value.

The aggregate SQL query

SELECT *sum(Salary)* **FROM** *Employers*
WHERE *Id* > *ran* **AND** *Id* < n_{CS} + *ran* **GROUP BY** *Department*

should be executed on a very large database table $Employers(Id, Departement, Salary)$, which stores data about employers' salaries and departments where they work. In a given query ran is randomly generated number from $[0, rowcount]$, which ensures that always a summation is executed on a different cluster of employers, and consequently eliminates unwanted potential database server optimisations with already processed similar queries. A cluster size is defined by n_{CS} and should be small enough in order not to load database server significantly, and at the same time big enough to provide accurate enough database server load computation. A dependence of t_{DB} from n_{CS} will be experimentally analysed in Section 8.4.1, where a database server is additionally loaded by threads that repeatedly send different SQL queries.

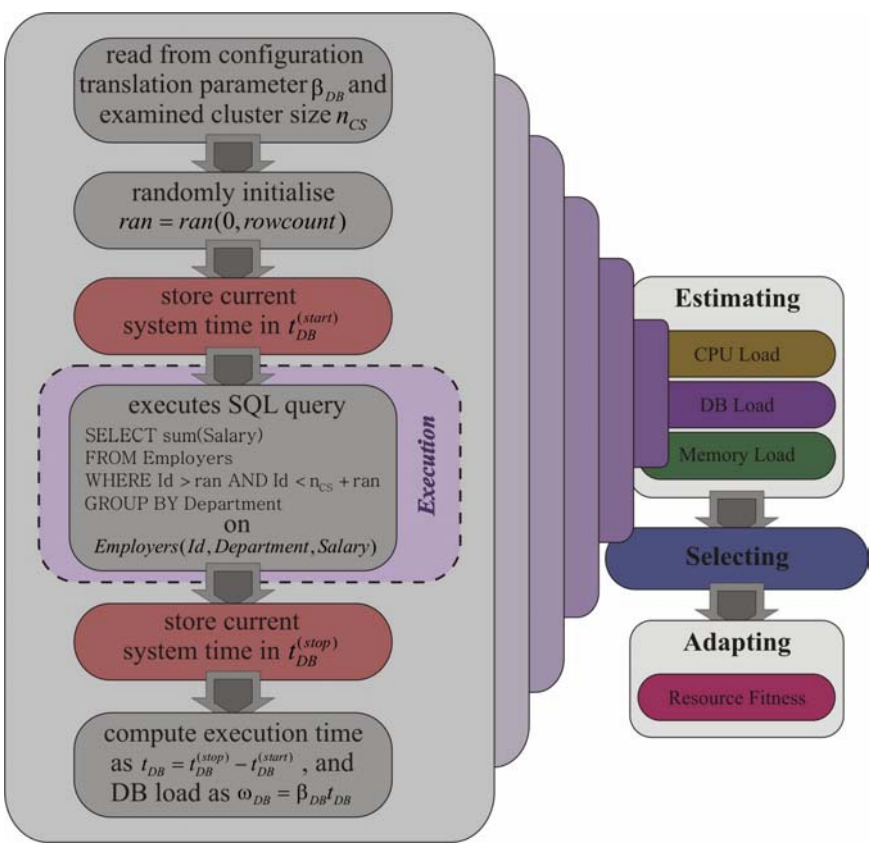

Figure 6.3: Estimating DB Load

Figure 6.3 briefly summarises a procedure of first taking the current system time $t_{DB}^{(start)}$ before querying a database server, then executing a SQL query with randomly generated starting cluster index, again retrieving the system time $t_{DB}^{(stop)}$ after a result set is obtained, and finally computing of a database server load ω_{DB}, being possible as all necessary data are available. A colour convention from Figure 6.2 of using red for marking steps where the system time is taken, as well as putting into the execution part a step, whose duration is measured, are still applied.

A memory load ω_m reflects the amount of used memory and it is computed as:

$$\omega_m = \beta_m s_m \tag{6.3}$$

where β_m is a tuning parameter and s_m is the size of the currently used memory in bytes. It is obvious from the expression (6.3) that more utilised memory s_m results in the larger memory load ω_m value. Because a system function for determining the amount of used memory usually exists, the problem of computing s_m is going to be assumed as a trivial one and will not be discussed in this Thesis.

After computing ω_{CPU}, ω_{DB} and ω_m load values a so-called resource fitness (F_r) can be computed as follows:

$$F_r = \frac{\omega_{CPU} F_{CPU} + \omega_{DB} F_{DB} + \omega_m F_m}{\omega_{CPU} + \omega_{DB} + \omega_m} \tag{6.4}$$

Each and every ω_x parameter, where $x \in \{CPU, DB, m\}$, plays a standard weighting role in a well known way that the larger particular ω_x implies the greater influence of corresponding $\omega_x F_x$ on overall F_r value. In the case where for example only a database server is highly loaded, ω_{DB} will have a large value, and $\omega_{DB} F_{DB}$ will dominate in a F_r computation. This further means that a filtering strategy with the highest F_{DB} value will probably have the highest overall F_r value.

6.3.2. Response Time Aware Resource Estimation

A general strategy, applied in the Section 6.3.1 for evaluating ω_x, $x \in \{CPU, DB, M\}$, values, is fundamentally based on an evaluation of a suitable expression, whose execution time is measured. A problem with such an evaluation strategy is that this execution time can be arbitrary large, which does not conform to the overall coordination scheme, which tries to reduce the response delay. It does not have any sense to deploy brilliant filtering algorithms, which can provide results in the assigned time slot, but to loose much more time on performing necessary coordination activities. Obviously, the evaluation of system resources has to become more time aware, as it is going to be shown in the following paragraphs.

A CPU load ω_{CPU} is thus computed as:

$$\omega_{CPU} = \begin{cases} 1 - e^{-\frac{\beta_{CPU} t_{CPU}}{t_{CPU}^{\max} - t_{CPU}}}, & t_{CPU} \le t_{CPU}^{\max} \\ 1 & , t_{CPU} > t_{CPU}^{\max} \end{cases} \tag{6.5}$$

The meaning of β_{CPU} and t_{CPU}, as well as the idea of measuring the execution time of a specially designed expression, are the same as in (6.1). The major novelty of expression (6.5) regarding (6.1) is that the execution time is now limited by $t_{CPU}^{\max}$, being the maximal amount of time that can be spent on ω_{CPU} estimation. In the case where a used algebraic expression cannot be completely executed within $t_{CPU}^{\max}$, the execution will be stopped, and the maximal load value of 1 will be assigned to ω_{CPU}. The higher CPU load naturally implies the longer computation time t_{CPU}, $e^{-\frac{\beta_{CPU} t_{CPU}}{t_{CPU}^{\max} - t_{CPU}}}$ decreases, and ω_{CPU} is closer to 1.

A realisation of the idea of spending at most $t_{CPU}^{\max}$ amount of time for estimating CPU load is algorithmically presented on Figure 6.4. The time awareness is simply achieved by conditionally entering into the execution step, which means that the summation will be actually performed only when the pre-defined maximal execution time $t_{CPU}^{\max}$ has not been expired. The role of the second step, being named as assigning load on Figure 6.4, is first to check whether the summation of n_{SL} elements has been already finished, or $t_{CPU}^{\max}$ has been spent on execution, and accordingly to compute the right ω_{CPU} load value by using the expression (6.5).

A database load ω_{DB} can be similarly computed as:

$$\omega_{DB} = \begin{cases} 1 - e^{-\frac{\beta_{DB} t_{DB}}{t_{DB}^{\max} - t_{DB}}}, & t_{DB} \le t_{DB}^{\max} \\ 1 & , t_{DB} > t_{DB}^{\max} \end{cases} \tag{6.6}$$

While the meaning of β_{DB} and t_{DB} is the same as in (6.2), $t_{DB}^{\max}$ is the maximal amount of time that can be spent on ω_{DB} evaluation. As a consequence of further loading DB, t_{DB} will increase, $e^{-\frac{\beta_{DB} t_{DB}}{t_{DB}^{\max} - t_{DB}}}$ decreases, and that results in a bigger ω_{DB} value.

Figure 6.5 gives one of possible solutions for determining the load of a database server at most within $t_{DB}^{\max}$ amount of time. A different approach for obtaining time awareness is necessary since there is unfortunately no way to control a database server as being done on Figure 6.4 with CPU. The proposed workaround is contained in lunching two threads, named trigger and query thread, into an execution part on the algorithm. While the trigger thread has simply a role to sleep $t_{DB}^{\max}$ and afterwards to send notification, the query thread is responsible for loading database server by sending a created SQL request. The second, assigning load, step on Figure 6.5 will check which thread has been returned as first, and perform the computation of ω_{DB} according to the expression (6.6).

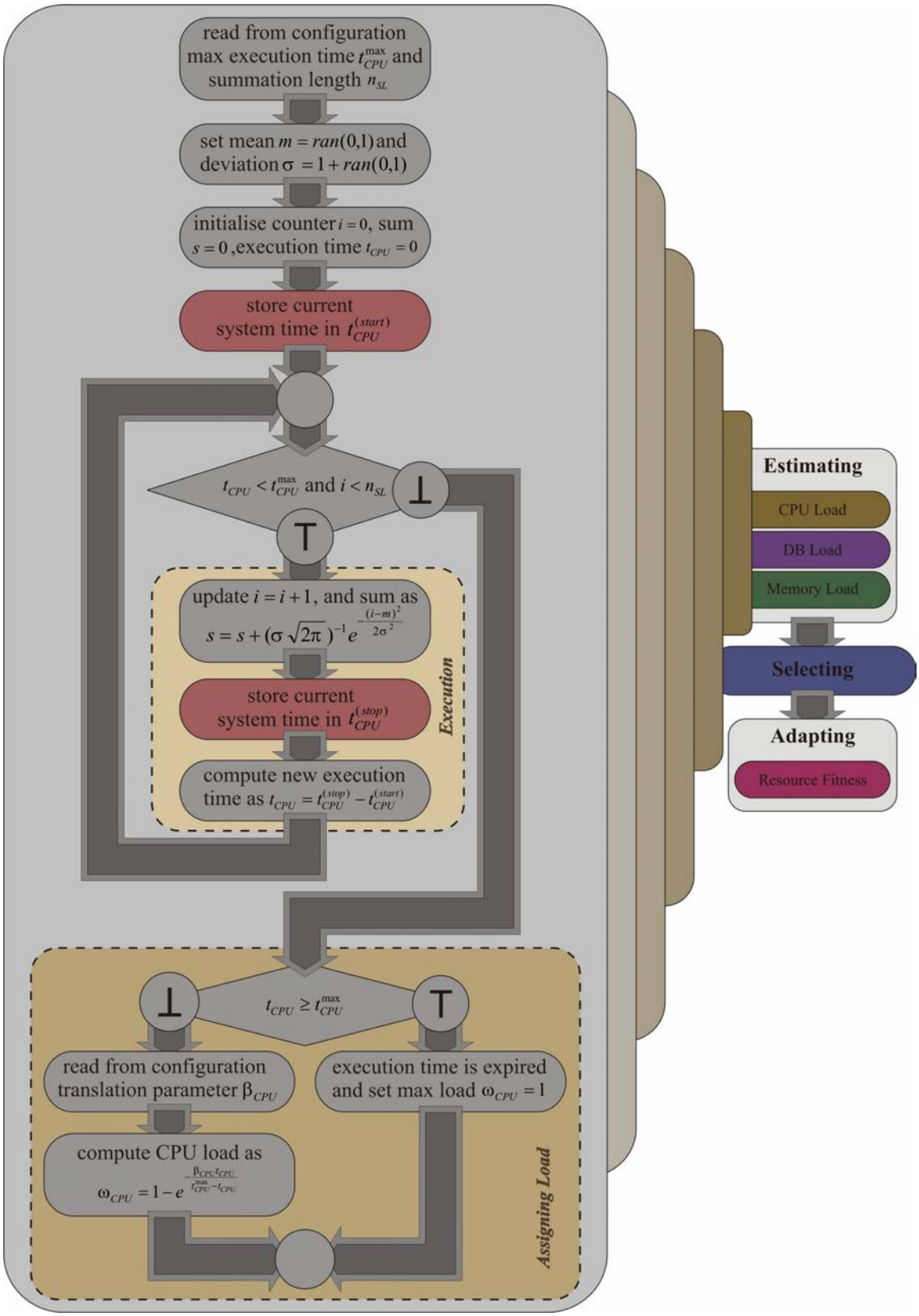

Figure 6.4: Response time aware computation of CPU load

Since the novel expressions (6.5) and (6.6) for computing ω_{CPU} and ω_{DB} additionally limit the possible load values to segment $[0,1]$, the similar transformation should be made when a memory load ω_m is computed. Instead of simply assigning $\omega_m = \beta_m s_m$ as in (6.3), a memory load is now computed as:

$$\omega_m = 1 - e^{-\beta_m s_m} \tag{6.7}$$

where parameters β_m and s_m have the same meaning as in (6.3). It holds that the more occupied memory s_m results in a smaller $e^{-\beta_m s_m}$, and consequently in a larger ω_m value.

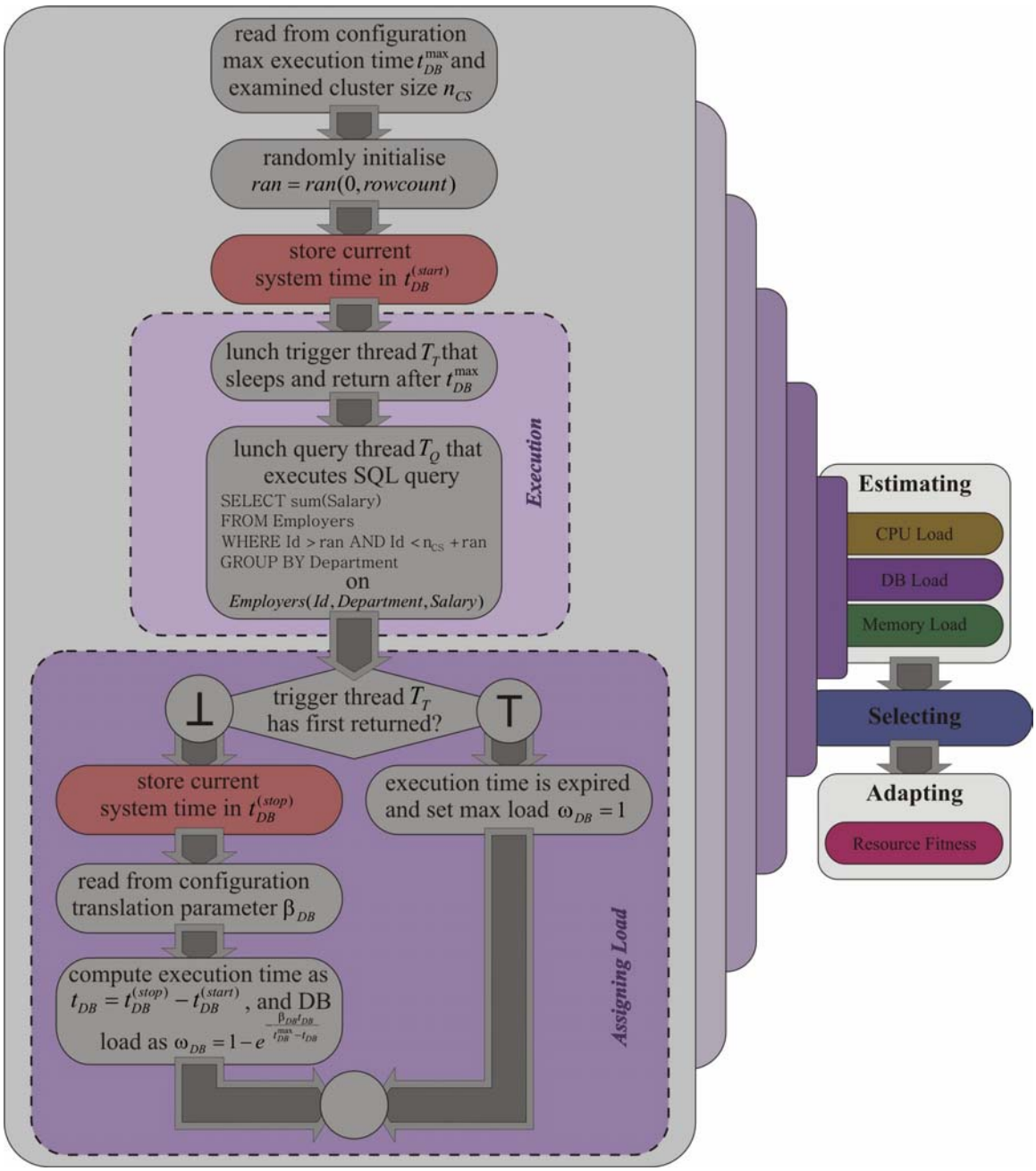

Figure 6.5: Response time aware estimation of database server load

6.3.3. Strategy Selection

Selection simulates an evolutionary process of a competition among available filtering strategies which are fighting for getting as many jobs as possible. In the case of the pure resource-aware coordination scheme, the only mean of fighting is contained in having the better estimation, regarding the ability to successfully work in a current runtime situation. These capabilities are consistently represented by a resource fitness F_r value, computed through the expression (6.4), and consequently a so-called total fitness F_t can be simply defined as:

$$F_t = F_r \tag{6.8}$$

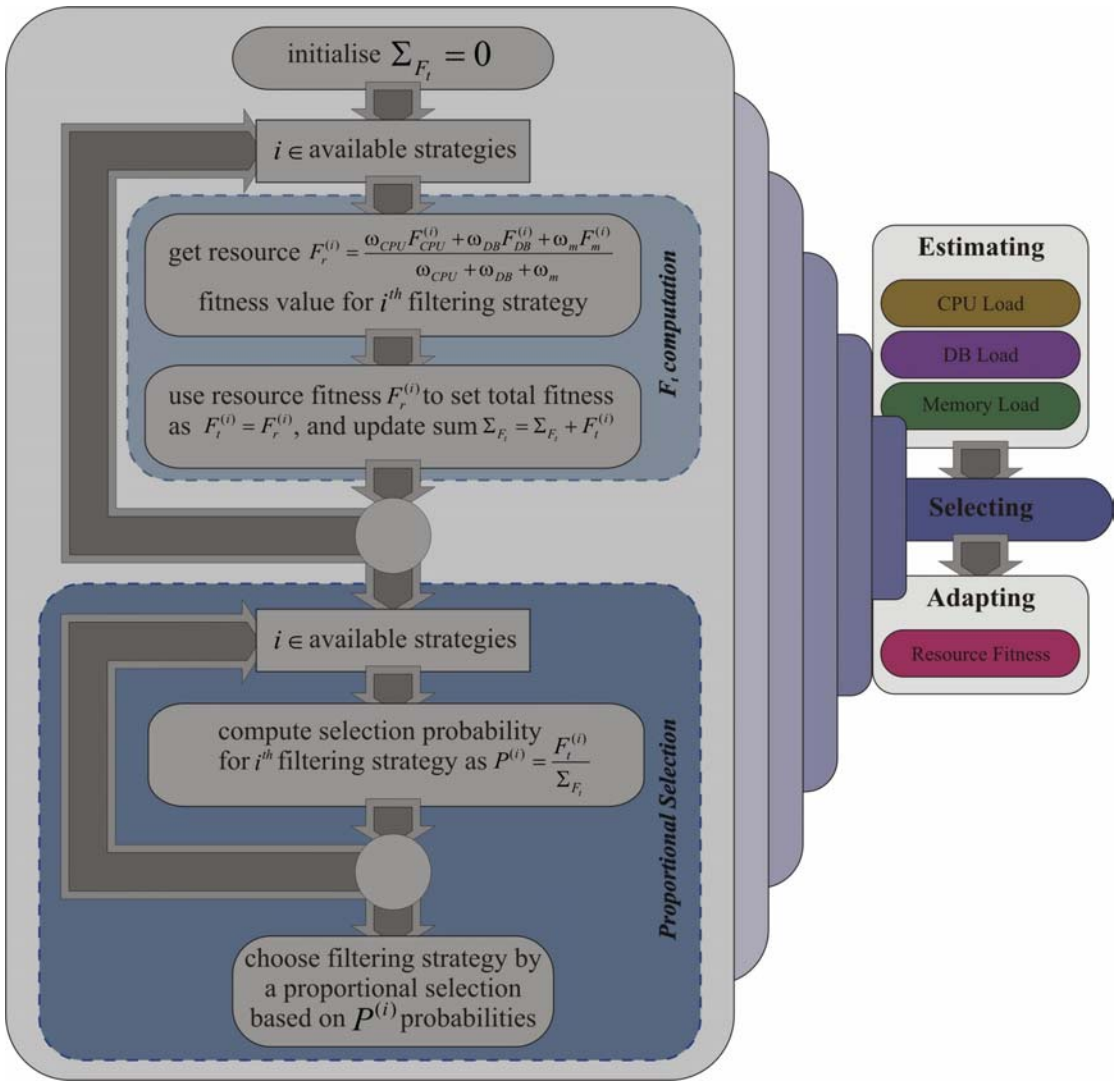

Figure 6.6: Strategy selection when only resource fitness is taken into account

The probability of being selected $P^{(i)}$, corresponding to the filtering strategy i, can be afterwards computed as:

$$P^{(i)} = \frac{F_t^{(i)}}{\sum_{j=1}^{n} F_t^{(j)}} \tag{6.9}$$

Such a defined probability $P^{(i)}$ is the growing function of $F_t^{(i)}$, and in extreme cases it holds $(F_t^{(i)} \to 0) \Rightarrow (P^{(i)} \to e^{-\infty} = 0)$, as well as $(F_t^{(i)} \to \infty) \Rightarrow (P^{(i)} \to e^{0} = 1)$. Under the assumption that n candidate filtering strategies participate in a selection, a denominator $\sum_{j=1}^{n} F_r^{(j)}$ sums all total fitness values, and therefore ensures $\sum_{j=1}^{n} P^{(j)} = 1$.

Because the probability of being selected $P^{(i)}$ is proportional to the filtering strategy $F_t^{(i)}$ fitness value, the used selection is nothing else than a well known proportional or roulette wheel selection [181]. Those filtering strategies with above average total fitness will, on average, receive more attention than those with below average total fitness. Even the one with the worst total fitness value will have a chance to be selected and to improve its fitness values, which is the cornerstone topic of the Section 6.3.4.

Due to a requirement that the sum of all selection probabilities has to be 1, the process of selecting a responsible filtering strategy is on Figure 6.6 separated into F_t computation and proportional selection sub-parts. While F_t computation is responsible for finding the denominator of expression (6.9), proportional selection ends the algorithm by choosing one strategy based on $P^{(i)}$ probabilities.

Example 6.1: Figure 6.7 represents one example of a selection, being based on the filtering strategy fitness values F_{CPU}, F_{DB} and F_m, and current system runtime situation, ω_{CPU}, ω_{DB} and ω_m.

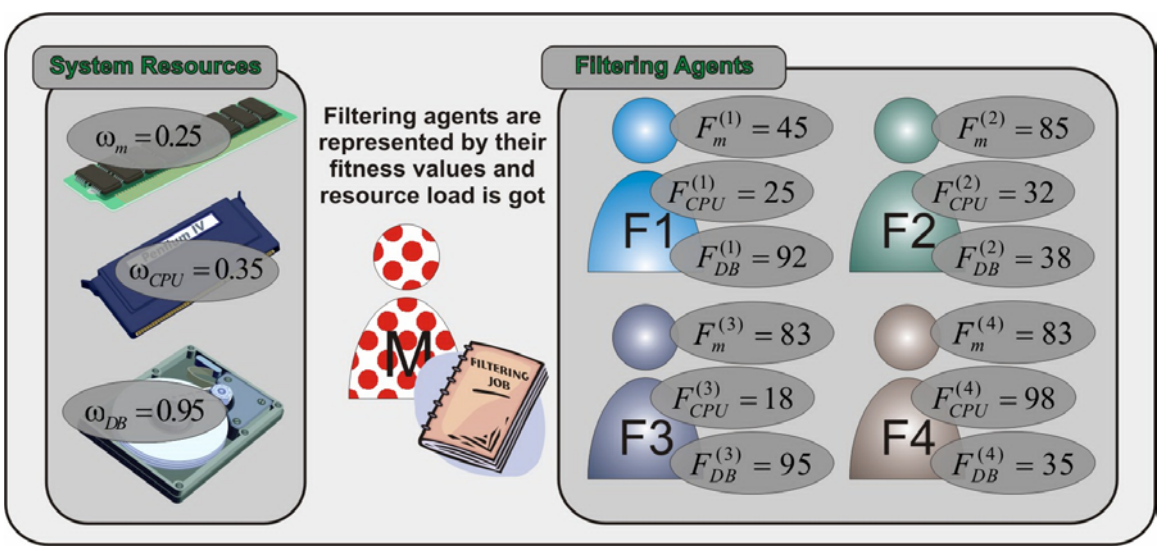

Figure 6.7: Coordination challenge seems to be resolvable after introducing strategy fitness values that formally represent different strategy capabilities towards exploiting available system resources

In a system runtime situation $(\omega_{CPU} = 0.35, \omega_{DB} = 0.95, \omega_m = 0.25)$ on Figure 6.7, total strategy fitness values are $F_t^{(1)} = 69.29$, $F_t^{(2)} = 44.22$, $F_t^{(3)} = 75.68$ and $F_t^{(4)} = 56.97$, and consequently for $\beta_t = 100$ the corresponding selection probabilities are $P^{(1)} = 0.31$, $P^{(2)} = 0.14$, $P^{(3)} = 0.34$ and $P^{(4)} = 0.22$. One should notice that the filtering strategies ($i = 1$ and $i = 3$), which have the greatest database server fitness ($F_{DB}^{(1)} = 92\%$ and $F_{DB}^{(3)} = 95\%$), have the best chances to be selected because of a very high database server load, being currently $\omega_{DB} = 0.95$.

6.3.4. Fitness Adaptation

After completing the assigned filtering job, learning by the adaptation of fitness values takes place in order to ensure that the actual fitness values reflect as accurate as possible the corresponding abilities of a responsible strategy. An adaptation of F_{CPU}, F_{DB} and F_m is solely based on the measurement of the current response time, and therefore is always applicable, as well as completely user independent.

After the filtering job has been finished, the response time t_r is available and it can be utilised for finding k nearest filtering jobs, which have been processed in similar runtime situations and had almost the same response time, being on Figure 6.8 marked as creating similar neighbourhood part. Because a system runtime is described by ω_{CPU}, ω_{DB} and ω_m, finding k most similar filtering jobs is a well known k- nearest neighbour search [127], being performed on ω_{CPU}, ω_{DB}, ω_m and t_r attributes. Under an assumption that an actual job $a = (\omega_{CPU}^{(a)}, \omega_{DB}^{(a)}, \omega_m^{(a)}, t_r^{(a)})$ is compared to any old filtering job $i = (\omega_{CPU}^{(i)}, \omega_{DB}^{(i)}, \omega_m^{(i)}, t_r^{(i)})$ by computing Euclidean distance $d_E(a,i)$, the k filtering jobs with the smallest $d_E(a,i)$ will form the similar request neighbourhood. The mentioned Euclidean distance $d_E(a,i)$, representing a cornerstone of creating similar neighbourhood on Figure 6.8, is defined as:

$$d_E(a,i) = \sqrt{\sum_{x \in \{CPU, DB, m\}} \left(\frac{\omega_x^{(a)} - \omega_x^{(i)}}{\sigma_{\omega_x}} \right)^2 + \left(\frac{t_r^{(a)} - t_r^{(i)}}{\sigma_{t_r}} \right)^2} \tag{6.10}$$

To help avoid a dependence on different attribute ranges in a distance computation, standard deviations σ_{ω_x} and σ_{t_r} are introduced in the last expression. Standard deviation σ_A for the attribute $A \in \{\omega_{CPU}, \omega_{DB}, \omega_m, t_r\}$ is computed as:

$$\sigma_A = \sqrt{\frac{\sum_{i=1}^{n} (A^{(i)} - m_A)^2}{n-1}} \tag{6.11}$$

It is assumed that attribute A has n values, $A^{(i)}$ represents i^{th} attribute value and attribute mean is $m_A = \frac{1}{n} \sum_{i=1}^{n} A^{(i)}$. In order to diminish the effect of the extreme attribute

values, zero mean or z-score normalisation [127] is applied for the standardisation of the attribute values, being especially useful when there are outliers that dominate the min-max normalisation. As the computation of the standard deviation values might be quite expensive, these values are typically pre-computed and only when necessary loaded. The really first task on Figure 6.8 is consequently concerned with a loading of these standard deviation values, being needed when the Euclidean distance is computed in (6.10).

The expected fitness value $F_x^{(e)}$, $x \in \{CPU, DB, m\}$, can be afterwards computed as:

$$F_x^{(e)} = \frac{\sum_{i \in S_k} sim(a,i) F_x^{(i)}}{\sum_{i \in S_k} sim(a,i)} \tag{6.12}$$

In the expression (6.12), the strategy fitness values, being responsible for the job i, are $(F_{CPU}^{(i)}, F_{DB}^{(i)}, F_m^{(i)})$, the formed neighbourhood of similar old jobs is S_k, and consequently a summation is performed on found k neighbouring jobs. As summation weights similarity $sim(a,i) = e^{-d_E(a,i)}$ is utilised to ensure that more similar jobs, having a smaller distance $d_E(a,i)$, have a greater influence on fitness value computation.

The found expected fitness values $F_x^{(e)}$ are then used in changing actual fitness values $F_x^{(a)}$ which describe filtering strategy capabilities, being responsible for the actual job. The used adaptation rule is defined as:

$$\Delta F_x = \gamma_x l(t)(F_x^{(e)} - F_x^{(a)}) \tag{6.13}$$

While γ_x is the tuning parameter, $l(t) = l_0 e^{-\gamma_t t}$ is the decreasing learning rate which ensures that the already learnt fitness values will not be easily destroyed. The filtering strategies with the solid history will, according to such learning rate, change their fitness values in smaller extent than the novel ones. This process of first computing the expected fitness values, and then moving the actual ones towards them, is on Figure 6.9 shortly marked as a fitness adaptation step.

Example 6.2: This example illustrates first expected fitness $F_{CPU}^{(e)}$, $F_{DB}^{(e)}$ and $F_m^{(e)}$ value computation and then the adaptation of actual fitness $F_{CPU}^{(a)}$, $F_{DB}^{(a)}$ and $F_m^{(a)}$ values.

Let's assume that an actual filtering job a was processed in a system runtime situation $(\omega_{CPU}^{(a)} = 0.17, \omega_{DB}^{(a)} = 0.5, \omega_m^{(a)} = 0.34, t_r^{(a)} = 208)$ by a filtering strategy with fitness values $(F_{CPU}^{(a)} = 43, F_{DB}^{(a)} = 52, F_m^{(a)} = 78)$ and that the needed time for processing was $t_r^{(a)} = 208$. Fitness values $F_{CPU}^{(a)}$, $F_{DB}^{(a)}$ and $F_m^{(a)}$ are adapted by using the history data about 10 past filtering jobs, being given in Table 6.1. For these jobs mean and standard deviation are $m_{\omega_{CPU}} = 0.474$, $m_{\omega_{DB}} = 0.541$, $m_{\omega_m} = 0.602$, $m_{t_r} = 275.4$, $\sigma_{\omega_{CPU}} = 0.2953$, $\sigma_{\omega_{DB}} = 0.17$, $\sigma_{\omega_m} = 0.2289$ and $\sigma_{t_r} = 114.58$. Columns $d_E(a,i)$ and $sim(a,i)$ in Table 6.1 correspond to Euclidean distance and similarity between actual job a and each past filtering job i $(i = 1..10)$.

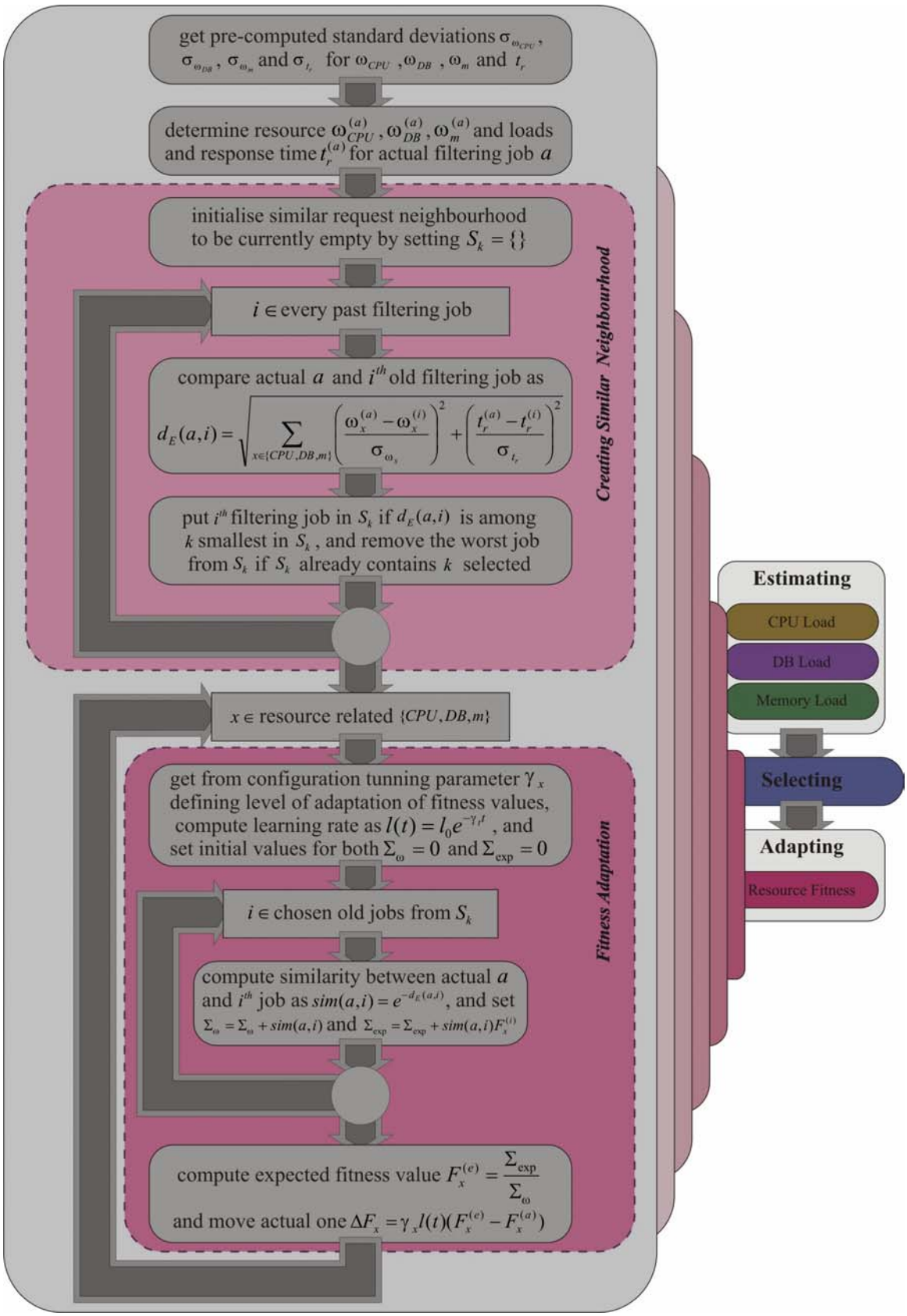

Figure 6.8: Resource fitness learning through neighbourhood creation and fitness adaptation

In Table 6.1 found $k = 3$ and $k = 5$ most similar past filtering jobs, being jobs $(1,2,7)$ and $(1,2,4,6,7)$, are marked by the different cell filling. As it can be seen from Table 6.2, increasing a neighbourhood size k does not change expected fitness values almost at all, because filtering jobs with low similarity with the actual one give very small contribution, being a desired effect of including $sim(a,i)$ as a weight in $F_x^{(e)}$ computation.

Table 6.1: List of 10 past filtering jobs, being processed within response time t_r by filtering strategy with fitness values (F_{CPU}, F_{DB}, F_m) in runtime situation $(\omega_{CPU}, \omega_{DB}, \omega_m)$. Columns $d_E(a,i)$ and $sim(a,i)$ correspond to $a = (\omega_{CPU}^{(a)} = 0.17, \omega_{DB}^{(a)} = 0.5, \omega_m^{(a)} = 0.34, t_r^{(a)} = 208)$.

i	ω_{CPU}	ω_{DB}	ω_m	t_r	F_{CPU}	F_{DB}	F_m	$d_E(a,i)$	$sim(a,i)$
1	0.17	0.51	0.33	232	23	57	73	0.22	80%
2	0.18	0.49	0.32	212	42	63	52	0.11	89%
3	0.65	0.78	0.56	329	15	25	32	2.71	6.5%
4	0.34	0.67	0.67	298	87	76	56	2	13.5%
5	0.23	0.72	0.78	124	34	87	92	2.43	8.7%
6	0.56	0.69	0.6	134	9	19	82	2.16	11.4%
7	0.16	0.53	0.35	198	32	72	67	0.2	81.5%
8	0.95	0.32	0.56	453	45	87	23	3.68	2.5%
9	0.67	0.41	0.87	345	87	67	43	3.15	4.2%
10	0.83	0.29	0.98	429	92	84	35	4.24	1.4%

After the expected strategy fitness values are determined and under assumptions that $k = 3$, $\gamma_x = 1$ and $l(t) = l(t = 0, l_0 = 1/2) = l_0 e^{-\gamma_t t} = 1/2$, a computation of ΔF_x then gives $\Delta F_{CPU} = -5.15$, $\Delta F_{DB} = 6.1$ and $\Delta F_m = -7.15$, and subsequently the new fitness values, which are going to be finally stored in a history table, are $F_{CPU}^{(n)} = 37.85$, $F_{DB}^{(n)} = 58.1$ and $F_m^{(n)} = 71.85$.

Table 6.2: Expected fitness values for different neighbourhood sizes k where both actual and past filtering jobs are given in Table 6.1 and in its caption.

k	$F_{CPU}^{(e)}$	$F_{DB}^{(e)}$	$F_m^{(e)}$
3	32.7	64.2	63.7
5	34.4	62.7	64
10	35	62.9	63.3

6.4. Job-Aware Coordination Algorithm

Since not each and every filtering job is the same, a much more advanced coordination scheme will take care not only of efficiently exploiting the available system resources as in the Section 6.3, but also of the properties of the received request. It is far away from being truth that every strategy can equally well support each type of the filtering job. In a domain of text documents, some strategies are for example better suited for queries with

many words, whereas the others are more efficient for the jobs with phrases. The major motivation for a job-aware coordination solution is therefore concerned with additionally taking into consideration the properties of the filtering jobs while selecting a responsible strategy.

The expected advantage of analysing the jobs, before deciding which strategy will do filtering, should be found finally in providing better results, and consequently increasing the user satisfaction. The unavoidable disadvantage is unfortunately contained in making the coordination scheme to be domain dependent. Filtering jobs are no more black boxes for the coordination engine, and their properties should be mined. Since a domain of text documents is probably the most commonly used, the necessary job analysis will assume that jobs are queries, having a different number of words and phrases. The corresponding capabilities of filtering strategies will be adequately concerned with processing jobs in a text domain, and the specialised adaptation procedures will be proposed.

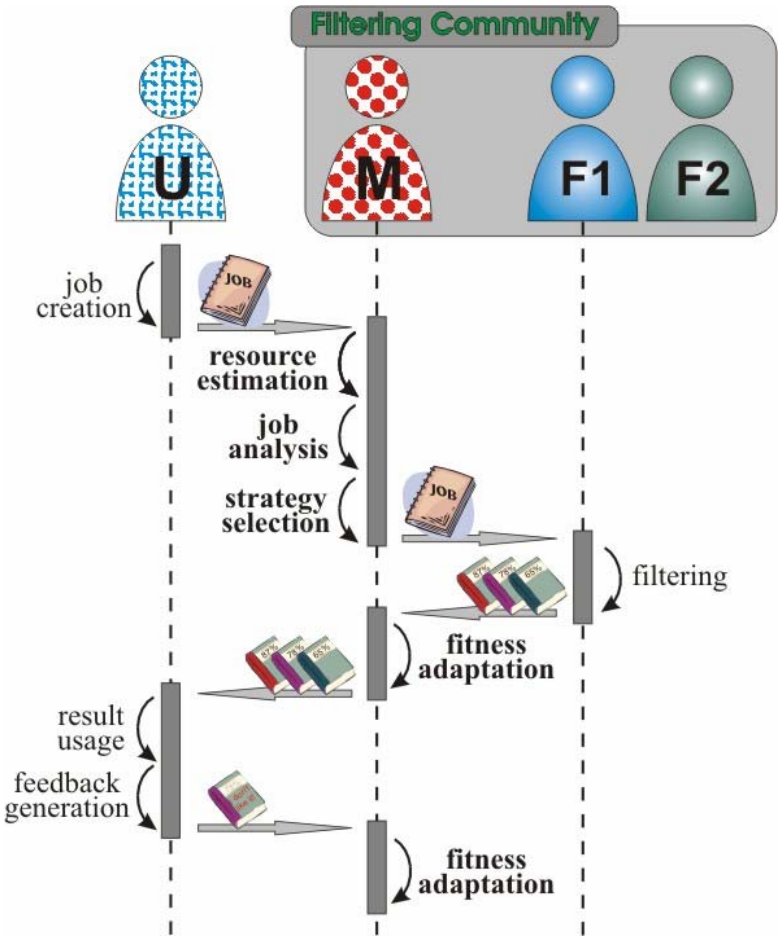

Figure 6.9: Coordination between manager and filtering agents through resource estimation, job analysis, strategy selection and fitness adaptation

Before showing in the remainder of this section how the job analysis can find its place in the overall process of the strategy selection, the extended system architecture will be given on Figure 6.9. *User agent* (U) is now responsible not only for a creation of filtering jobs by collecting user preferences, but also for generating feedback. *Filtering agent* (F)

again encapsulates one particular searching strategy that is used for resolving the received filtering job and providing the requested results. *Manager agent* (M) is the cornerstone that fulfils all coordination activities and ensures the satisfied quality of filtering services. It should be seen as an entity that first performs both resource estimation and job analysis in order to be able to optimally select strategy that will be asked to perform filtering. As soon as the activated filtering agent has produced the results, manager will then adapt the knowledge about its runtime capabilities based on the measurement of the response time. In a case of receiving any feedback from a user agent about the result relevance, manager agent will perform the adaptation of its job related fitness values.

6.4.1. Job Analysis

The analysis of a filtering job is necessary in order to estimate the applicability of each available strategy for processing a job, having n_w words and n_{ph} phrases. Job properties, which correspond to job related fitness values, being already introduced in Section 6.1.2, are phrase (ω_{ph}), big job (ω_{bj}) and small job (ω_{sj}) load, and their computation is given on Figure 6.10, and formally presented in the following paragraphs.

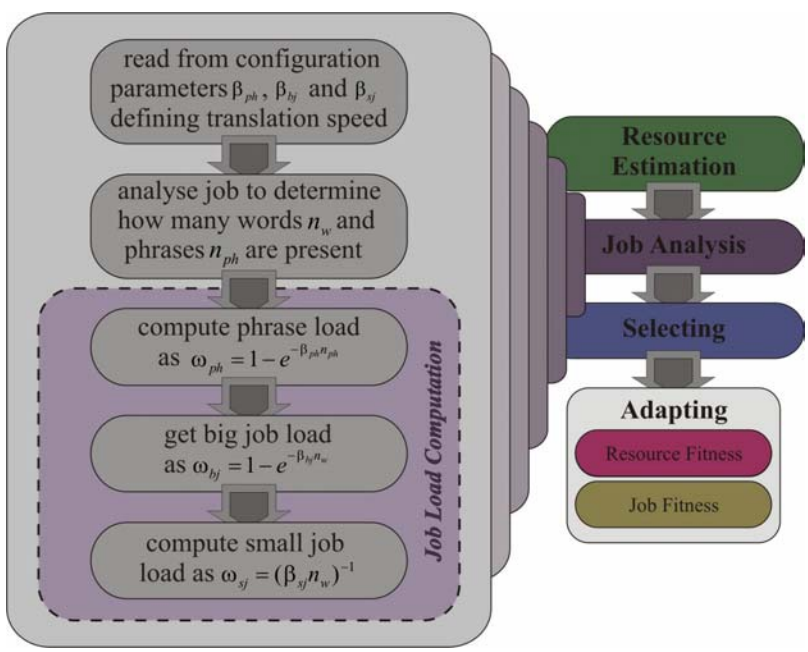

Figure 6.10: Computing phrase, big and small job load values

A phrase load ω_{ph} is computed as:

$$\omega_{ph} = 1 - e^{-\beta_{ph} n_{ph}} \tag{6.14}$$

where β_{ph} ($\beta_{ph} > 0$) is a tuning parameter. More phrases n_{ph} in a filtering job imply first smaller $e^{-\beta_{ph} n_{ph}}$, and consequently larger ω_{ph} value.

A big job load ω_{bj} is obtained as:

$$\omega_{bj} = 1 - e^{-\beta_{bj} n_w} \tag{6.15}$$

where again β_{bj} ($\beta_{bj} > 0$) is a suitable tuning parameter. The higher n_w value results both in smaller $e^{-\beta_{bj} n_w}$, as well as in larger ω_{bj} values.

A small job load ω_{sj} reflects how small the current job is, and it is computed through:

$$\omega_{sj} = \frac{1}{\beta_{sj} n_w} \tag{6.16}$$

where β_{sj} ($\beta_{sj} \geq 1$) is a tuning parameter. Because $n_w \in [1,+\infty)$, it also holds $\omega_{sj} \in [0,1]$, where higher n_w implies smaller resulting ω_{sj} value.

After computing ω_{ph}, ω_{bj} and ω_{sj} values, so-called job content fitness (F_c) can be obtained as follows:

$$F_c = \frac{\omega_{ph} F_{ph} + \omega_{bj} F_{bj} + \omega_{sj} F_{sj}}{\omega_{ph} + \omega_{bj} + \omega_{sj}} \tag{6.17}$$

Each and every ω_y parameter, $y \in \{ph, bj, sj\}$, has a weighing role. In the case where for example a very big job with many words has to be processed, ω_{bj} will have a large value, and $\omega_{bj} F_{bj}$ will dominate in F_c computation. This further means that a filtering strategy with the highest F_{bj} value will probably have the highest overall F_c.

6.4.2. Strategy Selection

The mining of major job properties, resulting finally with job content fitness F_c value, provides the nice possibility of computing a total fitness F_t more intelligently than it has been done in Section 6.3.3 by expression (6.8). Instead of simply specifying $F_t = F_r$ as in (6.8), total fitness can be computed as:

$$F_t = \alpha F_r + (1 - \alpha) F_c \tag{6.18}$$

Parameter $\alpha \in [0,1]$ controls the influence of F_r and F_c, where $\alpha = 1$ will lead to the pure resource based coordination algorithm, as presented in Section 6.3.3. The weighted sum of resource F_r and job content F_c components ensures that not only the abilities of a filtering strategy concerning the current availability of needed system resources, but also its support for the actual job, are taken into consideration. As soon as total fitness values are computed for each and every filtering strategy i, being performed as F_t computation on Figure 6.11, the requested selection probability $P^{(i)}$ can be obtained exactly the same as in Section 6.3.3 by using the expression (6.9).

The procedure of first loading pre-defined α value, then computing total fitness, and finally applying the proportional selection for choosing one filtering strategy is illustrated on Figure 6.11. While proportional selection parts on Figure 6.11 and Figure 6.6 are the same, the greatest strength of resource and job aware coordination scheme is contained in a F_t computation part on Figure 6.11. Its only potential weakness might be unfortunately related with a usage of fixed α value, and future plans will try to establish the adaptation procedure, which will search for α that is optimal in a particular situation.

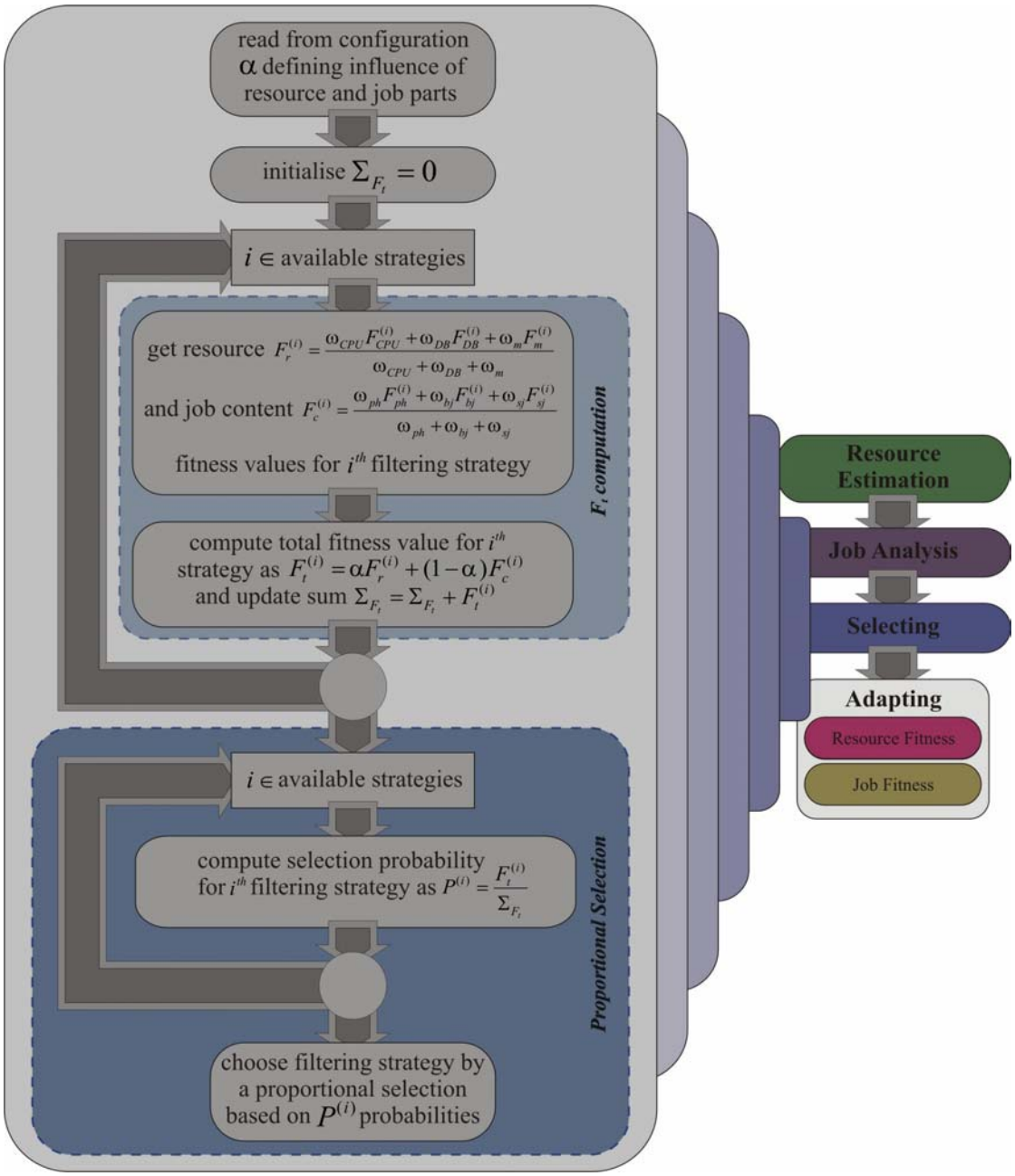

Figure 6.11: Strategy selection when both resource and job content fitness values are utilised

Example 6.3: Figure 6.12 illustrates the application of the introduced filtering strategy fitness values together with resource load and job properties. In the runtime situation ($\omega_{CPU} = 0.37$, $\omega_{DB} = 0.53$, $\omega_m = 0.95$) and for the filtering job with $n_{ph} = 3$ phrases and $n_w = 24$ words, or ($\omega_{ph} = 0.25$, $\omega_{bj} = 0.91$, $\omega_{sj} = 0.04$) for $\beta_{sj} = 1$ and $\beta_{ph} = \beta_{bj} = 0.1$, the obtained total fitness values and the probabilities of being selected are given in Table 6.3.

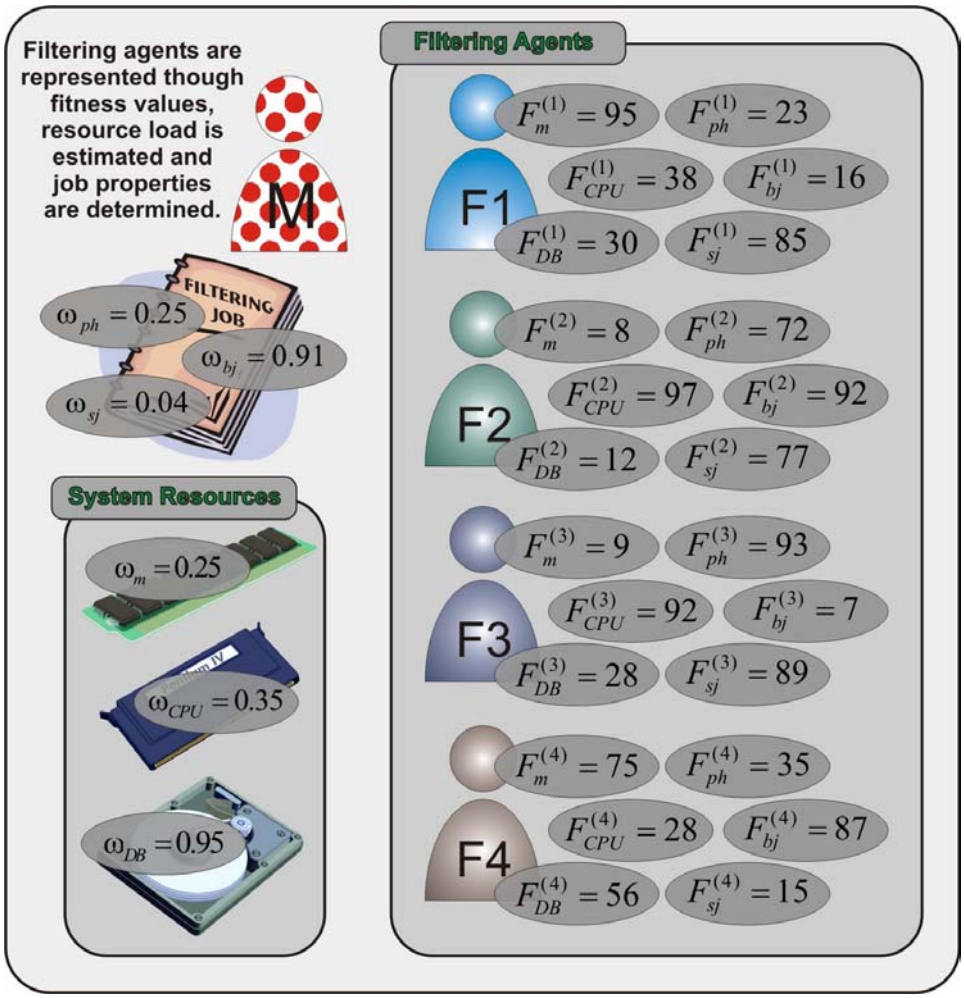

Figure 6.12. Coordination challenge seems to be resolvable after introducing strategy fitness values that formally represent different strategy capabilities towards both resources and job properties

Table 6.3. Resource, job content and total fitness values together with the probability of being selected, which corresponds to fitness and load data from the Figure 6.12

i	F_r [%]	F_c [%]	F_t [%]	P [%]
1	64.98	19.76	42.37	21.64
2	26.94	87.33	57.14	29.18
3	31.04	27.65	29.35	14.98
4	60.16	73.76	66.96	34.19

One should notice that the filtering strategy ($i = 4$), having the reasonable high memory ($F_m^{(4)} = 75\%$) and big job ($F_{bj}^{(4)} = 87\%$) fitness values, has the best chances to be selected because of high memory load ($\omega_m = 0.95$) and the fact that the current job cannot be assumed to be the small one ($\omega_{bj} = 0.91$). On the other hand, strategy ($i = 3$), having in general very high fitness values, will have minor chances to be selected, because of having bad memory ($F_m^{(3)} = 9\%$) and big job ($F_{bj}^{(3)} = 7\%$) fitness.

6.4.3. Fitness Adaptation

After completing a job and receiving the user feedback about the real result relevance, learning through the adaptation of fitness values should ensure that their assigned values reflect as accurately as possible the corresponding abilities of responsible strategy. While the adaptation of resource related F_{CPU}, F_{DB} and F_m values has been deeply analysed in Section 6.3.4, this section is going to be focused on the adaptation of job related F_{ph}, F_{bj} and F_{sj} values, based on a received user feedback.

The adaptation rule for F_y values, $y \in \{ph, bj, sj\}$ is based on the comparison between by a strategy predicted result relevance q_p and the actual relevance q_a that is generated from a user feedback. The used rule can be concisely formulated as:

$$\Delta F_y = \gamma_y \omega_y l(t) \left(\varepsilon - \left| q_a - q_p \right| \right)^{2k+1} \tag{6.19}$$

In the expression (6.19), parameter ε is a tolerance, defining how close the predicted relevance of a result q_P should be to the actual relevance q_a to reward the responsible strategy, k ($k > 0$) increases the influence of large q_a deviations from q_P, γ_y is a tuning parameter, and $l(t) = l_0 e^{-\gamma_t t}$ is a decreasing learning rate which ensures that already learnt fitness values will not be easily destroyed.

The novelty of the expression (6.19), mainly regarding to [17], is in a inclusion of ω_y, which ensures that F_y will be changed more when it corresponds to a more dominant job property, having larger ω_y. In the case where for example, job does nothing with phrases, ω_{ph} will have a small value, and F_{ph} will not be dramatically changed, being reasonable because nobody can expect to gain experience from unrelated jobs.

The learning of right F_{ph}, F_{bj} and F_{sj} values is on Figure 6.13 split on validity check and job fitness adaptation parts. On the one hand, validity check is necessary to figure out which filtering strategy has been actually responsible for providing the result that has got user feedback. As usually user feedback is generously propagated to every single entity that can potentially make benefit from it, a validity check might sometimes discover that nobody locally has been responsible. Such situation can for example occur when another filtering community has produced the result, in which case a received feedback is simply

ignored. On the other hand, in the case where the responsible strategy has been found, its F_{ph}, F_{bj} and F_{sj} values will be changed in the job fitness adaptation part on Figure 6.13, which actually computes the expression (6.19).

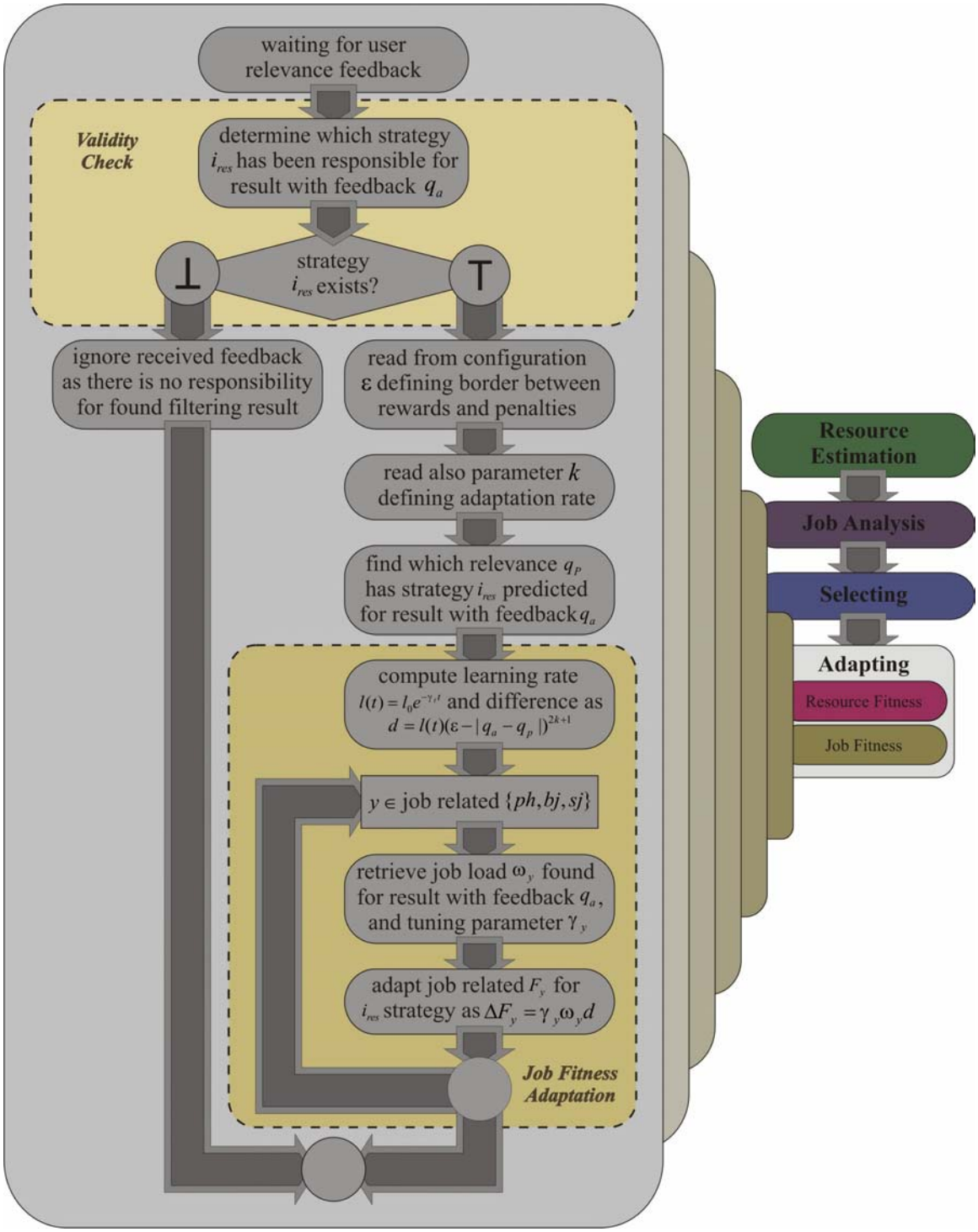

Figure 6.13: Fitness value adaptation

6.5. Self-Healing Coordination Algorithm

Both resource-aware and job-aware coordination schemes, being presented in Sections 6.3 and 6.4, respectively, have assumed that every filtering agent works without failures, and always delivers the requested results. The reality is unfortunately far away from such an ideal situation where filtering agents are failure-less. Their underlying capabilities are maybe not well represented by the associated fitness values, and thus they are potentially not able to successfully work. The runtime situation is also dynamically changing, and it might happen that the already running filtering process does not have anymore necessary resources. The robust coordination scheme has therefore to assume that sometimes the chosen filtering agent is not performing as it has been expected. As a workaround for the situations where failures may occur, this section will propose a self-healing coordination solution. The following sub-sections will show how the learning about the robustness of available strategies should take place to ensure that in the critical situations the reliable filtering algorithm is chosen.

The novel coordination activities, known as healing evaluation and adaptation of new fitness values, are together with already introduced resource estimation, job analysis and strategy selection, given on Figure 6.14, presenting one possible communication between agents. While the roles of User agent (U), as well as Filtering agent (F), are the same as on Figure 6.9 with job-aware coordination scenario, the greatest responsibility is now set on the Manager agent (M). It should be seen as the entity that first performs resource estimation and job analysis in order to be able to select strategy that will be initially asked to do filtering. In a case where a selected filtering agent has failed to respond adequately, the experience that is usable for the adaptation of healing related fitness values has been produced, and manager can therefore adapt its knowledge base. After that, manager will repeat resource estimation and healing evaluation steps in order to find out which strategy is the most promising to be used for recovery. Even though it is not presented on Figure 6.14, this recovery iteration can be repeated until either results are found or the assigned time has expired. In the case where the results have been found, manager will adapt the knowledge about the responsible filtering agent based on a measurement of the response time, and the results will be returned to the user agent. If receiving any feedback from the user agent about the result relevance, manager will perform the needed further adaptation activities.

A concrete solution, showing how these resource estimation, job analysis, healing evaluation, strategy selection and fitness adaptation coordination steps can benefit from the introduced resource (F_{CPU}, F_{DB}, F_m), job (F_{ph}, F_{bj}, F_{sj}) and healing (F_F, F_E) related fitness values will be described in the following sub-sections.

6.5.1. Healing Evaluation

The task, which should be accomplished through healing evaluation, is concerned with formally presenting how urgent is to somehow find results that will be returned to the user. That is simply achieved by defining failure (ω_F) and efficiency (ω_E) load, which correspond to the healing related fitness values, which have been already introduced in a Section 6.1.3.

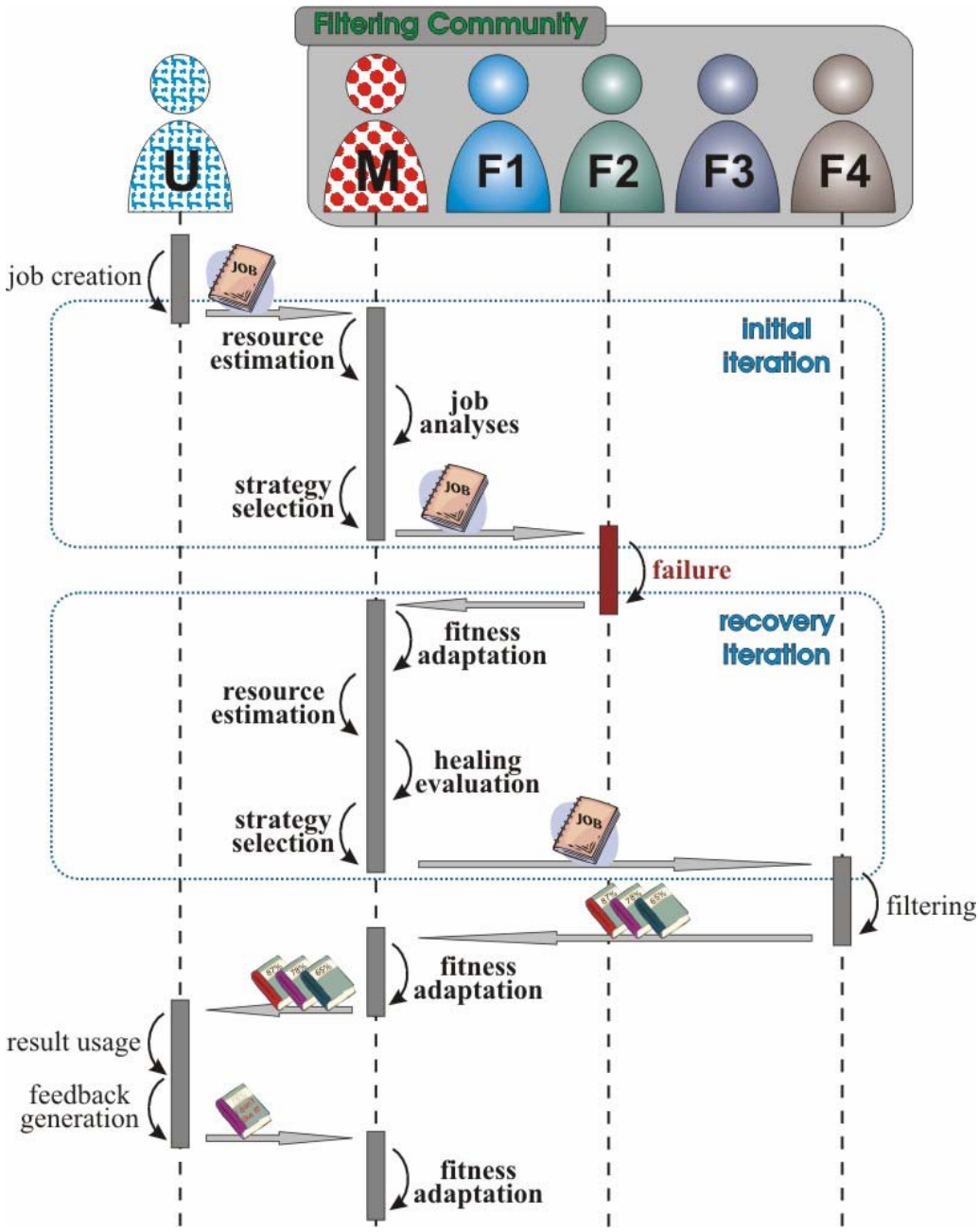

Figure 6.14: System architecture illustrating agent communications

A failure load ω_F is computed as:

$$\omega_F = 1 - e^{-\frac{\beta_F t_D}{t_M - \min(t_M, t_D)}} \tag{6.20}$$

where β_F ($\beta_F > 0$) is a tuning parameter, t_D is the time that is past from receiving a job, and t_M is a maximal tolerable delay for trying to find a filtering strategy that will provide results. After t_D reaches t_M, it becomes very important to try to select a strategy having excellent robustness. Such a defined ω_F always takes values from [0,1] because both t_D

and t_M are positive. Larger ω_F should be understood in the way that it is more important to select a filtering strategy that will work without failures. The complete dependence of ω_F from t_D, being actually a variable that says how much time is already spent, is given on Figure 6.15.

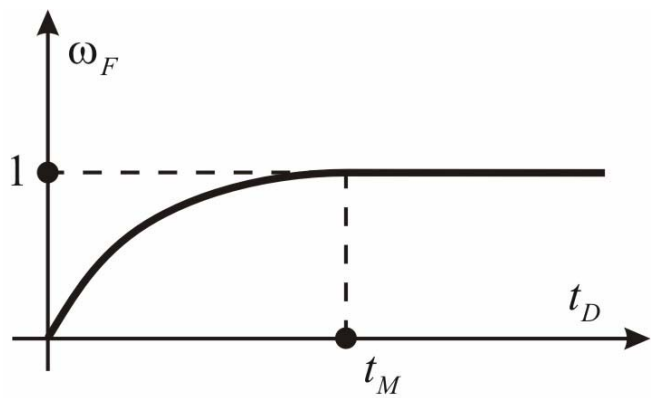

Figure 6.15: Dependence of ω_F from t_D

The idea behind ω_F adaptation becomes obvious from Figure 6.15. In the case where a filtering job has just been received, $t_D = 0$, and consequently $\omega_F = 0$, which means that it is not critical to somehow quickly produce results. But, after some time has been lost, let's say because of a few failures of the previously selected strategies, t_D is not anymore 0, and ω_F grows towards 1. How quickly ω_F will actually grow depends both from β_F and t_M. On the one hand, larger β_F will speed up ω_F grow regarding t_D, and therefore models that even a short delay t_D should be treated quite seriously through large enough ω_F. On the other hand, when a maximal permitted delay t_M is large, the contribution of already lost time t_D to ω_F is decreased. According to that, t_M models the reasoning that when a large delay is allowed, the already lost time t_D is less critical. Finally, for $t_D = t_M$, ω_F reaches its maximal value 1 with the meaning that the job should be delivered to somebody that is known as very reliable in finding results.

An efficiency load ω_E is computed as:

$$\omega_E = e^{-\frac{\beta_E n_{FR}}{n_{NR} - \min(n_{NR}, n_{FR})}} \tag{6.21}$$

where again β_E ($\beta_E > 0$) is the tuning parameter, n_{FR} is the number of until then found results that can be potentially returned to the user, and n_{NR} is the number of results that is asked by the user. Because both n_{FR} and n_{NR} are positive, $\omega_E \in [0,1]$, where higher ω_E means that is more important to select a strategy that is always capable of producing many results. Such a defined ω_E is a decreasing function from n_{FR}, as it is presented on Figure 6.16.

Figure 6.16 shows that ω_E has its maximal value 1 when no results are available ($n_{FR} = 0$), which always occurs for just received jobs. As soon as some results are found,

n_{FR} becomes closer to n_{NR}, and ω_E decreases. That decrease similarly depends on both β_E and n_{NR}. On the one hand, large β_E models user readiness to accept only few found results, and consequently even a small increase of n_{FR} will result in great ω_E reduction. On the other hand, in the case where n_{NR} is large because many results are needed, few found results (small n_{FR}) cannot significantly contribute, and ω_E will not be decreased in a large extent. As soon as enough results have been found, $n_{FR} \geq n_{NR}$, and consequently $\omega_E = 0$.

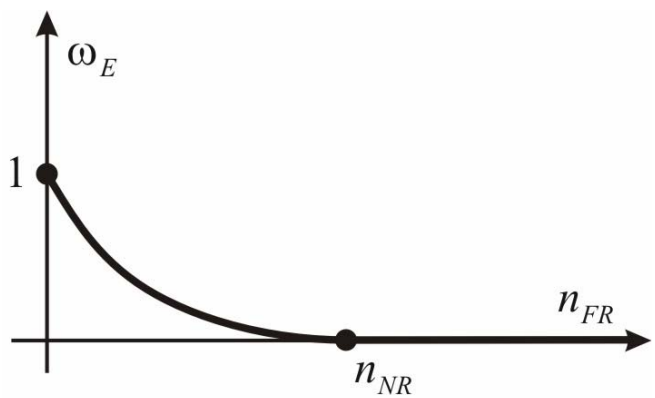

Figure 6.16: Dependence of ω_E from n_{FR}

After computing ω_F and ω_E values, so-called healing fitness (F_h) can be defined as:

$$F_h = \frac{\omega_F F_F + \omega_E F_E}{\omega_F + \omega_E} \tag{6.22}$$

Such a defined F_h shows how a particular filtering strategy, having given F_F and F_E values, is suitable to be selected regarding the current failure ω_F and efficiency ω_E load values, where each ω_z, $z \in \{F, E\}$, has a weighting role. At the same way as F_r does not think about particular types of jobs, and F_c does not care how promising is the current resource situation, F_h only illustrates how a given strategy is reliable to work without failures and does not take into consideration how the found results will be really good (responsibility of F_c) or will be produced fast (duty of F_r).

The purpose of a Figure 6.17 is twofold. Its left part illustrates the computation of ω_F and ω_E through estimating failure and efficiency loads, respectively. The maybe difficult expressions (6.20) and (6.21) are split on simple condition blocks, and thus made more understandable. The estimation of the failure load is first checking whether the failure has occurred, and only in a case of failure analyses how much time has been lost. The method for handling efficiency load similarly checks whether some results are already found, and therefore also easily identifies two extreme cases, being either $\omega_E = 0$ or $\omega_E = 1$. A right part of Figure 6.17 algorithmically represents both initial and recovery iterations from the Figure 6.14. While the job analysis is done only once because user is never changing the already sent job before receiving the first results, other coordination steps are repeated till

either enough results have been found or the assigned time for doing filtering has expired. A self healing is achieved exactly through these multiple iterations, being a fundamental difference as compared to the coordination approaches from Sections 6.3 and 6.4, where only single iteration exists and no recovery is possible.

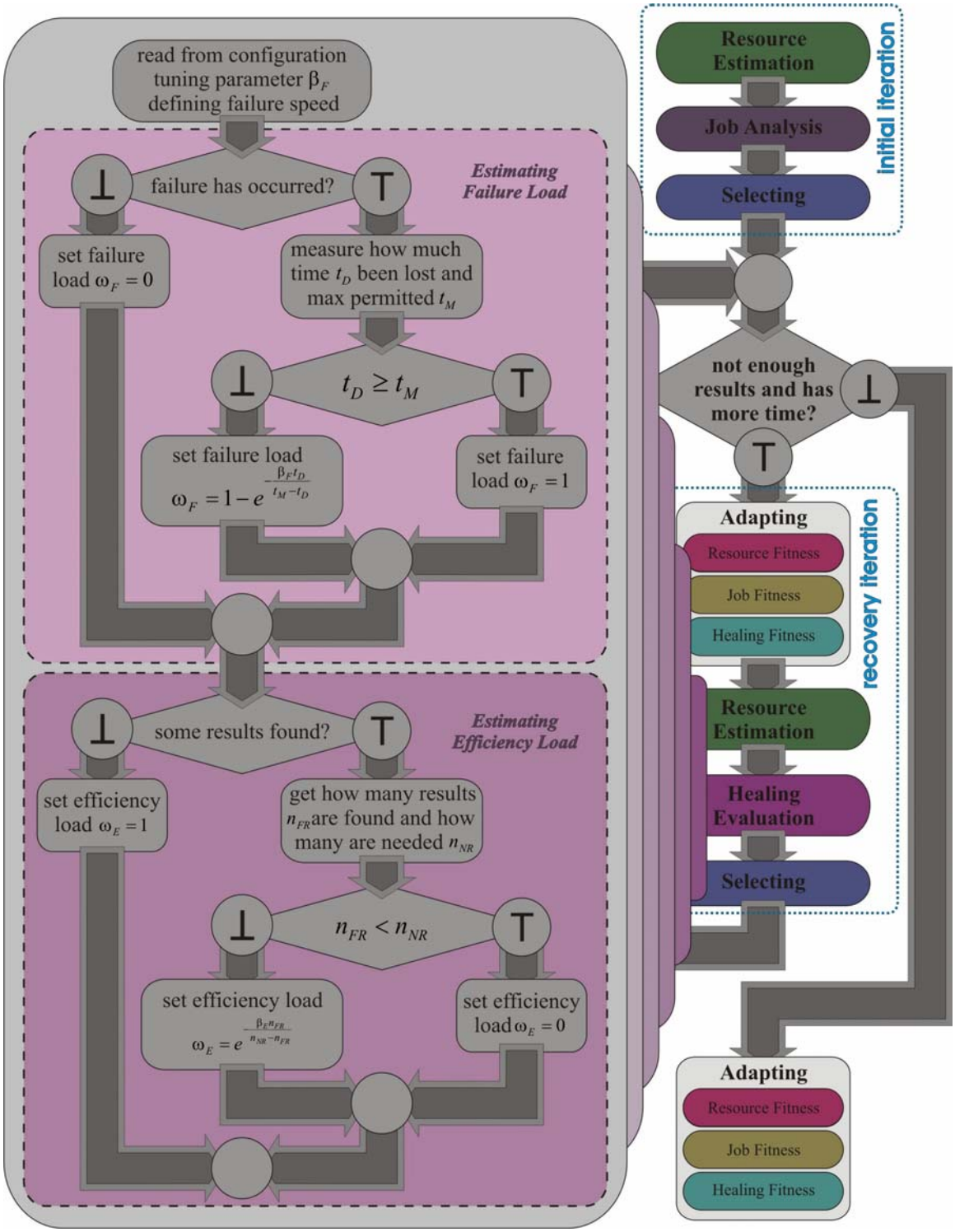

Figure 6.17: Evaluating failure and efficiency load values

6.5.2. Strategy Selection

A fundamental assumption, being behind selecting which strategy will finally perform the requested filtering, says that more information about the potential candidates means hopefully better final selection. The natural procedure for including more data about the capabilities of available strategies is concerned with using not only resource F_r and job content F_c fitness values, but also the recently introduced healing F_h fitness value, while computing the total fitness F_t. The concise expression for obtaining $F_t^{(i)}$, corresponding to the filtering strategy i, can be formally represented as:

$$F_t^{(i)} = \begin{cases} \alpha F_r^{(i)} + (1-\alpha-\beta)F_c^{(i)} + \beta F_h^{(i)}, & i \notin S \\ 0 & , \ i \in S \end{cases} \tag{6.23}$$

In the expression (6.23), S is the set of already selected strategies, which ensures that no one will be chosen more than once for the same job, and the parameters $\alpha, \beta \in [0,1]$, $\alpha + \beta \leq 1$, control the influence of F_r, F_c and F_h on F_t value. While setting $\beta = 0$ will completely diminish healing capabilities and thus produces the coordination scheme as in Section 6.3, setting $\alpha = 1$, $\beta = 0$ will lead to the pure resource based coordination, as in Section 6.4. By using $F_t^{(i)}$ value, the selection probability $P^{(i)}$, making the used selection to be nothing else than the well known proportional or roulette wheel selection [181], is again defined by the expression (6.9) from Section 6.3.3.

The noteworthy property of a strategy selection process, including healing capabilities, is on Figure 6.18 contained in F_t computation part. On the one hand, the total fitness is, as expected, computed by taking into account F_r, F_c and F_h fitness values. A weighting is realised by α and β, being again pre-defined, as it has been the case with α in (6.18). On the other hand, a selection procedure is designed to fully take care of which strategies have been already selected for trying to resolve a particular filtering job. These strategies are identified through the set S, and for them the total fitness value will be automatically set to 0, which will consequently disable their activation in a proportional selection part. The whole coordination algorithm is finished by updating S with the currently selected strategy in order to have the up-to-date information in, potentially needed, new recovery iteration.

Example 6.4: Figure 6.19 illustrates the application of the introduced filtering strategy fitness values together with resource load, job and healing properties. After $9.4s$ the first selected F_2 has failed to deliver any results, or $\omega_F = 0.58$, $\omega_E = 1$ for $\beta_F = \beta_E = 1$ and $t_M = 20s$. The recovery for the filtering job with $n_{ph} = 3$ phrases and $n_w = 24$ words, or ($\omega_{ph} = 0.25$, $\omega_{bj} = 0.91$, $\omega_{sj} = 0.04$) for $\beta_{sj} = 1$ and $\beta_{ph} = \beta_{bj} = 0.1$, has to be organised in a runtime situation ($\omega_{CPU} = 0.37$, $\omega_{DB} = 0.53$, $\omega_m = 0.95$). The obtained resource, job content and healing fitness values, together with the total fitness value for $\alpha = \beta = 1/3$ and the probabilities of being selected, are given in Table 6.4.

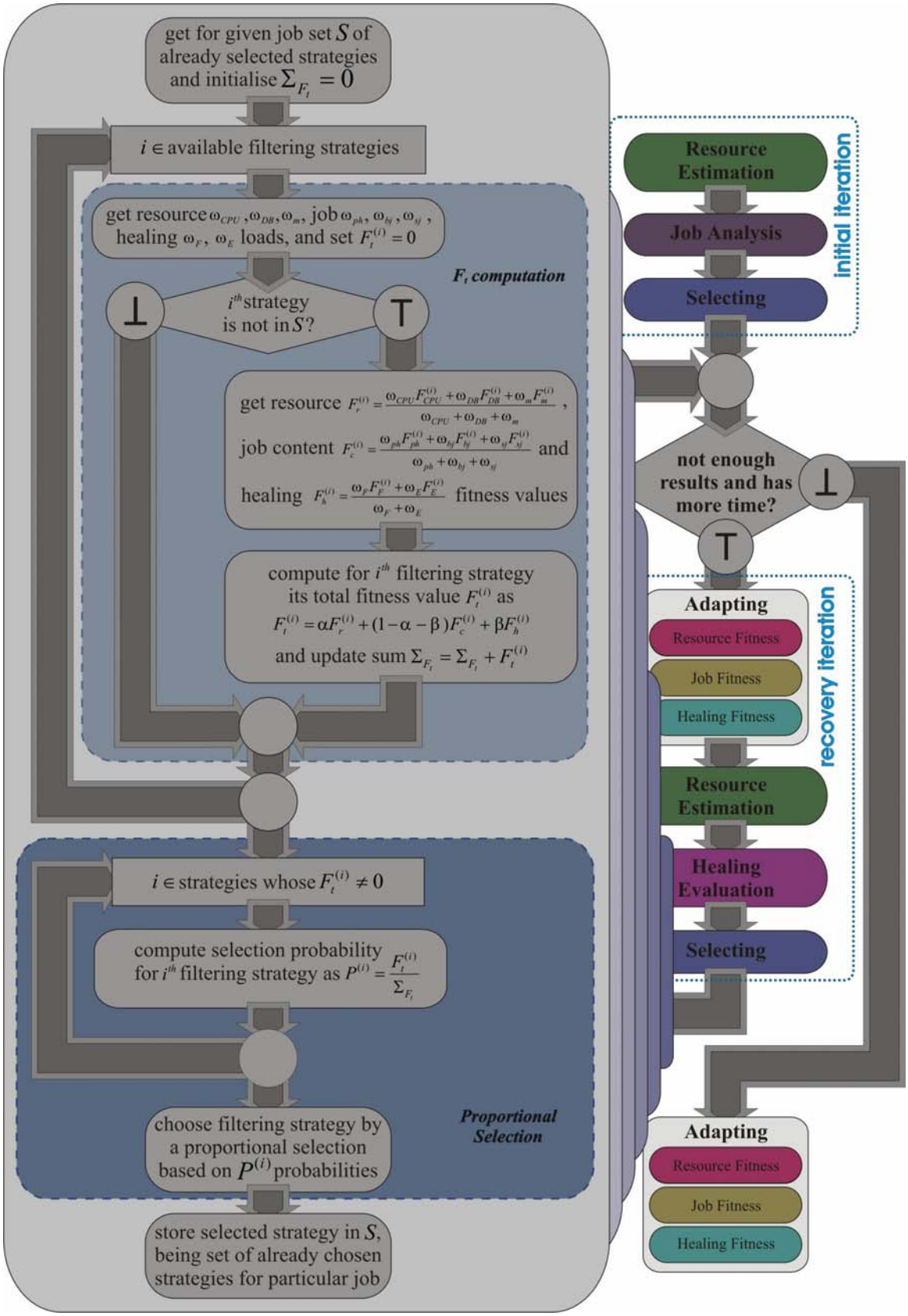

Figure 6.18: Selecting filtering strategy in the case where potentially recovery is necessary

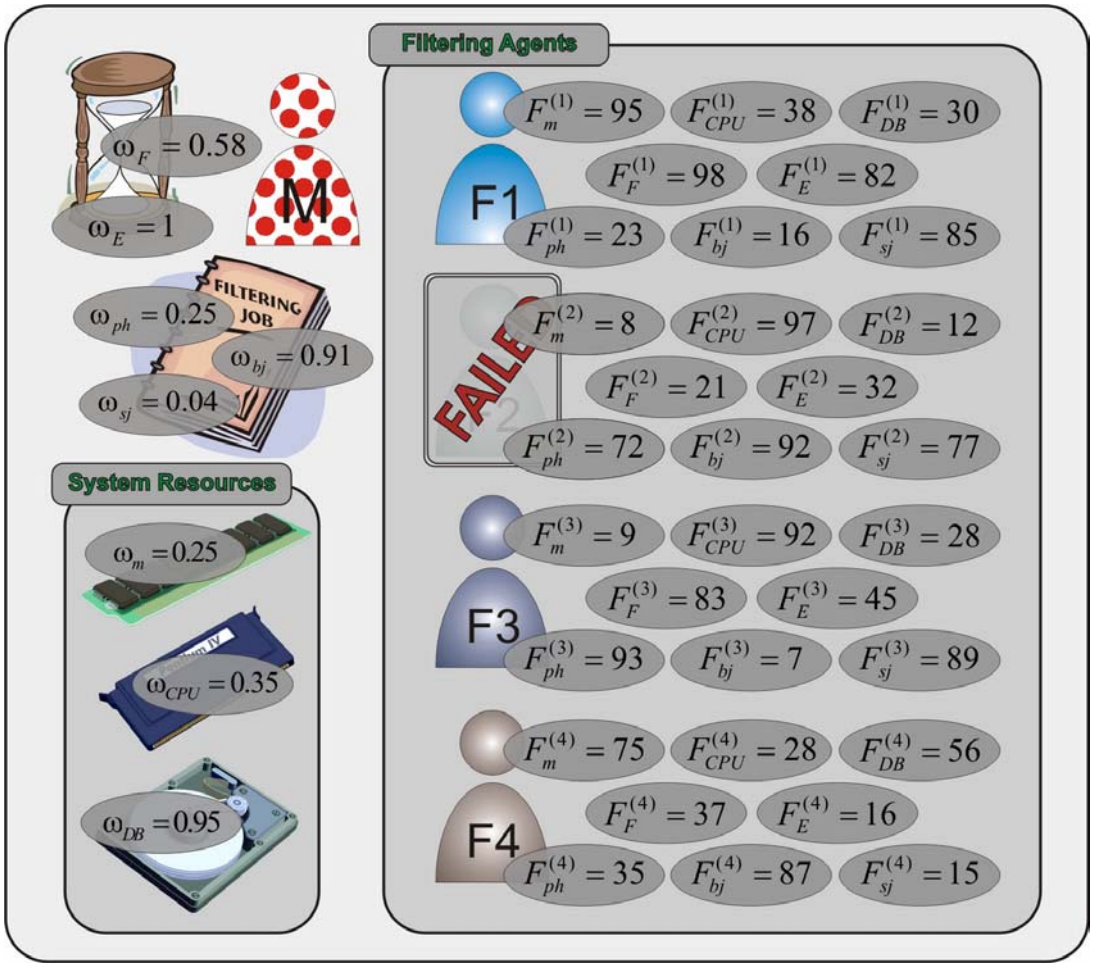

Figure 6.19: Coordination challenge seems to be resolvable after introducing strategy fitness values that formally represent different strategy capabilities

Table 6.4: Resource, job content, healing and total fitness values together with the selection probability, which correspond to fitness, load and delay data from Figure 6.19

i	F_r [%]	F_c [%]	F_h [%]	F_t [%]	P [%]
1	64.98	19.76	87.87	57.54	**29.38**
2	--	--	--	--	--
3	31.04	27.65	58.95	39.21	20.03
4	60.16	73.76	23.71	52.54	26.83

On the one hand, the filtering strategy ($i = 4$), in spite of having the high memory ($F_m^{(4)} = 75\%$) and big job ($F_{bj}^{(4)} = 87\%$) fitness values, which perfectly correspond to the high memory load ($\omega_m = 0.95$) and the current job that cannot be assumed to be the small one ($\omega_{bj} = 0.91$), has not the best chances to be selected by reason of having very bad characteristics to be used for a recovery. On the other hand, a filtering strategy ($i = 1$) is the best choice mostly by reason of being perfectly suited to be utilised for the recovery ($F_h^{(1)} = 87.87\%$). As far as the filtering strategy ($i = 3$) is concerned, in spite of owning

in general very high fitness values, it will have minor chances to be selected, by reason of having bad memory ($F_m^{(3)} = 9\%$) and big job ($F_{bj}^{(3)} = 7\%$) fitness.

6.5.3. Fitness Adaptation

Learning through the adaptation of fitness values should ensure that the given fitness values will precisely describe the corresponding abilities of every single strategy. On the one hand, the used adaptation of both resource F_{CPU}, F_{DB}, F_m, as well as healing F_F, F_E related fitness values, is simply based either on the measurement of the response time or on observing how many results have been found, and thus is completely user independent. On the other hand, a user relevance feedback is the main precondition for adapting job related F_{ph}, F_{bj}, F_{sj} fitness values. While the adaptation of resource and job related fitness values has been analysed in Section 6.3.4 and Section 6.4.3, respectively, this section will focus on the adaptation of F_F and F_E values.

The adaptation rule for F_F is based on a simple reasoning that always after a failure F_F decreases, and reverse, after successfully finished job F_F should increase. But, not each and every failure is the same and the idea is to punish the responsible strategy in a larger extent for a failure that costs a lot. It is very bad to fail after a big delay because much time has been lost, and consequently the adaptation rule should reduce F_F value a lot in such situations. The adaptation rule that incorporates the mentioned properties can be compactly represented as:

$$\Delta F_F = \gamma_F (-1)^{g(x)} l(t) \left(\varepsilon_F + \frac{t_{SD}}{t_M} g(x) \right)^{k+1} \tag{6.24}$$

In expression (6.24), γ_F is the tuning parameter, $g(x)$, $x \in \{failure, success\}$, is two value function, where $g(failure) = 1$ and $g(success) = 0$, $l(t) = l_0 e^{-\gamma_t t}$ is the decreasing learning rate which ensures that already learnt fitness values will not be easily destroyed, $\varepsilon_F (\varepsilon_F > 0)$ together with $k (k > 0)$ defines how big rewards and penalties will be, t_{SD} is a single delay being introduced solely by the selected strategy, and the meaning of t_M is the same as in expression (6.20). By replacing $g(x)$, the expression (6.24) becomes:

$$\Delta F_F = \begin{cases} -\gamma_F l(t) \left(\varepsilon_F + \frac{t_{SD}}{t_M} \right)^{k+1} & , \quad x = failure \\ \gamma_F l(t) \varepsilon_F^{k+1} & , \quad x = success \end{cases} \tag{6.25}$$

From expression (6.25) it is obvious that while in the case of the failure it is taken into account how much time has been lost t_{SD} relatively to the maximal permitted time t_M, in the case of a success reward is fixed and does not depend on t_{SD}. The relative ratio t_{SD}/t_M, and not only t_{SD}, is used to define penalties since failures after a long delay (big t_{SD}) in

time critical situations (small t_M) cost much more, and the responsible strategy should be then seriously punished (large negative ΔF_F).

The adaptation of F_E, which models the ability of a strategy to produce many results, is based on a reasoning that more results means better behaviour. The used rule is:

$$\Delta F_E = \gamma_E l(t)\left(\varepsilon_E - \frac{n_{NR} - \min(n_{NR}, n_{FR})}{n_{NR}}\right)^{2k+1} \tag{6.26}$$

In expression (6.26), $\gamma_E (\gamma_E > 0)$ is a tuning parameter, $\varepsilon_E (0 < \varepsilon_E < 1)$ is the tolerance which defines how close the number of found results n_{FR} should be to the requested or needed number of results n_{NR} in order to reward the responsible filtering strategy, and the meaning of the other parameters is the same as in the expression (6.24). A dependence of ΔF_E from n_{FR} is given on Figure 6.20.

Figure 6.20 shows that a strategy will be rewarded ($\Delta F_E > 0$) always when it holds $n_{FR} > (1-\varepsilon_E)n_{NR}$. For example for $\varepsilon_E = 0.2$, F_E will be increased always when at least 80% of the needed filtering results have been found. While the largest possible penalty ($-\gamma_E l(t)(1-\varepsilon_E)^{2k+1}$) occurs when no results have been found ($n_{FR} = 0$), the greatest reward ($\gamma_E l(t)\varepsilon_E^{2k+1}$) will be given always when as least the asked number of results have been produced ($n_{FR} \geq n_{NR}$). Returning more results than it has been asked is not treated here as a problem because the main purpose of F_E is to point to strategies that are good in finding results. How these results are really good concerning the user satisfaction is addressed through the adaptation of F_{ph}, F_{bj}, F_{sj} values in Section 6.4.3, and it is out of the adaptation scope of these healing related fitness values.

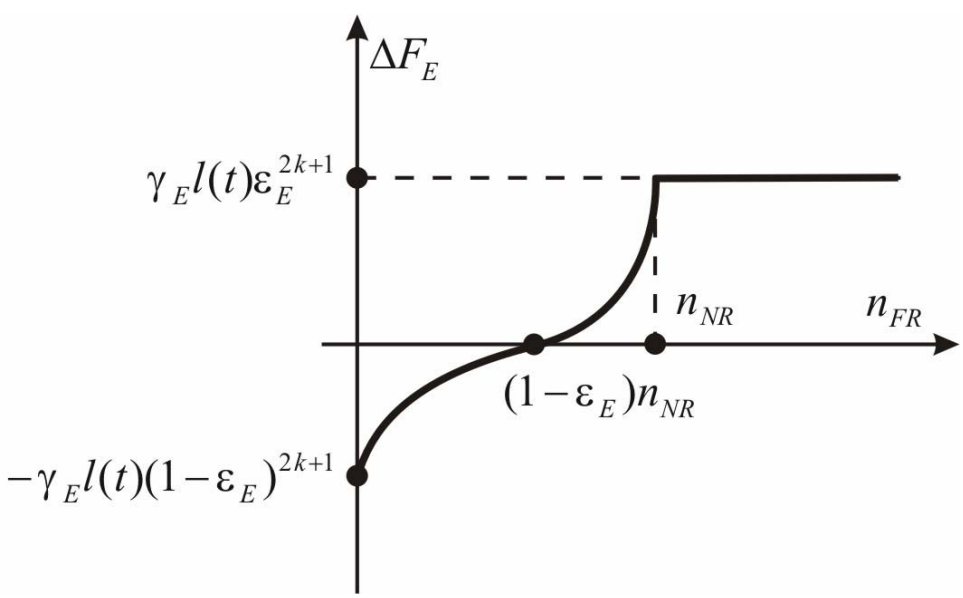

Figure 6.20: Dependence of ΔF_E from n_{FR}

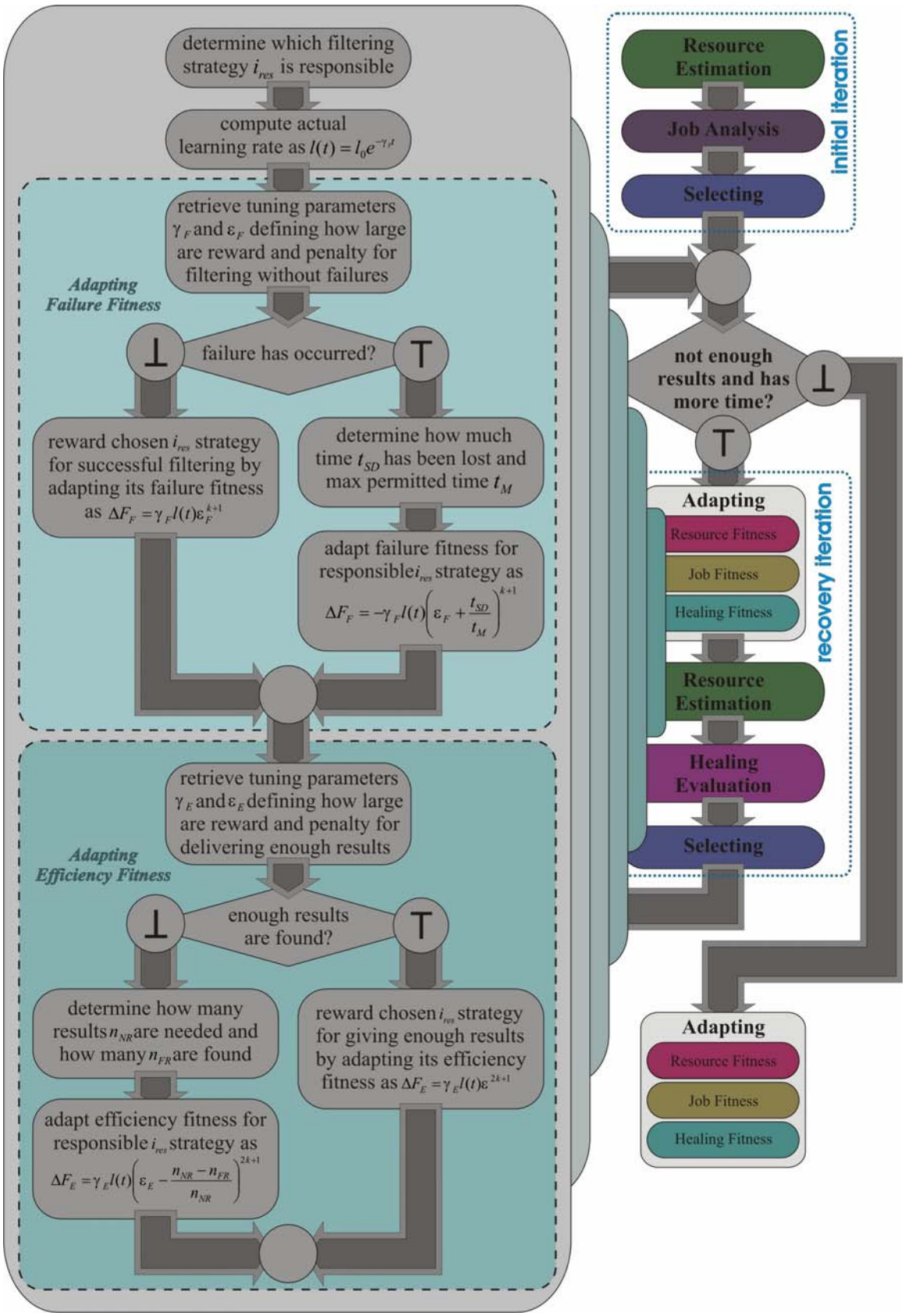

Figure 6.21: Adapting failure and efficiency fitness values

On Figure 6.21 expressions (6.24) and (6.26) are separated on parts depending on the presence of the failure and the occurrence of the situation where enough results are found, respectively. The adapting failure fitness thus clearly identifies when the initially found responsible strategy should be rewarded by maximally increasing its F_F for $\gamma_F l(t)\varepsilon_F^{k+1}$. The adaptation of the efficiency fitness similarly points out what is the maximal possible reward in the case where enough results have been found by the selected filtering strategy.

6.6. Conclusion

Three coordination schemes, looking from the different points of view on the problem of deciding which filtering strategy is currently the most promising to accept a job, have been deeply introduced in this Chapter. All these coordination solutions try to improve themselves during runtime by learning about the unknown capabilities of novel strategies. Such learning is fundamentally supported first by applying proportional selection, which gives chances to each and every filtering strategy to show its value, and then by using the specialised adaptation procedures, which learn from either measuring the response time, observing whether or not the selected agent behaves as expected, or using the explicitly provided user feedback.

A multi-domain applicable coordination scheme, which takes into consideration only to optimally utilise the available system resources, has been presented in Section 6.3. The cornerstone effort of such a resource-aware coordination approach lies in trying to escape the delegation of the actual job to a strategy for which the current runtime situation seems to be unfavourable. The adaptation of necessary fitness values, representing the different runtime capabilities of strategies, is solely based on a duration of filtering, which makes it to be user independent, and represents the additional strength of a coordination approach from Section 6.3.

Section 6.4 has introduced an extension of a pure resource-aware coordination scheme towards taking into account the properties of the actual job. Such a job-aware approach becomes unfortunately a domain dependent due to the made decision to support only jobs in a text domain. While the introduced job-aware coordination approach still achieves the optimal usage of resources by maintaining the resource related fitness values based on the measurement of a filtering duration, the capabilities towards supporting different jobs are learned from the provided explicit user feedback.

The self-healing version of the coordination algorithm, being able to autonomously recover in the case of the failure of a selected filtering agent, has been finally presented in Section 6.5. The simple idea of keeping activating new strategies, until either not enough results have been found or the assigned time has not expired, has been mainly supported by learning how reliably every strategy can work. In the time critical situations, occurring when a previously selected filtering strategy has already failed, this additional knowledge thus enables that the advantage should be given to somebody that is definitely known by its reliability.

Chapter 7
Implementation

The cooperation, exploration and coordination mechanisms, being given in Chapter 4, 5 and 6, respectively, are practically implemented as a manager agent in JIAC IV[1] (Java Intelligent Agent Component-Ware, Version IV), which is a comprehensive service-ware framework for developing and deploying agent systems, covering design methodologies and tools, agent languages and architecture, FIPA compliant infrastructures, management and security functionalities, and generic schemes for user access [20][104]. A noteworthy property of JIAC IV is its CASA (Component Architecture for Service Agents), having a modular internal structure consisting of an open set of components that can be adjusted to the different requirements both at design-time, as well as at run-time. Among one another, components are solely identified by roles they take within an agent, being a way in which dependences only exist among roles, and not among components [227][228][229].

As there is no dependence among components, there is also a wonderful flexibility to easily replace any realised functionality being built inside any of the proposed approaches. Cooperation, exploration and coordination mechanisms are consequently realised through numerous components not only to facilitate the independent changes of whole approaches, but also to make the effortless configuration of any underlying scheme. The cooperation approach can be for example modified only regarding the utilised solution for a database selection without affecting methods, being used for information fusion. The exploration mechanisms can similarly change only the routine, being responsible for finding similar old filtering requests, without having to make any interventions on components, which do either mining or adaptation. The nice opportunities for experimenting with a coordination scheme are even greater, and mostly are concerned with exchanging the utilised strategy selection principle, modifying methods for estimating resources and analysing requests, as well as adapting knowledge about available filtering agents on the more sophisticated manner.

While a currently deployed realisation of any particular functionality has been already deeply analysed and formally presented in its corresponding section from previous three chapters, the discussions about any component will be mostly oriented towards giving the perspectives for its further development. A typical structure of a component presentation

[1] JIAC IV has been developed inside DAI Laboratory of the Technical University Berlin through numerous projects, being funded by the Deutsche Telekom Berkom GmbH

will therefore briefly point out on the related section where the utilised expressions have been given, then usually discuss about the potential problems with its currently deployed realisation, and finally deeply analyse enhancements that are capable to address a noticed drawback. As any of these improvements actually leads towards a new version of a given component that can easily replace the old one, discussions about possible modifications also critically motivate the realised component structure.

To clearly discuss the contribution of every component, they will be presented in the separate sub-sections of this Chapter, being structured as follows. After briefly describing the most successful previous and current JIAC applications, the core contribution will be given in sections that present cooperation, exploration, and coordination components. A chapter is finished with the short summary about the most important improvements that will be certainly taken into consideration in the future.

7.1. Previous and Current JIAC Applications

The fruitful history of the successful JIAC applications has made especially difficult a decision about which its systems are worth to be shortly presented. The challenge is even harder as older systems are maybe no more in use, but about them usually there are solid scientific publications. The younger systems bring another problem as they might be still under construction and maybe there are also no covering papers. A remainder of this sub-section will consequently fairly present four previous and four current JIAC applications.

The JIAC agent-oriented technology has clearly shown to own the necessary potentials of well enhancing the intelligent network platforms [180]. An intelligent and autonomous behaviour, as well as used decentralised control structures, have happened to be the most important JIAC properties for extending existing intelligent network platforms by data services, being able to support the different terminal equipments. The transparent service access via the intelligent brokering has finally seen as the last part of the agent oriented intelligent network that definitely provides much more sophisticated intelligent network services.

A Unified Messaging Agent Service UniMAS [3] has aimed to address the increasing requirements for users' mobility and message universality. The different communication channels, like phone calls, faxes, e-mails, and SMS messages, have been handled by the integrated system of software agents. Instead of having only agents, being tailored for the particular communication channel, there are also agents that provide extra services, such as notification facility and message forwarding. All these services are available anytime and without any static communication, and therefore the users' mobility has been directly supported.

The advantages of agent-based traffic telematics services [2], being critically achieved due to JIAC based realisation, have been found to be an integration of all offered services into a uniform and open environment, an increased efficiency of adapting to an individual need and desire of a particular user, the decrease of communication costs due to the usage of mobile agents, as well as the independence from the particular service provider. These advantages have been practically evaluated by realising the numerous agents, responsible for navigation, on-board computing, routing, event notification, and so on.

The information anywhere and anytime has been a major motivation for implementing one multi-agent system on a core of the JIAC framework, being able to help users travel with the public means of transportations, both in a city area, as well as in a long-distance traffic [5]. It has been practically shown that the application of agent oriented modelling has the great potential while developing travel and reservation systems for the end users. The promising enhancements, mostly regarding planning whole holidays, have been also noted as being possible due to the well elaborated basic structure of JIAC.

The BIB3R has taken into account that each wireless network access technology, like WLAN, GSM, GPRS and UMTS, is good for specific location independent mobile use. It has subsequently analysed problems, which might happen due to a user movement away from its home position. Only some of studied difficulties are such as broken connections, increased bandwidth requirements, saturation of existing connections, and so on.

The large cities and metropolitan areas offer a rich variety of both cultural and leisure opportunities, which have been made easily accessible by BerlinTainment system. It has supported planning activities, being fundamental for events like concerts, films, plays, as well as shopping tours. The BerlinTainment system has practically managed to help users find interesting one-day events, discover many leisure options when only few of them are known, plan the efficient traffic routes, and finally meet people with the similar interests.

As the existing travel booking portals on the Internet typically have many weaknesses, regarding an inability of taking into account user preferences on an activity level, tending to provide weak recommendations that lack diversity, having bad explanation capabilities about selected holiday offers, and so on, the URLAUB system has been developed [12]. Through deploying various filtering strategies, being able to provide fine explanations on the activity level, URLAUB has demonstrated a power of the JIAC oriented multi-agent systems in a domain of recommending holiday offers.

Personal information agent, which is delivering the right information at the right time by accessing, filtering, and presenting recommendations in a situation-aware manner, has been realised as PIA system [19]. It has demonstrated a successful synergy between pull and push techniques, allowing a user to search explicitly for the given information on the one hand, and to be informed automatically about the relevant articles on the other hand. A filtering core of PIA system is made from communities, having not only agents that are tailored to do searching, but also a manager that knows well how to cooperate with other managers, to adapt the actual request to intelligently explore relevant information areas, and to coordinate locally available filtering agents. Such a PIA filtering framework has finally served as a test environment for every single evaluation activity that has assumed the user participation.

7.2. Cooperation Components

The whole cooperation approach, being formally presented in Chapter 4, is generally made to be effortlessly configurable by integrating its cornerstone activities into separate components, which can be easily replaced. The main cooperation steps, being description updating, community estimation, request dispatching, composing results, and reliability adaptation, and being separately analysed in sub-sections of Section 4.3, are consequently

realised as equally named components, as being illustrated on Figure 7.1. The additional *distance computation* DC component could be also found mainly by reason of making the already complex other components to be fully independent from a low level computation of the similarity.

The remainder of this sub-section is going to briefly describe each of the cooperation components from Figure 7.1, and try to motivate the reasons for their separate existence. In the course of giving as good reasons as possible, the many potential improvements will be discussed, being especially a case where *request dispatching* RD, *description updating* DU, and *result composing* CR components are taken into consideration.

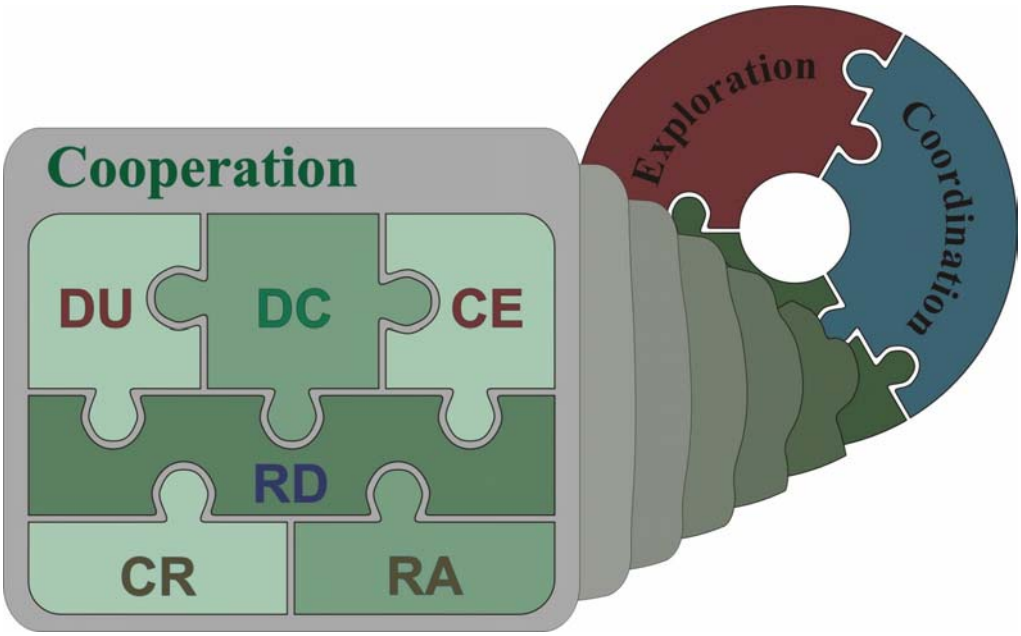

Figure 7.1: Description updating (DU), distance computation (DC), community estimation (CE), request dispatching (RD), composing results (CR), and reliability adaptation (RA) component structure of a manager agent, which is solely responsible for performing cooperation activities

7.2.1. Request Dispatching

A sensitive decision to which communities the actual request should be dispatched in order to find results, which will best satisfy user's information needs, is made by *request dispatching* RD component. While the formal specification of the functionalities behind RD component has been done in Section 4.3.3, a current sub-section will mainly motivate a design decision to have the separate component for the request dispatching activity. The following paragraphs will thus point out to two critical details, being the combination of community estimation with its reliability, as well as deciding to how many communities to dispatch the request. These two problems will be probably the area where the intensive research activities will be performed in the future.

By using the services of *community estimation* CE component, which will be analysed in Section 7.2.3, RD gets an impression how promising the underlying content of every community might be for processing an actual request. Basing a whole request dispatching procedure solely on such community estimation is most probably not so good idea since

the existence of needed underlying documents does not guarantee that the right ones will be found. It can happen that filtering strategies, which are installed in a given community, are not so good, and consequently a whole community is not able to deliver the expected results. A successfulness of each community in finding good results is thus modelled by its reliability, as already presented in Section 4.3.3. The open issue is finally concerned with combining the community estimation and its reliability, being currently simply done as a weighted sum with the fixed weights by expression (4.4). Since these fixed weights reduce the flexibility of a cooperation approach by reason of forcing that all communities are treated at a static manner, the learning procedure for finding the optimal weights can be the possible improvement of RD component. The directions of its development should enable that there are separate weights for each community and every user, with the vision to learn whether for the particular user is more important to take care of the reliability or the available underlying documents.

The second delicate problem, being addressed by RD component, is concerned with a number of communities that should be activated for a particular request. A needed control, regarding to how many communities the request will be dispatched, is currently bound to the expected quality of results, where a greater quality results that more communities are activated. The possible modifications of such a dispatching principle are at least twofold. On the one hand, a number of pending requests, being the ones that are received and that are still waiting to be dispatched, can influence on the number of activated communities. Since every dispatched request has as the final consequence that somebody will have to perform maybe the expensive filtering, more activated communities will result in greater system load. In a case where many requests are waiting to be dispatched, an idea of being more careful while activating communities is thus logical, as available system resources should be more wisely used. On the other hand, the load values of particular communities can help while deciding whether or not to dispatch a request. Dispatching the request to a currently highly loaded community sounds as a problematic decision, which will outcome in either the unacceptable duration of filtering or delivering low quality results. A simple idea can be thus to for example use the load of a chosen community as a post filter, which will disable the activation of already highly loaded communities.

The most appealing modification of RD component can be in a direction of supporting peer-to-peer processing, which assumes that not every community knows all others, and that a request might be subsequently propagated. A current implementation assumes that only requests, being obtained directly from the user, can be resolved through cooperation and potentially dispatched. The P2P improvement will assume both that each request has its time-to-live, as well as that RD more intelligently takes care whether the request might be further dispatched or has to be locally processed. The great challenge behind time-to-live parameter, defining the number of cooperation iterations or steps in which a request can be resolved, is concerned with setting its right value. The useful guideline might be to set larger time-to-live values to requests where a higher quality of results is sought, which will help find right communities.

7.2.2. Description Updating

Keeping community descriptions to be up-to-date is the well-known hard problem and many update policies are known [102][103]. A currently used policy, being encapsulated

into *description updating* DU component, tries to carefully load the underlying network while having as accurate descriptions about other communities as possible. The optimal trade-off is achieved by learning a period of asking each party to send its new description, being fulfilled by tracking the extent in which every community changes its description. While a method for comparing different community descriptions is inside a responsibility of *distance computation* DC component and will be addressed in Section 7.2.6, this sub-section will focus discussions on the strengths and weaknesses of the different possible update policies.

A currently used update policy from Section 4.3.1, which adapts the period of asking for the new content description by expression (4.1), maximally tries to save the available network resources. On the first place, that is done by assuming that communities, which more frequently change their underlying content, should be more often asked to send the new descriptions. The additional saving is achieved by sending these descriptions only on demand, and not periodically. A final optimisation regarding a usage of network is made by seriously limiting a size of the transmitted community content descriptions only to few hundred terms.

Although the introduced optimisations can dramatically save network resources, they usually might have negative implications on the response time. As obsolete descriptions are replaced with the new ones on demand, a dispatching of request has to wait. In a case of requesting several new descriptions that can be unacceptably long waiting and some improvements in the architecture of DU are necessary. The simplest optimisation can be to insist that the communities, which more often change their underlying content, should have fewer terms in their descriptions. Since they are more often asked to send their new descriptions, fewer transmitted terms will on average speed up the updating process. The drawback of such a simple solution is unfortunately in a reduced sensitivity when fewer terms describe the underlying community, which can finally severely limit the ability of RD to dispatch the request to the right communities.

A more advanced update policy, which also bases itself on differently treating various communities, might request new descriptions for important and highly dynamic databases not on demand, but periodically. While the periodical retrieval of community descriptions will not have anymore the direct implications on the response time, it will always load network in the learnt periods of time. By increasing the overall network load, the whole response time will be implicitly enlarged as both the dispatching of a request, as well as the collecting of results will last longer.

The most aggressive update policy is certainly a P2P one, which always broadcast the new descriptions to all interested parties. Such update policy has as the advantage that every community has the perfect knowledge about all other communities. The greatest disadvantage of always informing everybody is unfortunately in loading too much the underlying network, being usually the most critical resource for most distributed systems. A highly loaded network might either slowdown or can even disable the dispatching of a request and the collecting of results by making cooperative communities to be busy with permanently exchanging descriptions. As the main role of communities is to serve a user and provide the requested results, Section 4.3.1 has given up from such an aggressive propagation policy, and has tried to learn when new descriptions are really needed and should be retrieved.

7.2.3. Community Estimation

The *community estimation* CE component is solely responsible for giving a judgement about the applicability of other known communities for resolving the actual request based on their descriptions that are currently available. Since the whole cooperation framework accepts that community descriptions are always more or less obsolete, a major task, being fulfilled by CE component, is concerned with overcoming the problem of not having the perfect knowledge about the other parties. The currently utilised estimation strategy from Section 4.3.2 reduces the found suitability of the underlying content proportionally to the obsoleteness of the possessed descriptions. The expression (4.3) thus forces communities, about which not completely up-to-date descriptions are available, to pay the penalties. A variety of ways relating to how large penalties should be, and how they should be paid, has motivated making community estimation to be fully transparent to other components by encapsulating this functionality inside separate CE component.

The magnitude of penalties is currently globally set to be the same for all communities irrespective to a dynamic of changing the underlying content. As being specified by (4.3), the communities, which more often and more dramatically change their descriptions, will only faster reach the maximal value for the paid penalties. A possible improvement might be in the direction of making a difference between communities, for example by insisting that more dynamic communities pay on average larger penalties. Such variable penalties are finally the extra guaranty for a fair treatment of all communities, being also supported by DU component, which learns how often each party should be asked to send the update, and thus tries to reach the state where the retrieved descriptions are always changed in the approximately same extent.

Since the computation of the suitability of a given community for a particular request has many similarities with the comparison between different descriptions, CE similarly as DU utilises the services of *distance computation* DC component. Such a design decision separates a low level processing of both requests and descriptions, where terms and their weights are somehow used to generate final similarity scores, from a high level reasoning, regarding the combination of a content suitability and the actuality of the used description. While DC takes care of these low level distance computation tasks, CE can focus on the high level heuristics, having the very important influence on the efficiency of the overall cooperation process.

7.2.4. Result Composing

While both DU and CE will primarily help RD overcome a database selection problem, *composing results* CR component addresses the issue of the information fusion. Although CR realises one very light solution for combining the results found by different parties, a reach history of distributed text database technologies has also proposed other approaches for resolving information fusion problems. These other solutions can be easily integrated by replacing only CR without affecting any other components, being the major advantage of separately realising the algorithm for composing results.

The cornerstone idea behind CR component, being formally discussed in Section 4.3.4, tends to optimise a composition of a final result set from the response time point of view. A decision, which results are worth of being shown to the user, is therefore solely made

by using the relevance values, being predicted locally by the community that has offered a particular result. The small intelligence is introduced by changing this locally predicted relevance scores proportionally to the reliability of a responsible community. The system is therefore thrusting more to communities, which have delivered good filtering results in the past, and their recommendations will consequently have greater chances to be finally selected.

A completely opposite solution for handling the information fusion is based on the re-examination of results, and the re-computation of relevance scores. A motivation behind such a re-examination can be easily found as results are found by different parties, which all exploit their own algorithms for computing the relevance scores. The re-computation of relevance values thus seems to be a natural way of ensuring that all results are treated at the same way, as well as fairly ranked at the end. Although such a solution can provide the excellent quality of the finally created result set, computation efforts might make such an approach to be hardly applicable in the time critical situations. The re-computation of relevance scores requires that the recommended documents should be first retrieved, and then re-examined. Due to optimising the network usage, the results are always marked by their symbolic identifiers, and consequently CR will have, in the course of the document retrieval, to really reload articles from a database. As loading of articles can usually last much longer than all other cooperation activities, it becomes clear why the heavy solution with re-examination has given its place to the much lighter re-ranking of filtering results in CR component.

7.2.5. Reliability Adaptation

The reliability value shows how successful a corresponding community has been, and its usage is currently at least twofold. On the one hand, RD uses reliability values to give the slight advantage, while dispatching a request, to communities with a good behaviour in the past. On the other hand, CR thinks that results, which originate from more reliable communities, deserve greater attention. The realistic reliability values are thus important to be found, irrespective whether or not the reasoning behind RD and CR is the right one, and the vital role of improving their values, being encapsulated into *reliability adaptation* RA component, deserves the further attention in the following paragraphs.

The Section 4.3.5 has specified the adaptation policy, which increase reliability always when the predicted relevance is either the same or very close to the actual relevance, got from a user. The utilised expression (4.5) additionally makes the distinction between the closeness of these two relevance values by increasing reliability in a greater extent when their difference is smaller. The important aspects, which are behind (4.5), and which have been mainly ignored in Section 4.3.5, are concerned with utilising the right values for the numerous configuration parameters, being responsible for example for defining a desired adaptation speed, as well as the border between rewards and penalties. While the values of these tuning parameters are currently manually set on the experimental basis, possible improvements might be in the direction of autonomously finding optimal values for them. The right border for example between the rewards and the penalties might be essential by reason of directly defining whether the encapsulated reliability value will be increased or decreased, and consequently should be very carefully addressed in the future versions of RA component.

The auxiliary functionality, which is inside RA component by reason of also dealing with a received user rating, is the feedback propagation. The current realisation assumes that available user feedback should be forwarded to all other known communities, which can afterwards decide whether or not they can make any benefit. While such propagation policy goes fine with the assumption that the user attention is the precious resource that should be carefully used, the forwarding of feedback to communities, which have noting to do with the corresponding results, might be changed. Such and similar improvements, regarding the more intelligently propagating feedback, together with an easy replacement of the used adaptation rule, can be without any trouble fulfilled due to being encapsulated into a separate RA component.

7.2.6. Distance Computation

A sensitive handling of the weighted vectors of terms, which represent either the user information needs or the underlying collection of documents, makes the core of *distance computation* DC component. The single DC component provides the necessary similarity computation to both DU and CE, where DU requires the comparison between different descriptions, formally given by expression (4.2), and CE needs estimation how a content of every community is close to the particular request, specified by (4.3). By ignoring the underlying semantics, which says that weights illustrate the importance of corresponding terms for either user or underlying community, depending on being encapsulated in either request or description, and which is anyway outside the interest scope of DC component, both these similarity computations are practically the same. They are thus installed inside singe DC, which makes that DU and CE become independent from a distance function.

As expressions (4.2) in Section 4.3.1 and (4.3) in Section 4.3.2 show, currently utilised distance function is either pure Jaccard index or its weighted modification. A decision to base a distance computation on Jaccard index is mainly done due to its high efficiency, as well as its solid behaviour in sparce spaces [127]. Such characteristics guarantee that the necessary comparisons will be quickly performed, and that the overall response time will not safer a lot from numerous needed similarity computations. The used modifications of Jaccard index, being especially the one from the expression (4.3), still can be improved to better serve the underlying purpose. The bounding of weights is for example in (4.3) done by *arctg* function, being only one of possible solutions from a list of nice enhancements that might be tried in the future.

The definite argument, regarding the existence of the separate DC component, comes together with the huge abundance of the different distance functions [127]. Not only that Jaccard index knows to compute similarity scores between various objects, but there are many others, such as Euclidean distance, Pearson coefficient, Cosine measure, and so on. An easy integration of a new distance function, with affecting neither DU nor CE, which critically needs similarity computation, is essentially achieved by an encapsulation of this functionality inside a separate DC component.

7.3. Exploration Components

Two fundamental exploration components in a manager agent are both *adding missing attributes* AMA (analysed in Section 7.3.1) and *adapting attribute values and weights*

AAVW (presented in Section 7.3.2), being responsible for deciding which attributes are important to be added in the actual request by reason of being initially omitted, and for correcting the wrongly set attribute both values and weights, respectively. Although these two components are both concerned with adapting the actual request, they are realised by completely different algorithms, and consequently they are encapsulated into the separate components, which finally provide a better flexibility in independently improving any of the used approaches. Since both these components heavily depend on the right setting for exploration or prediction rate, ambitious future plans are concerned with the integration of *predicting adaptation rate* PAR component, being analysed in the Section 7.3.6. This component is going to be responsible for finding the optimal values for adaptation rates, separately for every user, and based on received relevance ratings, in order to maximise its satisfaction.

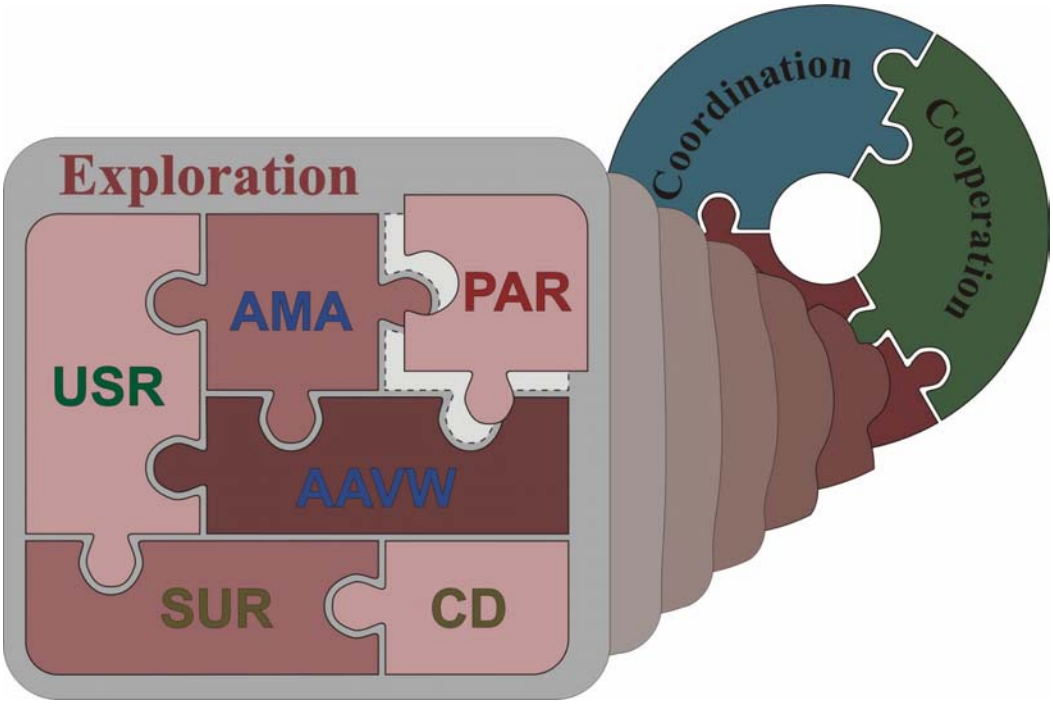

Figure 7.2: Using similar requests (USR), adding missing attributes (AMA), predicting adaptation rate (PAR), adapting attribute values and weights (AAVW), searching for usable requests (SUR), and combined distance (CD) components of a manager, being responsible for exploration tasks.

Both AMA and AAVW components utilise the found expected values and weights for all attributes, which are determined based on the created neighbourhood of the similar old requests. While the creation of the similar neighbourhood is inside the responsibility of *searching for usable requests* SUR component (discussed in Section 7.3.3), the mining of the found old requests is performed by a component that is named *using similar requests* USR, and that is presented in Section 7.3.4. The application of the right distance function makes an essence of the process of finding the similar old requests, and consequently this functionality is encapsulated into a separate *combined distance* CD component, given in Section 7.3.5. Since the numerous functions for measuring a similarity among the various objects are known in a literature, tuning system performances, by fining the best distance measure has been the motivation for a separate CD component. These briefly introduced

exploration components are given on Figure 7.2, and they will be in detail analysed in the following sub-sections, which additionally focus discussions on analysing their potential improvements.

7.3.1. Adding Missing Attributes

The *Adding missing attributes* AMA plays, together with *adapting attribute values and weights* AAVW component, one of main roles, which fundamentally defines how the request will be adapted, and therefore directly controls the level of exploration. While the major design decision behind AMA are going to be discussed in a remainder of this sub-section, AAVW component will be presented in Section 7.3.2.

Section 5.3.3 has formally given an algorithm that AMA utilises while deciding which attributes are worth enough to be added into an actual request. The necessary examination of attributes is done by expression (5.7), which first determines the relative importance values of different attributes in the formed neighbourhood. The addition of new attributes in the actual request can be afterwards performed by deciding to simply add all attributes, whose relative importance is beyond a given threshold. Although such an addition policy is relatively simple and thus quite efficient, its major weaknesses are twofold. On the one hand, the availability of novel attributes, which will be very useful for the actual request, critically depends on the algorithm that is used for creating the similar neighbourhood. It might happen for example that all important attributes from a neighbourhood are already in the actual request. Such unpleasant situation can be unfortunately quite possible as the neighbourhood is typically formed under a direct guidance of a single request. The AMA obviously highly depends on *searching for usable requests* SUR component, being solely responsible for deciding which old requests should be inserted into a neighbourhood. On the other hand, a predefined threshold, being used for deciding whether or not a particular attribute should be added, is currently universally set. It is logical to assume that the same threshold might have the difficulties to perform well in all possible situations. Having for example only few dominant and many not especially important attributes can be a tricky case, where a fixed threshold might have difficulties to identify attributes that are hidden behind the most important ones, which are at the same time not useful by reason of being already inserted. The high importance of having a right threshold in a particular situation therefore becomes a driving force for the development of *predicting adaptation rate* PAR component.

Although the used expression (5.7) completely makes a decision about performing the addition based on an importance of attributes in a neighbourhood, different improvements are possible. The one of them might also take care of the computed similarity between the actual and the old request, which represents the origin of a given candidate attribute that should be potentially added. An underlying motivation is to finally give an advantage to attributes, which are provided by the more similar old requests. The yet another potential enhancement of AMA might be in the direction of taking into consideration not only the similarity, but also the successfulness of old requests, where the requests with better user feedback values will be favoured. These two and many other possible improvements are solely inside the competence of AMA component, which should be the only part, whose replacement can make alive these promising modifications.

7.3.2. Adapting Attribute Values and Weights

As being already presented in Section 7.3.1, AMA component solely takes care that the important attributes, which are missing in the actual request, should be added. Such modifications aim to improve the actual request, and to increase its chances to find good filtering results while intelligently exploring the relevant information areas. An additional opportunity for an improvement still can be found by taking into consideration attributes, which are already present in the actual request. The *adapting attribute values and weights* AAVW component will focus on these already present attributes to fulfil the appropriate modifications on them.

The modification of values and weights, which correspond to attributes that are put by a user in the actual request, has been formally treated in Section 5.3.3 through expression (5.6). The major idea behind (5.6) is to utilise the expected attribute values and weights, whose computation is inside the responsibility of *using similar requests* USR component. These expected attribute values and weights, being exclusively found by USR, are finally used by AAVW as the guideline for modifying the already present attributes in the actual request. The used adaptation rule currently determines a level of modifications by reading a predefined exploration rate configuration parameter. As the exploration rate has a fixed value, modifications will be performed at the same extent for every user in all situations. A more sophisticated behaviour will assume that not each and every user is the same, and therefore different exploration rates are appropriate for them. The learning of the optimal level of exploration should take place, and will be analysed as *predicting adaptation rate* PAR component, in Section 7.3.6.

The transformation of attributes, which are already given in the actual request, should be performed more carefully than it has been the case with the addition of novel attributes. Although the exploration rate controls in which extent the attributes will be modified, it might happen that the expression (5.6) significantly reduces a weight for a given attribute, when its computed expected weight has very small value, irrespective to its original great importance. The influence of the corresponding attribute will be minimised, being maybe quite problematic by reason of being explicitly set by the user, and being expected to play an important role while searching for results. The one possible workaround, which is able to give the strict guaranties that the importance of manually set attributes will not be ever destroyed, is to additionally limit a maximal modification of the weights either absolutely or relatively. Such a simple solution will unfortunately have the inbuilt weakness that one extra tuning parameter, defining how large a change of weights can be, should be set. The more advanced solution for making the safe modifications of weights can give up from an exploited principle, saying that both values and weights should be modified at the same manner, and being a cornerstone of the expression (5.6). Weights should be for example modified relatively to its original values, where the larger initial values permit the greater modifications, and where the already small weights can be never dramatically changed. These and similar appealing modifications of a fulfilled processing, being behind AAVW component, will be certainly taken into account in the course of future investigations.

7.3.3. Searching for Usable Requests

The initial exploration activity, being formally described in Section 5.3.1, deals with the huge abundance of already processed past filtering requests, while carefully selecting

which will form the similar neighbourhood. Difficult scalability requirements to quickly handle millions of available old requests, as well as very restrictive constrains, regarding the quality of a finally created similar neighbourhood, have resulted that the functionality from Section 5.3.1 is realised by two separate components. While the high level creation of the similar neighbourhood is inside the interest scope of *searching for usable requests* SUR, a low level comparison of the highly dimensional requests is fulfilled by *combined distance* CD component. This sub-section will discuss the current implementation and the possible enhancement of SUR, whereas Section 7.3.5 is going in detail to present the CD component.

The fundamental problem, which is solely addressed by SUR component, relates with deciding which old requests are both good and useful enough to be included in the similar neighbourhood. A current realisation from Section 5.3.1 specifies that the actual request solely guides the selection process. The old filtering requests, which are chosen by such a selection policy, are evidently very similar to an actual request. Their high similarity may unfortunately reduce their usefulness for improving the actual request by reason of being very possible that nothing new can be learnt from them. The logical future improvement should thus modify a selection policy in order to ensure the greater diversity of a formed similar neighbourhood. The one of algorithms, which can provide the necessary diversity guaranties, is known as the aggregate creation of a similar neighbourhood. Its basic idea is in using not only the actual, but also the already chosen old requests in the course of a further investigation of still unchecked old filtering requests. Such aggregate formation of a neighbourhood will tend to provide more novel attributes, which can offer much better opportunities to AMA component, while adding missing attributes to the actual request.

The second hard challenge is concerned with the obvious inability to process even the tiny fraction of the available old requests due to very strict response time limitations. The current implementation practically escapes a serious treatment of this problem by simply assuming that at most a permitted amount of time can be spent on the examination of old requests. Such a simple reasoning might results that the really useful old requests remain undiscovered in the tons of others. The logical workaround assumes either clustering or indexing of old requests, which will permit that the potentially useful old requests can be quickly selected. As every system usage typically produces few novel requests that might be useful in a future, a candidate clustering or indexing algorithm has to be able to handle the high dynamic of the insertions of new requests in a knowledge base. A future version of SUR component will probably make the experiments with the different techniques for organising the collection of old requests, and examine a trade-off between the complexity of maintaining such knowledge bases and the provided searching facilities.

7.3.4. Using Similar Requests

The mining of similar old filtering requests, being found by SUR, is formally analysed in Section 5.3.2, and practically realised by *using similar requests* USR component. The main idea, being behind the existence of USR, is in summarising the information that is contained in a neighbourhood of similar requests. The produced summarised information is afterwards utilised as a collection of the expected attribute values and weights by both AMA and AAVW components, which add missing and adapt existing attributes in actual requests, respectively. Such a role of USR will actually put this component to be a bridge

between AMA and AAVW on the one side, and SUR on the other side, being also clearly noticeable on Figure 7.2.

An expression (5.5) is mainly responsible for computing the needed expected attribute values and weights by taking into consideration the similarity between the actual and the corresponding old request, which provides a particular attribute. The reasoning behind is concerned with an idea that the more similar old requests should have the larger influence when the expected values are computed. An advantage of these expected values is mainly contained in the realistically representing the formed neighbourhood. The neighbourhood is also quite compactly represented which makes AMA and AAVW to be more efficient while using these expected values and weights. An unavoidable price, being paid for such an excellent efficiency, is unfortunately contained in the reduced flexibility regarding the different treatments of various attributes. It is expected that for attributes, where outliners are present, the computed expected values and weights are not really as representative as being preferred. For such attributes, a more suitable summarisation can be the one based on a medoids principle [127], where the most important values and weights are taken as representatives. The drawback, which a medoids representation brings, is naturally in the increased complexity, regarding the usage of the summarised information. An extra logic is also needed for deciding how every attribute is going to be treated, and which actually finds the optimal combination between mean and medoids representations of the formed neighbourhood. To facilitate the practical examination of these open issues, the separate USR is created, acting as a gateway between most other exploration components.

7.3.5. Combined Distance

The high importance of comparing filtering requests, as well as the easy modifications of the used similarity function, are motivated the creation of a separate *combined distance* CD component. It is likewise as *distance computation* DC, being studied in Section 7.2.6 where the cooperation components have been presented, responsible for determining the similarity score for any pair of filtering requests. While fulfilling a similarity computation, CD goes one step further by reason of utilising both the generalised versions of Euclidean distance and Jaccard index, as well as combining them to exploit the best their properties. The final reasons for installing the separate CD, in spite of already having DC component, relate to enabling independent experiments in both cooperation and exploration domains, when the distance computation is concerned.

The combined distance has been formally presented in Section 5.3.1, where searching for usable old requests has pointed out to a necessity to compare objects, whose attributes own both values and weights. The generalised versions of Euclidean distance and Jaccard index, which are fully capable to address these requirements, have been therefore given by expressions (5.1) and (5.2), respectively. The most fundamental part, regarding their weighted combination, has been afterwards given by expression (5.4), being the essential advantage of CD component. Due to the usage of Jaccard index CD is well able to handle requests in the high dimensional space. The Euclidean distance also additionally enables to precisely estimate the quality of the found overlapping between the compared requests. Although such a synergy between Euclidean distance and Jaccard index seems appealing, a major problem relates to a necessity to determine weights, being used in their weighted combination. This hard problem is currently easily alleviated by setting the combination

weights based on the fulfilled evaluation in Section 8.3.1 to be fixed in all situations. As it has been found that the role of Jaccard index is slightly more important then is the case with Euclidean distance, their weights are universally set on 0.6 and 0.4, respectively.

The future improvements of the CD component should naturally try to find the optimal weights, which are best suited for the actual request that is used while forming the similar neighbourhood. The extensive experiments are necessary to determine which properties, being behind the actual request, can be a useful guideline while searching for the optimal weights. The simplest possible conclusions might for example specify that Jaccard index should play more important role when the actual request has many attributes with large weights, whereas the advantage should be given to Euclidean distance when the standard deviation of attribute values is large. The found experimental observations can at the end form the base of heuristic rules, which are then used for configuring the optimal distance function in a particular situation. The separate CD component should finally facilitate all these experiments, tending to first recognise the useful features, and afterwards generate the applicable heuristic rules.

7.3.6. Predicting Adaptation Rate

How radical *adapting attribute values and weights* AAVW, as well as *adding missing attributes* AMA components will be, depends mainly from a currently set exploration and prediction rate, respectively. Their large values will influence both that existing attributes are changed a lot, as well as that many novel attributes are inserted, and consequently the actual request is moved faraway from the state that has been manually set by a user. The opposite situation is found when small exploration and prediction rates are not powerful enough to significantly change the actual request, and thus no benefit can be got from the deployed exploration mechanisms. The optimal solution is obviously somewhere between these two extreme cases, and the *predicting adaptation rate* PAR component should aim in the future at finding best suited exploration and prediction rates.

As being already stressed in Sections 7.3.1 and 7.3.2, where AMA and AAVW have been presented, not each and every user is the same, and thus learning the best adaptation rates tends to hopefully increase the user satisfaction. While one user likes to be surprised by getting relevant results that are not completely consistent with a submitted request, the other one prefers the results that are strictly bound to its specified information needs. The situation becomes even more difficult by naturally assuming that the expectations of the single user are quite different for various requests. The explorative filtering results might be desirable when news articles are searched for, whereas the request for scientific papers should be treated without being so radically changed. The roles of PAR component will therefore be twofold. On the one hand, it should be capable to well discover the personal characteristics of every user based on the received feedback values. Users, who are happy with high exploration, will have given larger exploration and prediction rates, and reverse. On the other hand, the fine granulation on a level of a particular user should be performed to make distinctions among various requests. They might be for example clustered, where the separate exploration and prediction rates are assigned to each cluster, and the obtained relevance feedback will not globally affect all adaptation rates, but only the ones where a responsible request belongs.

The separate PAR component should be finally seen to facilitate experiments with the different clustering algorithms. As the user information needs often more or less overlap, it sounds reasonable to use fuzzy clustering [181], which will assign requests to multiple clusters, and also specify their belonging. A level in which every request belongs to every cluster might be used while processing a received feedback as a weight, defining for how much the corresponding adaptation rate should be changed. Even though fuzzy clustering is known to have quadratic complexity regarding the number of objects, its application inside PAR should have no performance problems as only the requests, originating from a single user, should be clustered. An exchange of PAR components should finally enable the practical comparisons between the deployed different clustering approaches.

7.4. Coordination Components

The practical realisations of resource aware, job aware, and self-healing coordination mechanisms, being formally given in Chapter 6 in Sections 6.3, 6.4, and 6.5, respectively, are made to be maximally flexible due to the component structure of a manager agent, as presented on Figure 7.3. The *strategy selection* SS (in detail discussed in Section 7.4.1) is the cornerstone component that takes care of which filtering strategy will be selected, and consequently makes a trade-off between assigning a request always to the most promising one and exploring the unknown capabilities of others. In order to make that this trade-off works well, there are also *healing fitness adaptation* HFA, *job fitness adaptation* JFA and *resource fitness adaptation* RFA components, being in detail considered in Sections 7.4.7, 7.4.6 and 7.4.5, respectively. They take care of improving knowledge about the available filtering agents, and they are realised separately mostly by reason of facilitating an easy replacement of the algorithm that is used for the adaptation of the corresponding healing, job and resource related fitness values.

The *healing evaluation* HE, *job analysis* JA and *resource estimation* SE components, analysed in Sections 7.4.4, 7.4.3 and 7.4.2, enable simple replacement of algorithms that are used for evaluating the importance of a recovery, for checking a given job property, and for estimating a current load of a system resource, respectively. Because of utilising some job properties, such as the requested number of results, HE has to cooperate with JA that is solely responsible for job analysis. Future plans are concerned with the usage of the *situation prediction* SP component (briefly presented in Section 7.4.8), which will most probably cooperate with SE in order to provide not only the estimation but also the prediction of future resource load values.

7.4.1. Strategy Selection

A successful behaviour of a manager agent, regarding the optimally selecting available filtering strategies in a long run, is essentially supported by an installed *strategy selection* SS component. It is directly responsible for making a trade-off between sending a request always to the most promising strategy, and exploring the unknown capabilities of others. The mentioned trade-off is especially hard as typically the most can be learnt by selecting the unknown strategies. These novel strategies might unfortunately perform not very well, and consequently the whole filtering framework risks producing the low user satisfaction. The remainder of this sub-section will first briefly present the currently utilised selection

policy, and afterwards discuss which other solutions can be tried while choosing filtering strategies.

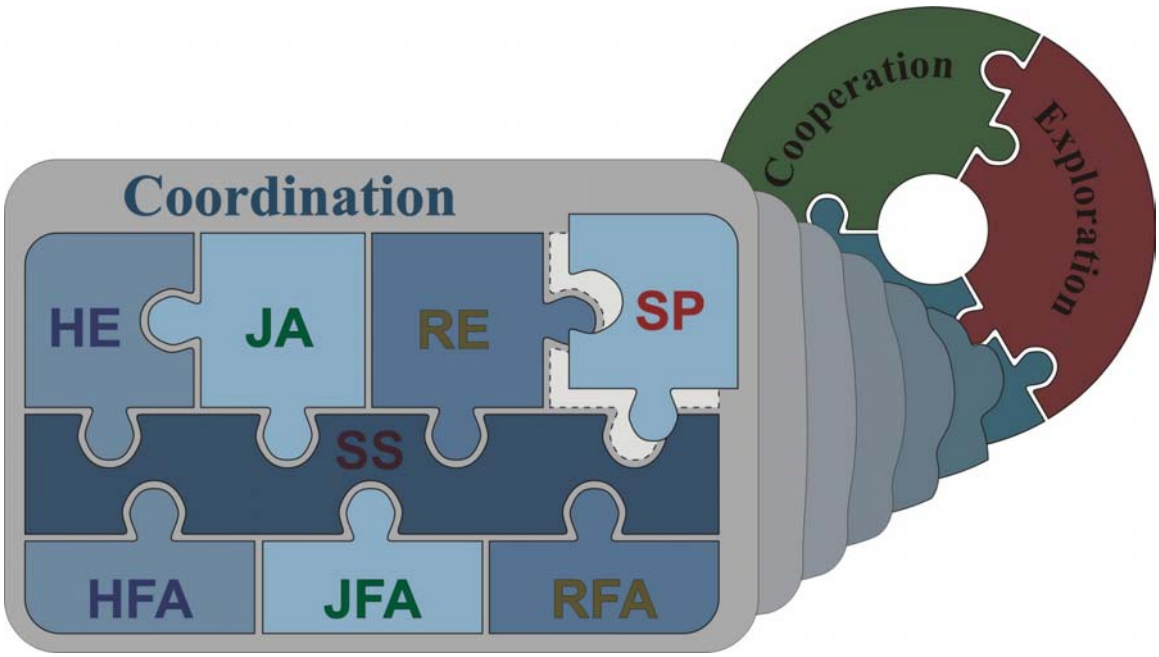

Figure 7.3: Healing evaluation (HE), job analysis (JA), resource estimation (RE), situation prediction (SP), strategy selection (SS), healing fitness adaptation (HFA), job fitness adaptation (JFA), and resource fitness adaptation (RFA) components, responsible for performing coordination activities

The selection of strategies has been formally discussed in a context of resource aware, job aware, and self-healing coordination mechanisms in Sections 6.3.3, 6.4.2, and 6.5.2, respectively. While differences between these three coordination schemes are contained in computing the total fitness values for every available strategy in a particular situation, a currently deployed policy is proportional or roulette wheel selection, as being formally given by expression (6.9). The utilised proportional selection works by specifying that the probability of being selected for the given strategy is proportional to its total fitness value. The exploitation-exploration trade-off seems to be handled by giving the greater chances to better strategies, but also by providing an opportunity even to the worst one to show its real abilities.

There are so many other selection schemes in evolution programming, such as linear and non linear ranking, tournament, (μ, λ) and $(\mu + \lambda)$ selections [181], and therefore it sounds reasonable to discuss whether or not any of them might find its application while coordinating filtering strategies. The one noticeable problem with a proportional selection is concerned with outliners, regarding the computed total fitness values. On the one hand, a filtering strategy, having the extraordinary large fitness, will have maybe unreasonably high selection probability. Such a strategy will be consequently almost always selected, and other strategies will practically have no chance to validate themselves. On the other hand, the strategies with very small total fitness values will have so tiny probability to be selected, and practically they will never get the chance to perform filtering. The discussed problem is hopefully in reality not very serious due to the highly changing both resource availability and request properties. Such a dynamic environment minimises the chances

that a single strategy can dominate, and makes that the simple proportional selection does a good job.

A simple solution for an outliners' problem can be found in linear ranking, which prior to computing selection probabilities, ranks strategies based on their total fitness values. As soon as a ranking is performed, the selection probabilities are computed not anymore based on total fitness values, but by utilising the found ranks. Such a selection principle obviously disables the domination of strategies with the great fitness values, as they will not have much larger selection probability than the others. The strategies with a very low total fitness will be anymore discarded, since their selection probabilities will not be so small. A linear ranking is therefore performing more cooperatively, where even the worst filtering strategy gets reasonably good chances to improve itself. What is also nice, the requested ranking might hardly introduce the significant computation overhead, since the number of available strategies is usually not very large.

An opposite effect, which gives even greater chances to already fitted strategies than a pure proportional selection does, is reached by integrating the elements of the tournament selection scheme. The original tournament selection unfortunately does have no sense to be utilised while selecting the single filtering strategy, as it randomly creates the pairs of candidates, and keeps from each pair the more fitted one. In spite of that, the tournament idea of having the pair of strategies, and in finally choosing the one with the larger fitness, can be used as the post selection for the two filtering strategies, which are initially chosen for example by the proportional scheme. Such a combined selection policy will obviously give big advantages to the more fitted strategies, and practically diminish chances of bad strategies to be selected. The already high selection pressure towards better strategies can be even increased by organising the final tournament selection not on a pair basis, but on several initially selected strategies, from where the most fitted is finally chosen.

The huge abundance of different selection schemes, and even greater possibilities for establishing the combinations among them, should be definitely researched in the course of the author's future work. The separately realised SS component will greatly facilitate these experiments by making a replacement of the utilised selection principle to be fully transparent to other parts of a manager agent.

7.4.2. Resource Estimation

To alleviate dependence from operating system, which is installed on a real hardware where the filtering communities are running, the load of system resources is estimated by measuring the execution time of specially designed expressions. As being already shown in Section 6.3.1 and being practically realised by *resource estimation* RE component, the longer response time will simply mean that the responsible resource, being either CPU or database server, is more loaded, and reverse. The major advantage of such an estimation of resources relates to its easy deployment on any machine irrespective to the supporting operating system. The system functions for returning the load of CPU, being for example typically available under unix system, but not in windows, are not used, and consequently the realised filtering communities become abstracted from a real hardware. Although the given appealing idea about resource estimation has been already realised in Section 6.3.1, where CPU load is measured by expression (6.1) and a database server by (6.2), there are many opportunities for the potential improvements. A remainder of this-section will thus

discuss these enhancements, which relates first in improving the methods that are utilised for estimating a load of a particular resource, and afterwards in integrating the additional resources that might be relevant while getting the impression about the overall runtime situation.

The high importance of measuring the current load of a particular system resource, as well as the strict constrains regarding the maximal permitted amount of time that can be spent on checking a given resource, are motivated the improvements of the basic routines from Section 6.3.1 to become response time aware. These more sophisticated methods for checking both CPU and database server have been presented in Section 6.3.2, through the expressions (6.5) and (6.6), respectively. Wanted response time awareness is achieved by simply limiting the amount of time that can be invested on the resource estimation, and in assigning a maximal load value always when a checked resource has not responded while being analysed. A tricky question, whose treatment has been ignored in Section 6.3.2, is concerned with determining a maximal amount of time that can be spent on the resource estimation only as a configuration parameter. On the one hand, longer checking time will result in more reliable load estimation, being obvious as the temporal oscillations of load values are better smoothed when the test expression is executed longer. On the other hand, spending too much time on the resource estimation is also not very good idea by reason of forgetting that the actual filtering, and not coordination activities, should be the most time consuming operation. It is worth nothing the very fast filtering agent, being capable to find results only within few seconds, when much more time has been already spent on other activities. While this trade-off, between getting a reasonably accurate estimation of a load and not spending too much time on its creation, is currently empirically addressed by judging that one second for estimation activities sounds reasonable, the future versions might decide how much time can be spent on estimation based on the configuration of the underlying communities. While estimation time should be much shorter for communities with very fast filtering agents, the existence of really many slow agents permits that more attention can be invested in resource estimation.

The second question regarding the estimation of CPU and database server load relates to expressions, which are specially designed to be executed, and whose execution time is measured to estimate the load. While the currently utilised expression for estimating CPU load is a sum of Gaussian variables with random standard deviations and means, database server is checked by one aggregate SQL query that computes the sum of the salaries of a randomly selected group of employers on a department basis, as being already presented in Section 6.3.1. The exploited expressions should be as comprehensive as possible while taking into account the different aspects of a given resource. The not well designed SQL query may for example focus only on a speed of reading from index files, which depends mostly from the load of the hard drive where database files are maintained, and ignore the estimation of CPU load where its server is running. Although a currently used aggregate SQL query and a sum of Gaussian variables try to take care of these various aspects, the more appropriate methods for checking corresponding resources are potentially possible. The new versions of RE component will thus certainly investigate the possible benefits while using other expressions for estimating the load values.

Even though the most important filtering resources seem to be CPU and database, the separate RE enables the easy integration of novel resources without affecting at all other

components. It might happen that the particular community contains filtering agents that utilise network while performing its activities. A typical example of such a filtering agent might be a meta-search engine, which finds requested results by propagating the received request to the publicly available search portals, such as Google, AltaVista, Yahoo, and so on. The most critical resource for such a filtering agent is obviously underlying network, whose high load will critically slowdown meta-search functionalities. Since RE currently computes the overall resource fitness value by expression (6.4) that is a simple weighted sum, the new resource is basically integrated by adding the new element in that sum. The problems that should be afterwards addressed are in defining methods that are capable to estimate the load of these new resources. The network load might be for example got by measuring the transfer time, being needed for sending the packets of the given size to the particular destination. The integration of novel resources obviously requires the solutions for many unavoidable challenges, and the fully transparent RE component fundamentally aims at increasing a flexibility of making experiments with their different realisations.

7.4.3. Job Analysis

The most commonly occurring filtering requests in a domain of text documents are in a form of queries, which encapsulate both words and phrases that describe the underlying information needs. Since not each and every filtering strategy is equally capable to handle requests with a highly different number of words and phrases, Section 6.4.1 has presented a formal usage of these simple job properties. A final outcome of a job analysis is a very understandable, job content fitness value, being got by expression (6.17), which basically shows the suitability of every available strategy for processing the actual request. The *job analysis* JA component aims to address the numerous sub-activities for reaching this final judgement, as well as intends to offer the great flexibility while both exchanging methods for using a particular property and taking into consideration the additional characteristics of jobs that might be useful.

While the counting of words and phrases is trivial, the usage of these numbers raises a lot of challenges. On the one hand, the number of phrases should show their importance for a given request, and finally favour strategies, being capable to exactly search for them, in a case where many of them are given. On the other hand, the number of words is used both to define how big the current request is by expression (6.15), as well as to illustrate its smallness by (6.16). Although exact expressions are given in Section 6.4.1 for making these translations, the question whether or not a particular job is big or small still attracts attention. The tuning parameters in (6.15) and (6.16), which essentially define the major characteristics of a used translation, are currently both pre-defined and fixed. Their more appropriate values might depend on the available filtering agents, where weaker support for example for big jobs might speed up a translation of the number of words into a value, illustrating the job bigness.

The most important benefit of having the separate JA component should be found in an easy treatment of other properties, for which there is the adequate support on the side of the installed filtering strategies. Only few typical and appealing properties are such as the number of boolean operators and wildcards, the presence of actuality preferences, and requesting notification services. Not all filtering strategies can support queries with either boolean operators or wildcards by reason of either not having the necessary procedure for

checking boolean expressions or not being able to expand queries with wildcards. A fully different support for such a boolean and wildcard queries consequently makes reasonable to take care of these properties while computing the total job content fitness value. The actuality preferences can be extraordinary important in a news domain, where the usage of filtering strategies, which are able to take into account how old every article is, should be seriously considered. In such a dynamic domain, users are typically requesting also the notification functionalities, which assume that the relevant article will be pushed as soon as being found. The required notification can be consequently the additional job property, which will bias the final selection of strategies towards the ones that are able not only to passively pull, but also to actively push the results. These and similar job properties will be certainly taken into consideration, while ambitiously designing the future versions for JA component.

7.4.4. Healing Evaluation

Delivering some results is usually much better than leaving the user without anything. Not proving the requested recommendations will most probably create an awful filling on a user side that something is not working well, leading unfortunately towards the reduced confidence in the deployed filtering engine. As no agent is perfect, failures might happen. While the seriousness of failures has been formally treated in Section 6.5.1, the *healing evaluation* HE component aims to encapsulate related and realised evaluation approaches, as well as to facilitate their effortless replacement with the potentially more sophisticated versions.

A situation, which occurs after a particular filtering agent has not managed to respond as being expected, has been formally described by both failure and efficiency load values, being computed by expressions (6.20) and (6.21), respectively. On the one hand, a failure load aims to define, how serious a given situation is, regarding an amount of the lost time. On the other hand, the number of already found results shows the importance of selecting a strategy, being able to always provide the requested number of results. Even though the currently utilised solution separately analyses the dimensions of the response time and the number of results, future experiments might try to define the universal load value. Such a single, known for example as healing load value, will be harder to understand and define, by reason of having to make the distinctions between situations, such as having lost much time but also owning some results, possessing no recommendations but having plenty of time, and so on. It will be at the same time more compact, and therefore easier to handle. A separate HE component consequently becomes a perfect playing ground for practically recognising the strengths and weaknesses of these different healing related load values.

The yet another direction of potentially improving HE component relates to the study of a currently available set of results. While the expression (6.21) currently takes care of only the number of found results, its more sophisticated version might check the quality of the available results. The results obviously have different qualities, and therefore most probably only one excellent recommendation is worth more than several that are not truly good ones. The idea for enhancing (6.21) is consequently concerned with not only simply counting a number of found results, but also in using their predicted qualities as weights while summing. Such an obtained sum should be compared not anymore to the expected number of results, but JA component should be used for reading an expected quality from

the underlying request. When the user requests excellent results, and until now only weak results are found, an efficiency load should be larger, to guide a selection process towards strategies, which are delivering not only plenty of results, but also the high quality ones. Such and similar modifications of already introduced healing related load values, as well as the integration of novel ones, are hopefully easily possible due to the existence of the separate HE component.

7.4.5. Resource Fitness Adaptation

An exploitation-exploration trade-off, being established by choosing strategies through proportional selection that honestly gives chances to everybody, becomes evident mostly due to the adaptation of fitness values, which relate to a responsible filtering agent, based on the gained experience. An adaptation of resource related fitness values, being given in Section 6.3.4, and being encapsulated into *resource fitness adaptation* RFA component, is solely based on a measurement of the response time, and therefore can be always done. Although a currently utilised solution works well in most cases by performing k-nearest neighbour search for finding the most similar old runtime situations that are then mined to adapt the actual fitness values, it also has its weaknesses that are going to be discussed in the remainder of this sub-section.

A probably greatest difficulty, regarding the adaptation of the resource related fitness values based on the response time, relates to the high dynamic of changing the actual load. The currently fully available resource might very quickly become highly loaded, and thus it will make great difficulties for all processes, which depend on it. A database server for example can dramatically change its load due to an insertion of many new articles, which have to be pre-processed in order to be later available as the recommendations. As it will be shown in Section 8.4.3, a fluctuation of load values makes an experience, being gained from jobs, whose processing lasts longer than 10 seconds, to be unsafe for an adaptation of corresponding fitness values. The load values, being measured at the beginning of the coordination activity, are probably changed a lot in the course of filtering, and therefore they become much less useful while searching for the most similar old runtime situations. The possible workaround for this instability of load values might be to measure them not only at the beginning, but also at the end of a filtering process, and for example to assume that their average values represent more realistically a responsible runtime situation. The even more accurate solution can estimate the load of resources in the fixed intervals, and subsequently calculate their final values as the arithmetical average. While more frequent resource estimation provides more reliable final values, each fulfilled measurement costs something by reason of having to execute the test expression, as shown in Section 7.4.2. The more sophisticated solution might thus try to learn intervals for computing the load, for example by using the past knowledge about the dynamics of load values in the given period of a day. Such an advanced solution will therefore know to check resources more often when being expected that their load values will dramatically change, and reverse. A richness of these shortly presented potential enhancements, regarding an adaptation of the resource related fitness values, only proves the reasonability for an existence of a separate RFA component. It can be transparently exchanged with its new version, which tries for example to more intelligently gain experience from the longer filtering jobs.

The second noticed problem with the current RFA is concerned with the application of k-nearest neighbour search to first find the most similar old runtime situations, and then to practically summarise them by computing their mean. As being already deeply pointed out in Section 7.3.4 where an exploration usage of the found similar filtering requests has been analysed, mean representation tends to be weak when there are outliers in a formed similar neighbourhood. In a context of similar old runtime situations, such outliers might have the very extreme associated fitness values, which will significantly influence a mean representation. Similarly as in Section 7.3.4, it is worth of trying to use medoids, being a set of few old runtime situations that are centrally located in the formed neighbourhood, instead of computing the mean values. A necessary evaluation, which will show whether or not medoids tend to offer better results, is only maximally simplified by the effortless replacement of RFA component.

7.4.6. Job Fitness Adaptation

The ability of the particular filtering strategy to deliver high quality results is formally illustrated by job related fitness values, being introduced in Section 6.1.2. The job aware coordination mechanisms are mainly improving themselves during the runtime by using the available user relevance feedback to learn the real goodness of a responsible strategy. These improvements are practically realised through the adaptation of job related fitness values, as being deeply discussed in Section 6.4.3, and being accomplished by expression (6.19). Their explicit implementation is given by *job fitness adaptation* JFA component, having naturally its strengths and weaknesses, which will be discussed in the following paragraphs.

The great simplification, being currently utilised while adapting job fitness values, lies in ignoring which user is providing the feedback. Although job fitness values are adapted based on a satisfaction of a given user, all received ratings are equally treated by adapting the same fitness. A finally reached fitness value therefore only represents an average user satisfaction with the particular filtering strategy, being hopefully the valuable knowledge. Such a simple solution is unfortunately not giving enough attention to the unique tastes of users. Not each and every user is the same, and as being already noticed by the Adaptive Web Site Agents in Section 2.6.2, it is reasonably to assume that everybody has its own favourite strategy. The one user may prefer the accurate content based analysis, whereas another user really appreciates an exploration ability of the collaborative filtering. Taking the ratings of these two different users together, while adapting a single job related fitness value, obviously tends to produce average scores for both content based and collaborative filtering agents. An apparent improvement is concerned with maintaining the separate job related fitness values not only for each strategy, but also for every user. Even though the storage requirements might be dramatically increased due to making a difference between various users, a separate treatment of feedback values permits that every user has its own favourite strategy. The coordination engine will be thus able to learn personal affiliations of each user towards available filtering agents, being generally the same goal as Adaptive Web Site Agents are tending to achieve. The advantage of a proposed solution, regarding the one from Section 2.6.2, lies in additionally taking case of the different job properties, such as a number of words and phrases. The future versions of JFA component will thus surely evaluate the evident potential of installing the separate job related fitness values for offering the more personalised selection of strategies.

7.4.7. Healing Fitness Adaptation

The accurate knowledge about the healing capabilities of installed strategies is critical to reasonably bias, when being necessary, a selection process towards the filtering agents that are known to be good for recovery. This knowledge can be acquired and maintained by adapting healing related fitness values, simply through observing the introduced delay in a case of a failure, as well as the number of provided results, relating to the responsible filtering strategy. The possible solution for adapting failure and efficiency fitness values has been presented in Section 6.5.3 by expressions (6.24) and (6.26), respectively. Even though this solution is currently effectively integrated as *healing fitness adaptation* HFA component, there are several natural enhancements that will be discussed in the following paragraphs.

The failure fitness is currently adapted by simply rewarding the responsible strategy always after being able to provide the requested results, and by introducing the penalties in a case of a failure. While the rewards are fixed and predefined, the penalties depend on an introduced delay, where larger response time means bigger caused damage. A possible extension of such an adaptation principle, being compactly specified by expression (6.24), might by in a direction of taking care of the actual runtime conditions while a responsible agent was performing filtering. It sounds fair to reward more if working conditions have been difficult, being formally illustrated by resource load values. The higher load values will thus result that the associated failure fitness for the responsible strategy is increased in a greater extent after a successfully accomplished task. The opposite situation occurs in the case of a failure, when the hard working conditions should influence that the resulting penalties are smaller. It is not logical to punish so much a particular strategy for a failure by reason of having to use highly loaded resource.

The yet another direction for more intelligently performing an adaptation can be found when an efficiency fitness value is concerned. It is currently adapted by simply observing the number of delivered results, where more recommendations mean better behaviour. As being already mentioned in Section 7.4.4, instead of only counting how many results are available, their predicted quality might provide the useful extra information. Results with the higher quality are more important, and they should result that the efficiency fitness is more enlarged. The second enhancement relates to taking into account, how difficult the assigned job is, being formally represented by job load values. An underlying assumption, being behind the consideration of job properties, says that not providing enough results for hard requests, having large job load values, is not as bad as when the request is easier. This also means that a strategy will not be rewarded a lot when an assigned job had small load values, in spite of delivering enough results.

The briefly analysed improvements, when the more advanced adaptation of failure and efficiency fitness is concerned, can be practically checked by replacing the utilised HFA component. Its appealing novel versions will be therefore certainly tested in the course of further increasing the robustness of the deployed coordination scheme.

7.4.8. Situation Prediction

While the activities of pre-processing articles can be always slightly postponed until a load of needed system resources does not become acceptable, fluctuations in a number of

users, posting their requests to a system, cannot be unfortunately influenced in any polite way. This asynchronous system usage results in the dramatic changes of current resource load values, which makes reasonable not only to measure the current load values, but also to try to predict their values in a near future. The computation of resource fitness value by expression (6.4) in Section 6.3.1 has then more supporting data, and consequently better chances to perform much more intelligently. Not only that actual resource load values are available due to *resource estimation* RE, but also *situation prediction* SP component will be able to offer the data about their expected future values. A remainder of this short sub-section will consequently briefly analyse possible solutions for predicting the future load values based on the available logs with their values in the past.

The most appealing prediction algorithm happens to be exponential smoothing, mostly because that method seems to be quite robust in practice, while yielding forecast accuracy that is very comparable to much more sophisticated alternative projection methods [72]. The major underlying reasons are mostly twofold. On the one hand, there is obviously the seasonal component when a single day is seen as one season, having 24 periods. While a system is marginally used during the night, it is logical to assume that in the peak hours, a usage will be much more intensive. On the other hand, both the numbers of users and the insertions of articles have a tendency to increase, and one has to take into account also a trend component. Due to the existence of both trend and seasonal components, the most suitable happens to be triple exponential smoothing, being capable to handle the different dynamic of a system usage during the single day, as well as taking care of the increasing number of users and articles.

Although it seems that theoretically triple exponential smoothing can address the hard challenges of somehow predicting the future load values, its practical realisation by a SP component will prove its capabilities. A separate SP basically also supports experiments with other prediction methods, such as the linear and non linear regression models, which surely open the appealing area for the future researching activities.

7.5. Conclusion

A component structure of a manager agent, being responsible for fulfilling cooperation, exploration and coordination activities, has been given, and the possible enhancements of each used module have been deeply analysed. Due to motivating the existence of separate components mostly by pointing out on promising improvements, this Chapter can be also seen as an extensive discussion about the perspectives for the author's future work.

The most appealing research area, regarding cooperation approach, is an integration of peer-to-peer request processing, being discussed in Section 7.2.1. The P2P enhancement generally assumes that there are no more only few filtering communities, which all know each others, but that requests can be propagated in a course of finding the most promising databases. The also worth of trying is the integration of other techniques for resolving the information fusion, such as the re-examination of results, being analysed in Section 7.2.4.

As a quality of the achieved exploration greatly depends on the formed neighbourhood of similar requests, the best enhancements of the exploration approach has been reported in Section 7.3.3, where the aggregate selection technique has been given. Instead of using

only an actual request, the aggregate neighbourhood is formed also under the influence of already selected old requests, being the way of increasing its diversity and thus improving the adaptation performances. An influence of outliers has been discussed in Section 7.3.4, where the application of medoids instead of mean representations has been discussed.

The interesting modifications of the most important coordination approach have been found to be an integration of other policies for a strategy selection, the more sophisticated estimation of resources, and the prediction of future resource load values. Section 7.4.1 has thus discussed about the usage of both linear ranking and tournament selection, being the major candidates for providing better performances than a proportional selection. The estimation of additional resources, such as network, has been found as very important for specialised filtering communities in Section 7.4.2. Section 7.4.8 has finally briefly given the appealing idea of utilising the triple exponential smoothing, being able to take care of both trend and seasonal aspects while predicting the future load values.

Chapter 8
Evaluation

The cooperation, exploration, and coordination mechanisms aim to provide solutions for finding which distributed databases can be most appealingly searched while providing the requested results, for adapting the actual request to more intelligently explore relevant information areas, and for selecting which existing strategy is the most rational choice for doing needed filtering. A main strength of these mechanisms, being especially noticeable when coordination schemes are concerned, lies in their applicability in real environments, where for example the availability of the needed resources is not taken as the ultimatum. While these real time considerations strengthen the proposed algorithms, they at the same time introduce great difficulties when their evaluation is taken into consideration. Some aspects, such as the reduction of the average duration of filtering due to taking care of the available resources, can be thus tested only in the real world scenarios.

Although the study based on the measurement of the response time and the collecting of user relevance feedback values make the substantial part of this Chapter, the simulated small-scale experiments will be performed as often as possible. Every mechanism will be therefore analysed in such a controlled environment at least once, where the most typical are the evaluation of the cooperation abilities to find the right communities by computing precision and recall values (Section 8.2.1), the testing of the exploration functionality to more efficiently detect relevant requests by using a specially designed combined distance function (Section 8.3.1), and the estimation of coordination capabilities to learn the right fitness values (Section 8.4.3).

The unavoidable evaluation in the real time conditions is made feasible due to the PIA system [19], which has as the foundation previously mentioned cooperation, exploration, and coordination mechanisms. The most of the performed experiments, involving the PIA, are typically based on first changing a particular strategy for fulfilling either cooperation, exploration, or coordination activities, and afterwards monitoring what is happening with the noticeable indicators, such as the response time, and the user relevance feedback. The problematic strategies, whose evaluation during a longer period of time is not feasible by reason of not risking too much the patience of users, are tested by being activated on the request basis, and by maintaining a statistics that shows for which filtering results which particular algorithm has been responsible.

The remainder of this Chapter is structured as follows. The Section 8.1 gives the brief overview regarding the realised searching strategies that are encapsulated into the single filtering agents. The core contribution of this Chapter can be afterwards found in sections which provide the deep evaluation of cooperation (Section 8.2), exploration (Section 8.3), and coordination (Section 8.4) approaches. The main evaluation observations are finally summarised in Section 8.5.

8.1. Supporting Filtering Strategies

The cornerstone of this thesis is in offering a framework that facilitates the integration of different filtering strategies, whose realisation has been clearly set to be out of a scope. Although no its Chapter is dealing with any particular filtering strategy, few of them have been realised in order to enable the elaborate evaluation of cooperation, exploration, and coordination mechanisms. A short description of a prototypical implementation of these strategies will be therefore given in the following paragraphs.

By using a term frequency inverse document frequency scheme, the importance values of different words can be computed, and each and every document can be modelled by a corresponding weighted vector. While the so-called *Large Filtering Strategy* will always build a model with all words from a given document, *Optimal Twenty*, *Optimal Ten*, and *Optimal Five Filtering Strategies* will take into consideration only twenty, ten, and five most important words, respectively. The models, created by these optimal strategies, are thus smaller, and consequently can be faster both loaded into memory and compared with a filtering request. As they are omitting many words, they are at the same time potentially less accurate, and the coordination engine has a chance to decide which one in the given situation can be the best solution.

Since the examination of every single document for each request becomes infeasible even for a collection with the modest size, two different indexing filtering strategies have been also implemented. The first one, named *Inverted List Filtering Strategy*, creates for every word the list of documents having that word. The inbuilt simplification, tending to dramatically reduce a size of inverted lists, is made by not storing the positions of words in the corresponding documents. While a strategy due to such a design decision becomes more efficient, it loses its ability to support requests with a phrase. The second *Position Filtering Strategy* will not utilise such a simplification regarding not storing the positions, and thus will be able to accurately find documents with requested phrases. As this second strategy is naturally more expensive, the trade-off, between providing the accurate results and responding quickly, becomes evident and unavoidable for requests with phrases.

A property of fuzzy clustering [181], to assign documents to multiple clusters together with specifying a degree to which a particular article belongs to a given cluster, has been used as the inspiration for a realisation of a dedicated *Fuzzy Filtering Strategy*. While its strength is in keeping the short cluster summaries in the high speed memory, its greatest weakness lies in a used simplification to cluster documents in advance fixed clusters. The few different versions of this fuzzy filtering strategy are finally implemented by limiting the amount of a memory that is utilised for cashing the cluster summaries, having as the implication that different trade-offs between the response time and memory requirements are possible.

Every mentioned filtering strategy is also exploited for creating its appropriate clone, which will take into account only words from a manually created dictionary. By limiting the vocabulary to few thousands instead to more than half a million, underlying models are much smaller, and consequently the underlying strategies become more efficient. The paid price lies in the lost of a support for all requests with words that are not pre-selected, resulting in the potentially lower quality of filtering results. These clone strategies finally provide even more fruitful playing ground for cooperation, exploration, and coordination mechanisms, which might have many difficulties while resolving the mentioned trade-off problems.

8.2. Evaluating Cooperation Approach

The most important benefit of using the cooperation approach, introduced in Chapter 4, should be found first in making possible to install cooperative communities around many small, more or less dynamic, document domains, and then in deploying many wonderful filtering strategies that are only small-scale applicable. Even though these sophisticated techniques can provide excellent results, a price of splitting a whole document collection on parts has to be paid, and it will be first examined in Section 8.2.1 by comparing both precision and recall values, got for centralised and distributed testing scenarios. The next three sub-sections base afterwards their evaluations on a real PIA system, being the nice testing environment by reason of grabbing daily around 3 thousand new semi-structured and unstructured documents, and having more than 750 thousand already pre-processed articles. Section 8.2.2 examines an ability of a distributed cooperative system to eliminate long lasting filtering jobs, and checks how large penalties have been paid in the feedback domain. The average user satisfaction and filtering duration will also serve as the basis in Section 8.2.3 for evaluating strategy from Section 4.3.1 for updating descriptions. Section 8.2.4 finally tests the abilities of the cooperation scheme to use the inaccurate community descriptions, being crucial in the highly dynamic domains, where a perfect picture cannot be ever reached.

8.2.1. Precision and Recall Comparisons

The ability of cooperation mechanisms, being presented in Chapter 4, to successfully find communities, which will provide the most relevant results for the actual request, can be best assessed by using precision, recall and fallout values [29], being broadly accepted measures for the comparison of different algorithms in the area of information retrieval [127]. On the one hand, both for a precision, being the proportion of retrieved documents that are relevant, and for a recall, representing the proportion of relevant articles that are retrieved, greater values correspond to a system with better properties. On the other hand, fallout relates to a proportion of irrelevant documents that are retrieved, and thus smaller values are preferred. More formally, these measures are defined as:

$$
\begin{aligned}
p &\equiv precision = n_r^{(r)} / (n_r^{(r)} + n_{ir}^{(r)}) \\
r &\equiv recall = n_r^{(r)} / (n_r^{(r)} + n_r^{(nr)}) \\
f &\equiv fallout = n_{ir}^{(r)} / (n_{ir}^{(r)} + n_{ir}^{(nr)})
\end{aligned}
\tag{8.1}
$$

where $n_r^{(r)}$, $n_{ir}^{(r)}$, $n_r^{(nr)}$ and $n_{ir}^{(nr)}$ correspond to either retrieved $n^{(r)}$ or not retrieved $n^{(nr)}$ documents that are either relevant n_r or irrelevant n_{ir} in a way that is concisely presented in Table 8.1.

To compare the information retrieval system based on a centralised single community with the one that has multiple cooperative communities, a small controlled domain $D_{all}^{(500)}$ with only 500 documents is formed. For each of 10 testing filtering requests $\{FR_i\}_{i=1}^{10}$ at least 10 and at most 20 documents are manually selected as the relevant ones, as formally given in manual examination block on Figure 8.1. In the first scenario, filtering requests $\{FR_i\}_{i=1}^{10}$ will be resolved by utilising a system that has only one centralised community. That centralised community will use as the underlying document collection $D_{all}^{(500)}$ and will return 10 results for every request from $\{FR_i\}_{i=1}^{10}$. The necessary computation, which checks whether every found result is relevant or not, is marked on Figure 8.1 as a block, named the centralised solution. The computed precision, recall and fallout values, which correspond to a centralised system with only one filtering community, are finally given in Table 8.2.

Table 8.1: $n_r^{(r)}$, $n_{ir}^{(r)}$, $n_r^{(nr)}$ and $n_{ir}^{(nr)}$ definitions

	relevant	irrelevant
retrieved	$n_r^{(r)}$	$n_{ir}^{(r)}$
not retrieved	$n_r^{(nr)}$	$n_{ir}^{(nr)}$

Table 8.2: Precision, recall and fallout values for the system with single centralised community for each of 10 filtering request from $\{FR_i\}_{i=1}^{10}$, where the last row gives corresponding average values

i	$n_r^{(r)}$	$n_r^{(nr)}$	$n_{ir}^{(r)}$	$n_{ir}^{(nr)}$	p [%]	r [%]	f [%]
1	8	12	2	478	80	40	0.42
2	7	11	3	479	70	38.89	0.62
3	7	5	3	485	70	58.33	0.61
4	8	7	2	483	80	53.33	0.41
5	5	11	5	479	50	31.25	1.03
6	7	5	3	485	70	58.33	0.61
7	9	8	1	482	90	52.94	0.20
8	6	4	4	486	60	60	0.82
9	8	6	2	484	80	57.14	0.41
10	7	6	3	484	70	53.84	0.62
	7.2	7.5	2.8	482.5	72	50.41	0.58

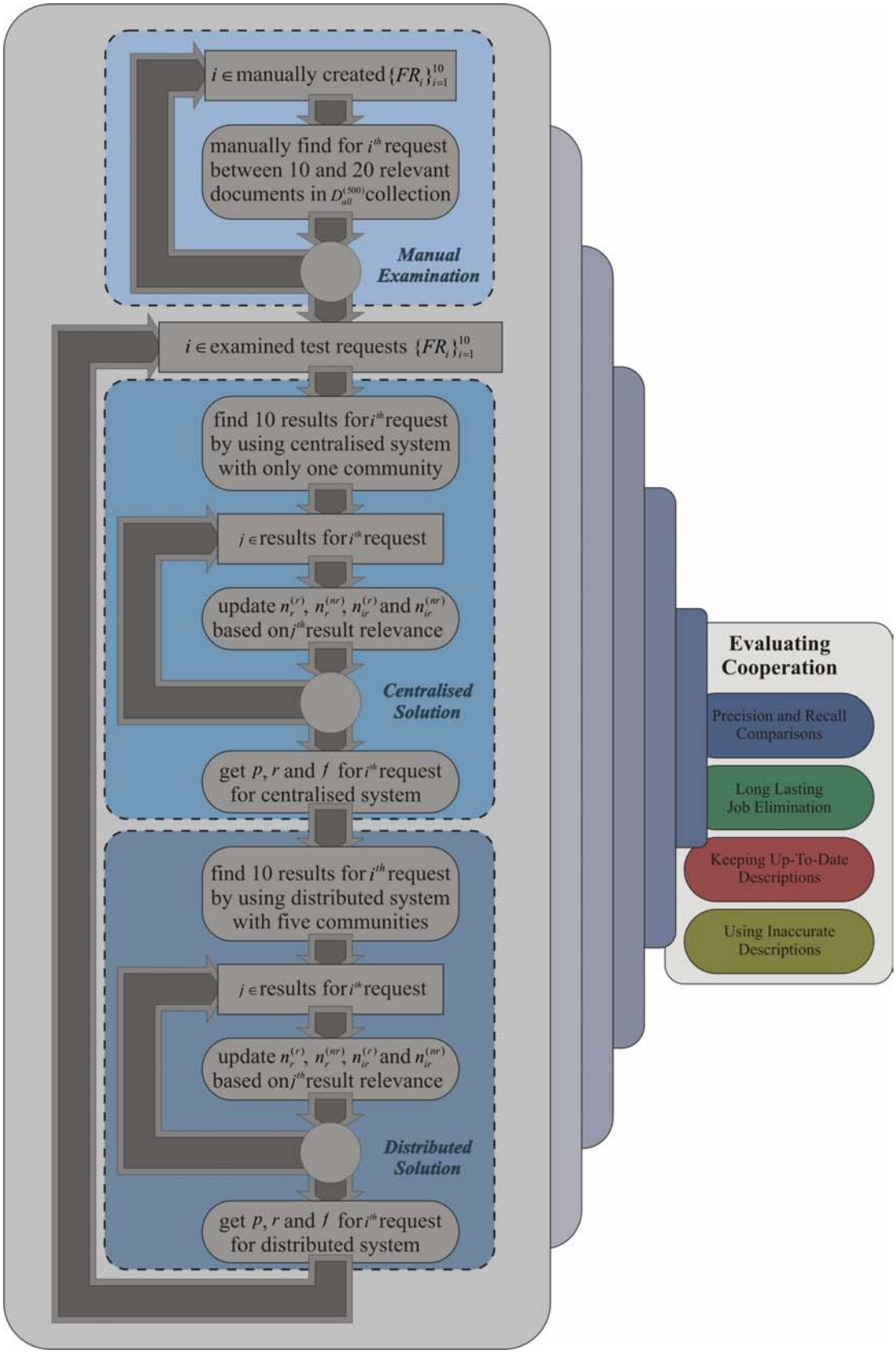

Figure 8.1: Computing precision p, recall r, and fallout f for centralised and distributed system

In the second scenario, domain $D_{all}^{(500)}$ is manually split on agent technology $D_{at}^{(108)}$, telecommunication $D_{t}^{(83)}$, renewable energy $D_{re}^{(147)}$, political news $D_{pn}^{(75)}$ and sport news $D_{sn}^{(87)}$ domains, where the index in the exponent says how many documents belong to the particular sub-domain. Around every sub-domain a separate community is installed, and the same 10 filtering request $\{FR_i\}_{i=1}^{10}$ from the first scenario are again resolved by using such a system with 5 distributed communities. The needed processing for again obtaining precision, recall and fallout values, being given in Table 8.3, is on Figure 8.1 represented as a distributed solution block.

Table 8.3: Precision, recall and fallout values for the system with 5 distributed communities for each of 10 filtering requests from $\{FR_i\}_{i=1}^{10}$, where the last row gives the corresponding average values

i	$n_r^{(r)}$	$n_r^{(nr)}$	$n_{ir}^{(r)}$	$n_{ir}^{(nr)}$	p [%]	r [%]	f [%]
1	7	13	3	477	70	35	0.63
2	7	11	3	479	70	38.89	0.62
3	6	6	4	484	60	50	0.82
4	6	9	4	481	60	40	0.82
5	5	11	5	479	50	31.25	1.03
6	7	5	3	485	70	58.33	0.61
7	8	9	2	481	80	47.06	0.41
8	6	4	4	486	60	60	0.82
9	7	7	3	483	70	50	0.62
10	7	6	3	484	70	53.85	0.62
	6.6	8.1	3.4	481.9	66	46.44	0.70

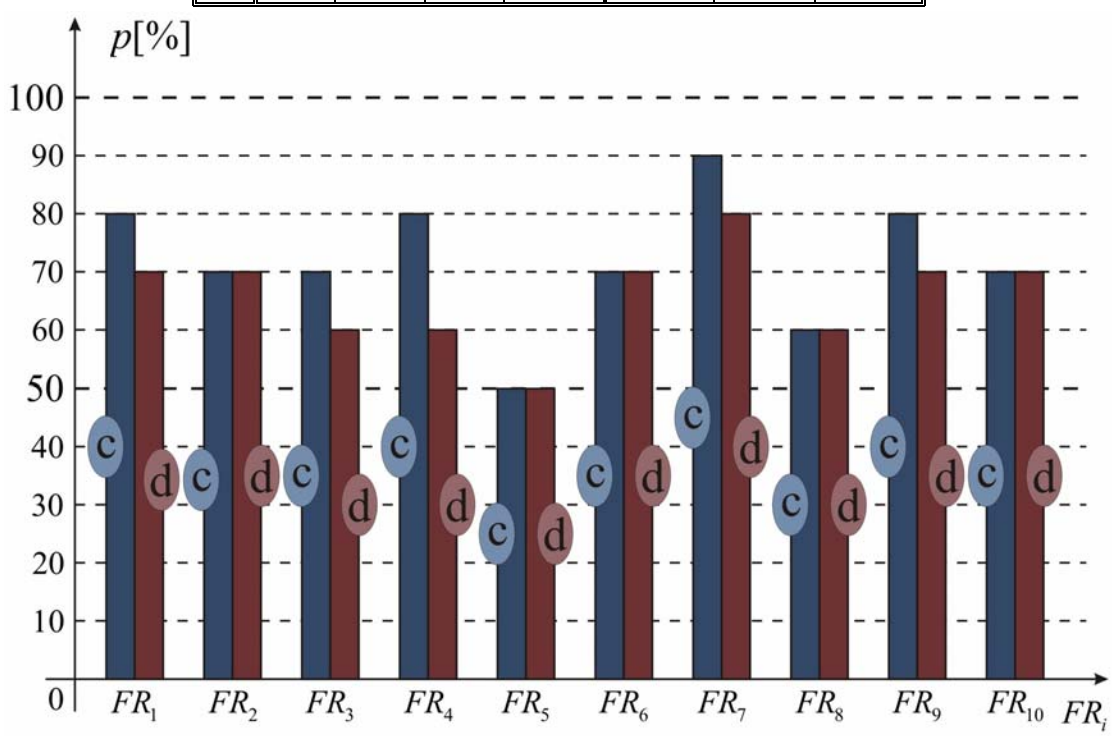

Figure 8.2: Direct comparison between old centralised (blue bar filling marked with c) and new distributed (red bar filling marked with d) precision values that are got for each of 10 tested requests

The precision values, got separately for each filtering request $\{FR_i\}_{i=1}^{10}$, are presented on Figure 8.2, where the architecture of a system with the single centralised community is compared with the distributed solution. While being natural that a centralised system has to provide better results due to taking into account always all available documents, a very encouraging observation is that with an exception of a filtering request FR_4, the obtained precision values are either exactly the same, or quite close, for both tested systems. Even the largest decrease in the precision value of 20%, being got in the case of FR_4, becomes not so bad when thinking that in the real situations the solution with the single centralised community will either deliver results after the unreasonably long delay or crash due to the huge abundance of documents that cannot be supported.

Figure 8.3 similarly gives a direct comparison between recall values that are obtained for every single filtering request in two tested system architectures. Because sometimes the relevant results are spread over many communities, the distributed solution with five cooperative communities is not able always to provide as good recall as the centralised system. The decrease of separately got recall values is with the exception of request FR_4 always within 10%, being even tolerable when the all other advantages of the distributed cooperative solution are not taken into consideration.

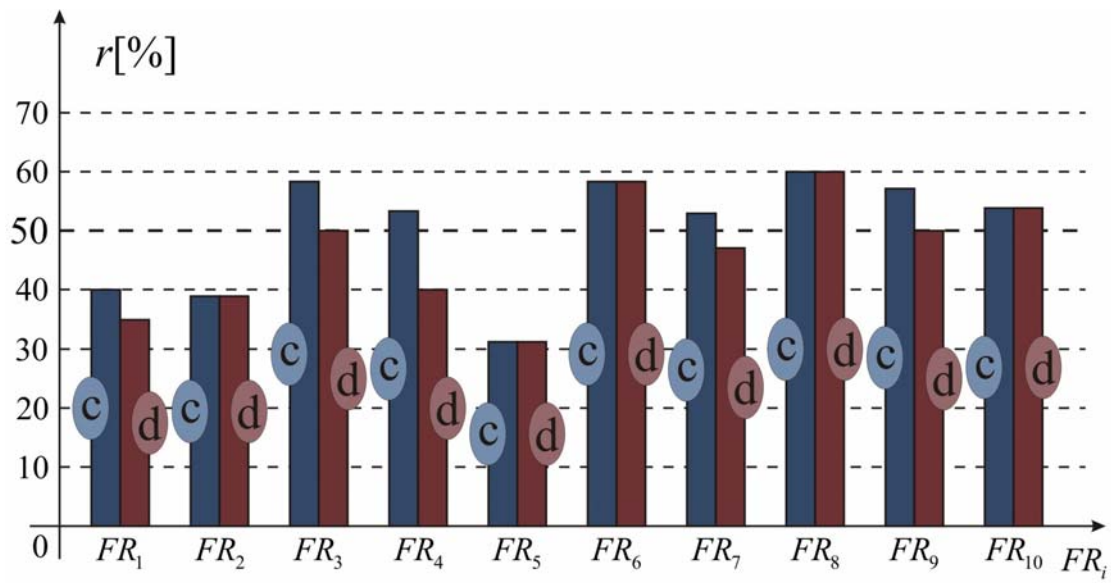

Figure 8.3: Direct comparison between old centralised (blue bar filling marked with c) and new distributed (red bar filling marked with d) recall values that are got for each of the 10 tested requests

The fallout values that are got in both examined scenarios are generally very small by reason of returning only the very small fraction of all irrelevant documents that exist. The most critical requests, where the increase of fallout values is the largest one, are FR_4 and FR_7, as being given on Figure 8.4. For these two requests the fallout values have been two times increased, probably due to the insufficiency of relevant documents in selected communities, and a requirement that always 10 relevant results should be returned. Even though relatively compared the difference between some fallout values before and after seems huge, the absolute comparison uncover that a situation is not so terrible. As it can

be seen from the Table 8.2 and Table 8.3, the request FR_7 has for example resulted that a centralised solution has delivered one irrelevant article and a distributed system only two such unwanted documents, which is not so bad by taking into consideration that all other results have been good.

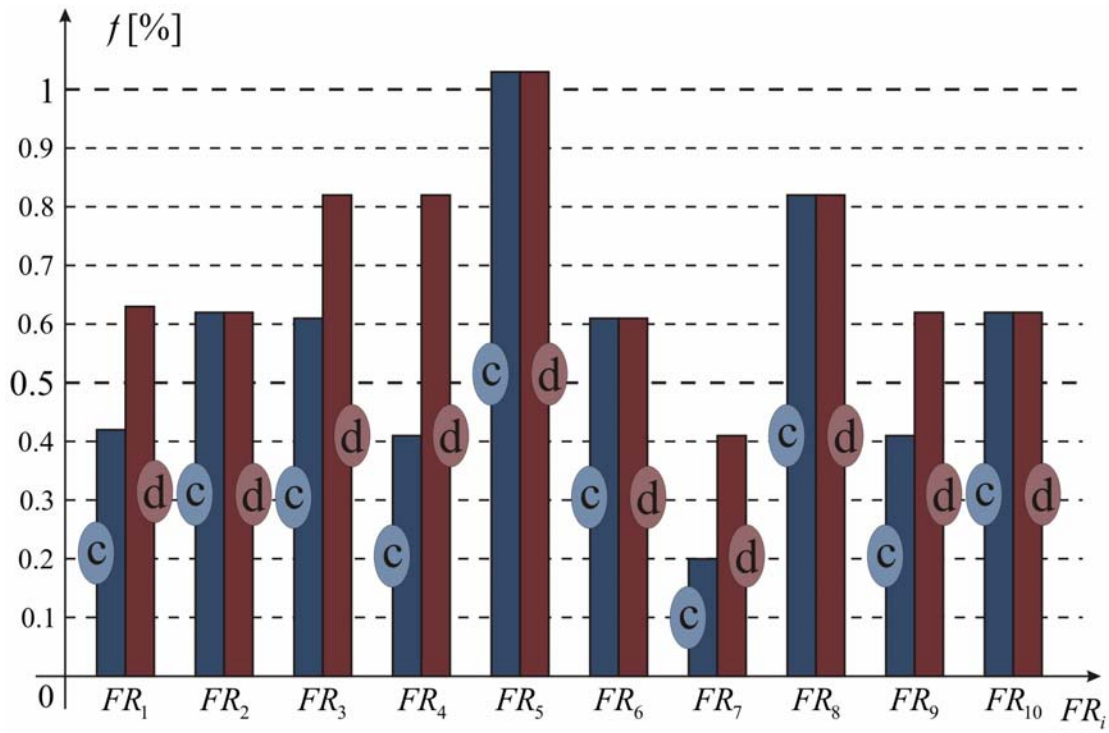

Figure 8.4: Direct comparison between old centralised (blue bar filling marked with c) and new distributed (red bar filling marked with d) fallout values that are got for each of 10 tested requests

As it has been expected, the centralised system has slightly better precision, recall and fallout values than the distributed one. While the average precision and recall values have decreased for 6% and 4%, respectively, the average fallout value has increased for 0.13%. This is unfortunately the price that has to be paid for the much better flexibility, which is crucial as being hardly possible that always everything can be stored in a single database. The much shorter duration of filtering can be also expected to be obtained in real cases, where small-scale scalable filtering strategies, which cannot be used on huge collections, but which can provide excellent results while being deployed in small environments, are deployed. This shorter duration of filtering can be afterwards seen as the extra factor that can increase the satisfaction of inpatient users.

8.2.2. Long Lasting Filtering Job Elimination

One of design goals, being behind the realisation of cooperation strategies, is certainly concerned with the elimination of long lasting filtering jobs, being usually a consequence of the small scale filtering strategy applicability inside the single community. In the case where the millions of documents should be altogether supported, it might happen that the deployed filtering strategies have scalability problems, which will sometimes result in the unacceptably long response time. As the users are impatient beings, a used idea is to split

the whole collection of documents, and to bind its smaller parts on separate communities. While Section 8.2.1 has compared the precision and recall values that have been obtained as the result of a comparison between the systems, either being centralised or having five cooperative communities, this sub-section will examine what is happening with the user feedback and response time values in the real PIA environment when the same scenario is repeated.

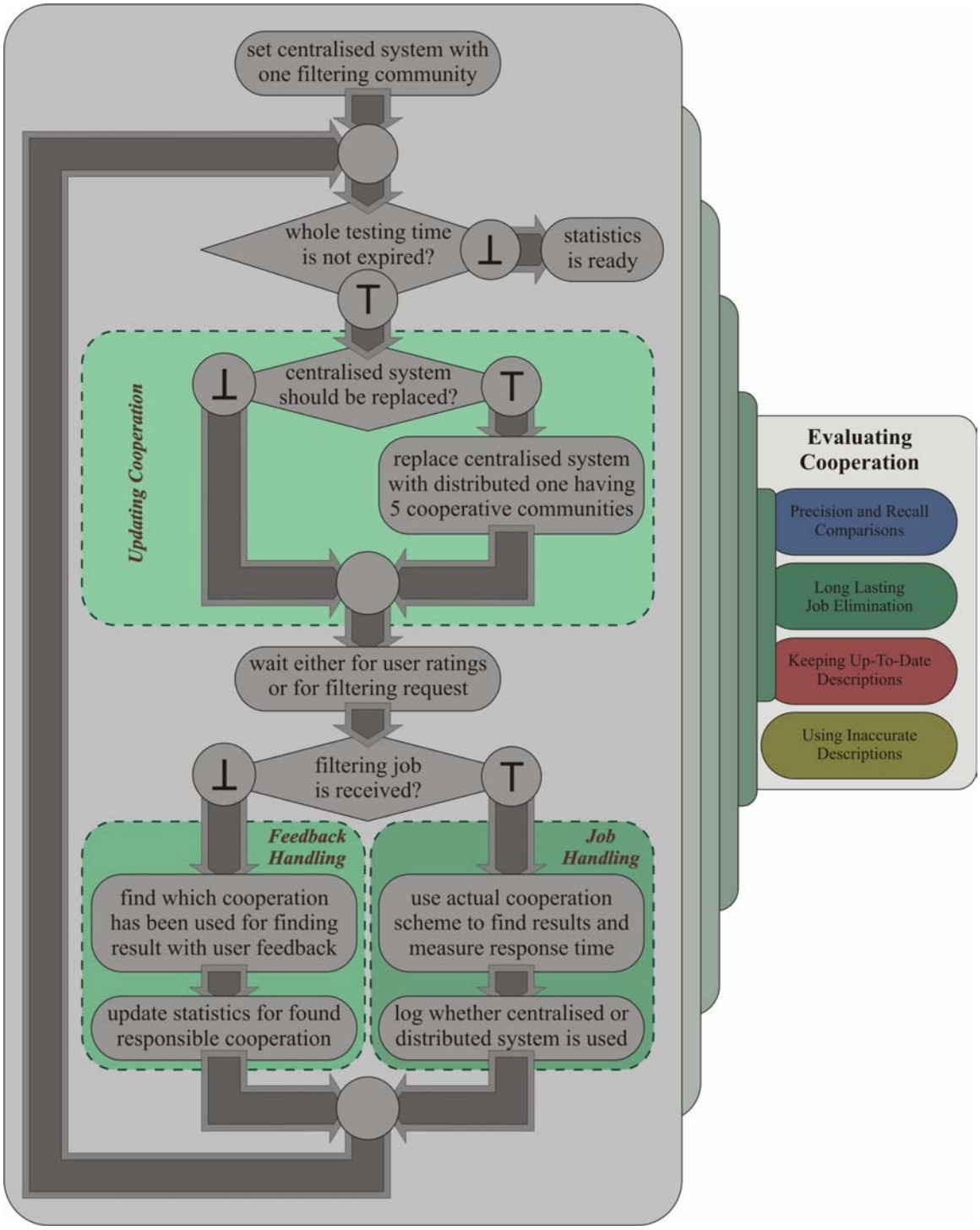

Figure 8.5: Tracking feedback and response time for centralised and distributed cooperation systems

Figure 8.5 gives a framework for performing necessary comparisons, being contained first in installing only one community and logging the response time and the got feedback, and afterwards replacing a centralised system with a distributed one, and again collecting statistics. On the one hand, updating cooperation block is responsible for deciding when a centralised system has been enough tested, and thus should be replaced. In the reality that happened on 18th of June 2004, after 37 user feedback values have been received from a beginning of evaluation. These obtained ratings and corresponding measured response time values are given on Figure 8.6 and Figure 8.7, respectively.

On the other hand, job and feedback handling blocks on Figure 8.5 are responsible for finding filtering results and updating statistics about the responsible cooperation strategy. The whole evaluation is finished after 37 new user ratings have been got when distributed cooperation system has been used. These new feedback and corresponding response time values are finally given on Figure 8.8 and Figure 8.9, respectively.

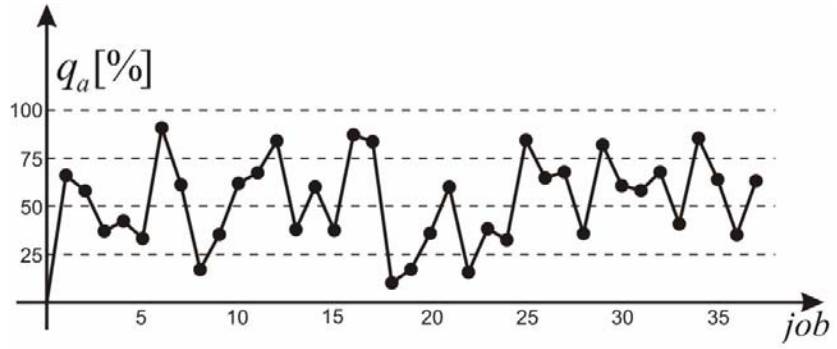

Figure 8.6: Received feedback without community cooperation

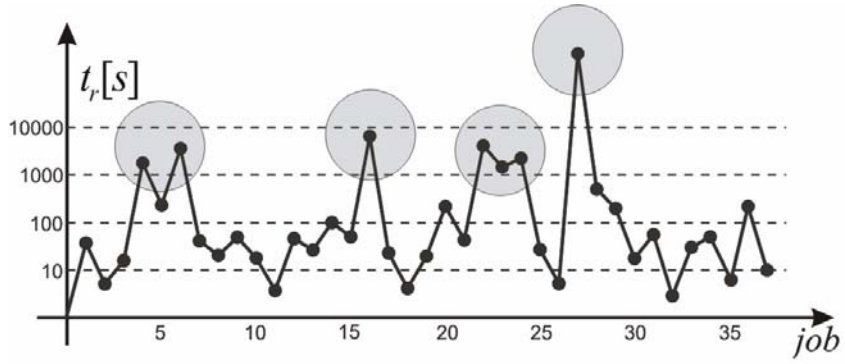

Figure 8.7: Response time without community cooperation

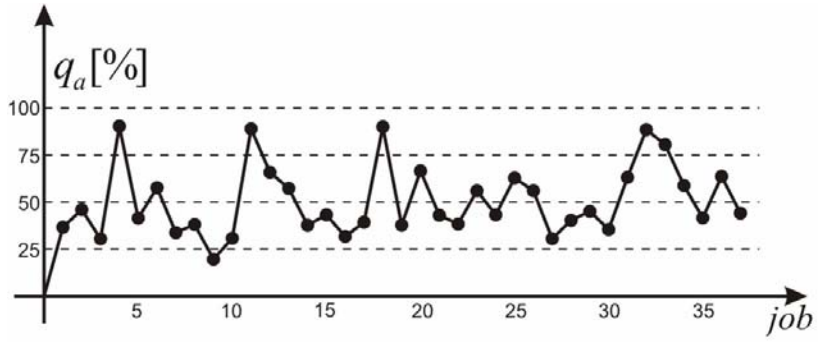

Figure 8.8: Received feedback with cooperation among multiple communities

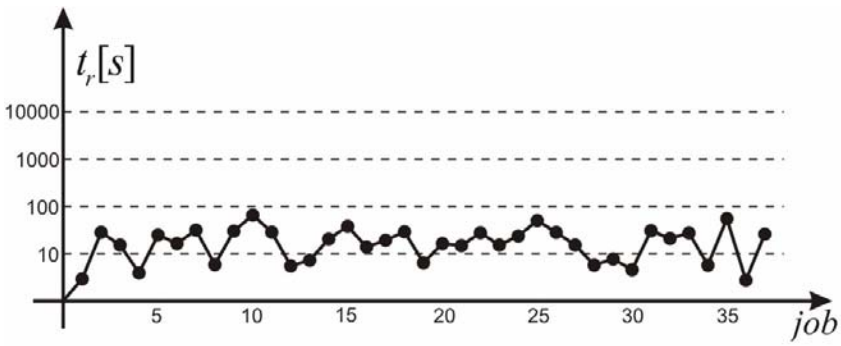

Figure 8.9: Response time with cooperation among multiple communities

Figure 8.6, Figure 8.7, Figure 8.8 and Figure 8.9 clearly show that while an integration of multiple cooperative communities does not significantly affect user relevance feedback (Figure 8.6 and Figure 8.8 show only the slight feedback value decrease that is hopefully within 3%), it successfully eliminates the long lasting filtering jobs (7 problematic long lasting jobs marked with circles on Figure 8.7, having a response time that is longer than 1000 seconds, do not occur anymore on Figure 8.9). Although Section 8.2.1 has pointed out that accuracy of results is reduced by a distributed system, an even bigger fluctuation in a quality has not been detected by users probably because they have been satisfied with the shorter waiting time. What is more important, by integrating the cooperative multiple communities, PIA system can provide filtering services on significantly larger document collections.

8.2.3. Keeping Descriptions Up-to-date

The added value of the cooperation mechanisms, being presented in Chapter 4, should be found in learning when the description of the particular community should be replaced by the new one, as being formally discussed in Section 4.3.1. It is very hard in the real world to find a collection of documents that is static, and consequently there is a need to model the dynamic nature of the content of every community. Possessing the accurate descriptions thus happens to have direct implications on the successfulness in finding the right communities, which will deliver results with the expected quality. The efficiency of updating community descriptions, towards realistically representing underlying document collections, is going to be consequently assessed by analysing the average values of both received user feedback values and filtering durations.

Six days before the 28th of January 2005, PIA I was working without any strategy for keeping community descriptions to be up-to-date. Between the 29th of January and 15th of February 2005, PIA II was using the strategy that updates all community descriptions in fixed intervals. Within 18 days long tests, PIA II was used with three different intervals, being 1 day, 6 hours and 30 minutes. After the 16th of February the update strategy with abilities of learning the period of asking for a new description is plugged in PIA III. The mentioned subsequent exchange of the different cooperation strategies is algorithmically presented on Figure 8.10, showing that job handling block logs for every delivered result which cooperation scheme was used, and that feedback handling block updates statistics about the responsible strategy based on the received user rating.

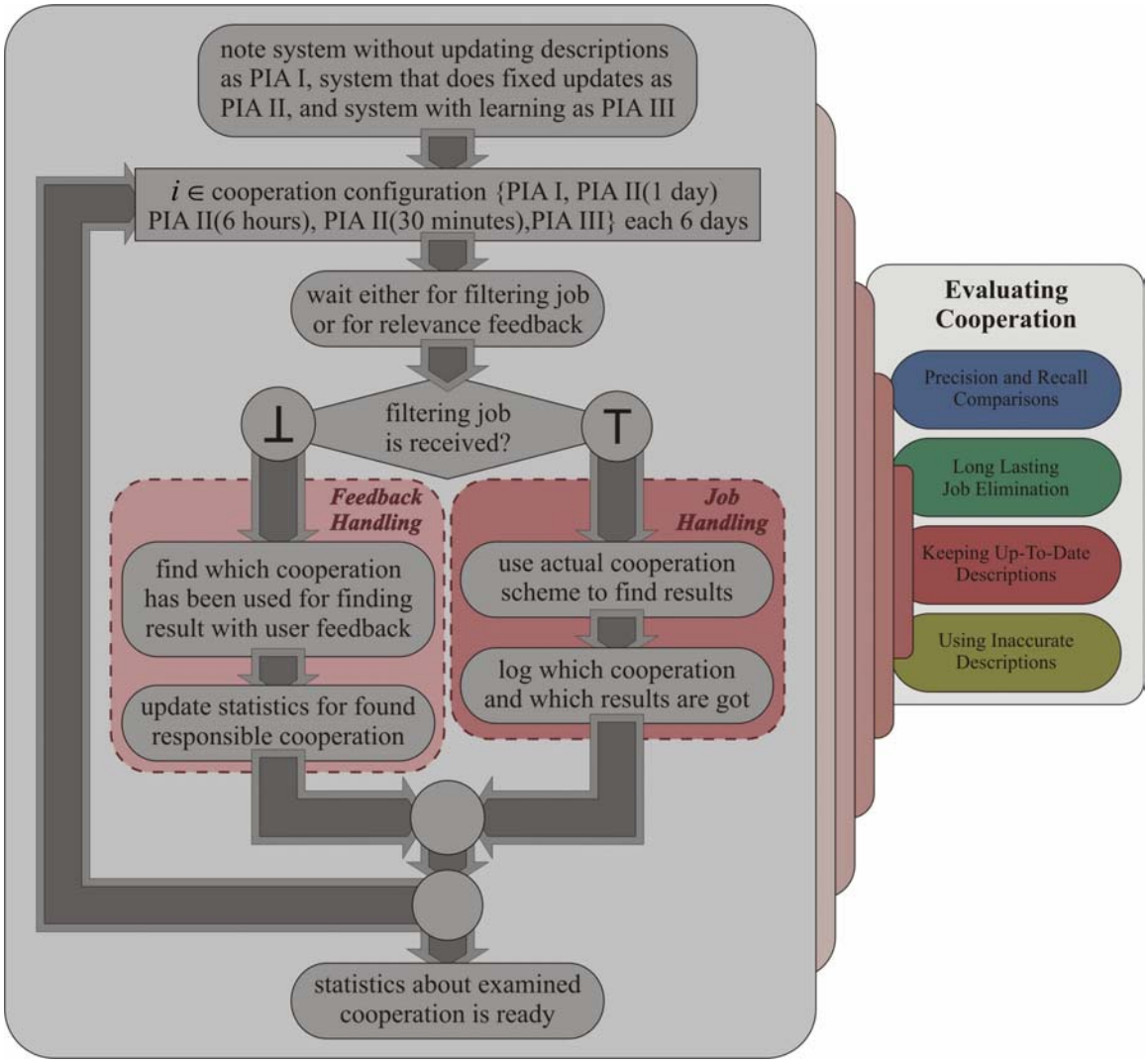

Figure 8.10: Evaluating different strategies for keeping community content descriptions up-to-date

The statistics in Table 8.4 clearly show that update strategy with learning abilities (PIA III) manages to significantly enlarge the user satisfaction (for more than 10% with respect to PIA I) without significantly increasing the average duration of the filtering (average filtering duration for PIA III is for less than 4s larger than for PIA I). As far as PIA II system is concerned, by decreasing the period of asking for the new community description the average duration of the filtering increases, because more often the new descriptions have to be retrieved. By having more accurate descriptions of communities, better results can be delivered, and average received used feedback also grows. But in the case where PIA II has checking period of 30 minutes, in spite of delivering very accurate results, the user satisfaction decreases (PIA II that checks each 30 minutes for the new descriptions has produced lower user satisfaction than PIA II that asks for descriptions each 6 hours). Users are obviously not ready to wait unreasonably long even in the case of getting very accurate results.

Table 8.4: Average filtering durations and received user feedback values for the tested different update strategies.

	Average filtering duration [s]	Average received user feedback [%]
PIA I	47.53	58.21
PIA II (1 day)	50.24	65.18
PIA II (6 hours)	69.88	67.25
PIA II (30 minutes)	81.21	66.71
PIA III	51.08	69.01

8.2.4. Working with Inaccurate Descriptions

Achieving the optimal trade-off of having always as good community descriptions as possible together with keeping the reasonable costs of updating them has stimulated both adapting t_C values, presented in Section 4.3.1, and reducing e_C values under the control of γ_d, described in Section 4.3.2. While the adaptation of t_C, fulfilled by expression (4.1), should eliminate the unnecessary retrievals of not novel enough community descriptions, the reduction of e_C in (4.3) is performed in order to enable the usage of not 100% up-to-date descriptions. The usefulness of both t_C adaptation, as well as e_C reduction is going to be assessed by tracking the t_C values and a number of new terms n_t in the community descriptions.

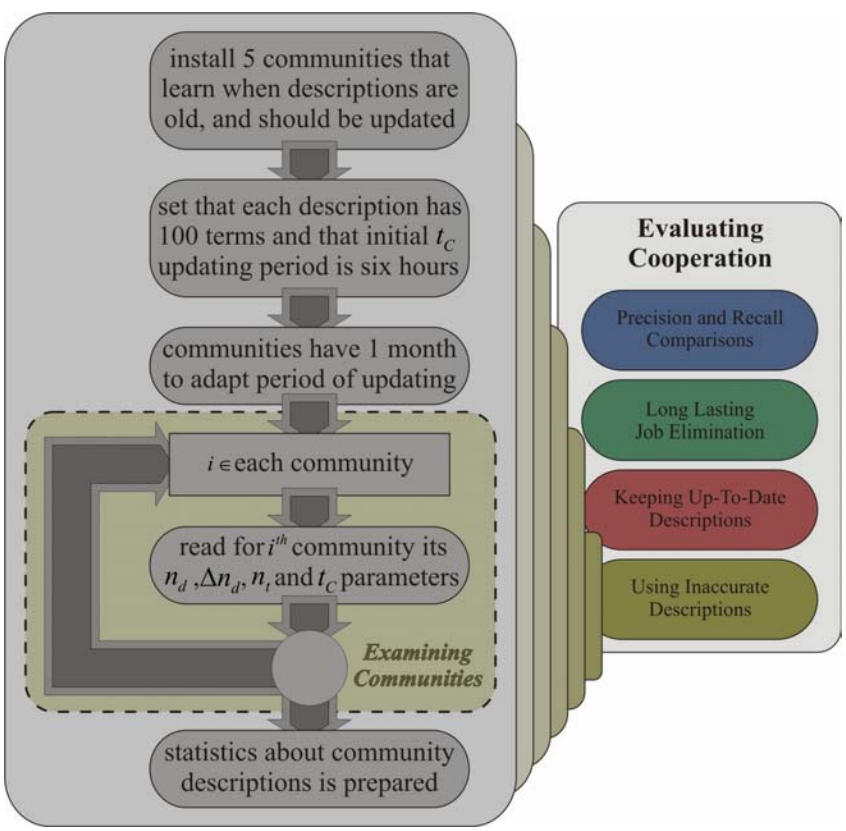

Figure 8.11: Experiment for testing abilities of communities to adapt period of updating descriptions

Starting from 18th of July 2004, PIA has five cooperative filtering communities [10], each being bound to one of the following domains: agent technology, telecommunication, renewable energy, sport news and political news. On 16th of February 2005, a component for updating community descriptions was plugged in PIA system with the initial t_C value of 6 hours for all communities. As being algorithmically presented on Figure 8.11, a size of each community description is also initially set to 100 terms, being the most important terms for the particular community. After one month of waiting, examining communities block on Figure 8.11 checks what has been learnt mostly by retrieving the new value for updating period t_C, and by determining the number of new terms n_t in the descriptions. The complete statistics is afterwards given in Table 8.5, containing the average t_C and n_t values that correspond to each community, as well as a number of documents n_d in each domain at the end of experiment, together with the number of documents Δn_d that have been inserted within this one month long experiment.

The encouraging observation is that system has managed to determine that sport and political news domains have much larger dynamic than the other domains (t_C values for sport and political news are much smaller than for other domains even though initially the same t_C value has been set to all used domains). Neither the number of documents in the underlying domain nor the number of newly inserted documents has the influence on t_C, being obvious when t_C for renewable energy domain is compared with other t_C values. Even though the most insertions are noted in the renewable energy domain (32 thousands of new articles are inserted, being almost half of the inserted articles in all domains), t_C for renewable energy domain has much higher value than for news domains. Obviously, articles in news domains are much more distinctive, resulting in the greater changes of community descriptions and consequently smaller t_C values.

Table 8.5: Average t_C and n_t values one month after the integration of a component for updating community descriptions

Domain	n_d	Δn_d	t_C [hours]	n_t
Agent technology	158.763	6.873	11.3	8.90
Telecommunication	122.008	5.873	17.9	9.12
Renewable energy	227.763	32.084	10.5	10.25
Sport news	106.342	11.765	2.3	12.39
Political news	134.982	12.432	1.7	11.91

The average number of new terms in each community description is tracked by reason of estimating the justification of e_C reduction for not up-to-date community descriptions through γ_d, done by expression (4.3). An average number of new terms n_t in community descriptions is for all domains around 10% regarding to the total size of the description. A difference among domains practically does not exist thanks to a successful adaptation of t_C values, and these new terms in descriptions point out that always descriptions are more or less inaccurate. Consequently, it sounds reasonable to reduce the estimation e_C

for the community whose description is just going to be replaced at most for 10%. This is achieved by simply setting 0.1 as γ_d value, being the value that has been used in all PIA experiments, and that has been obtained by using Figure 4.8. The future experiments will be focused on a better examination of γ_d parameter in a situation of having well formed t_C values.

8.3. Evaluating Exploration Approach

The expected strength of a multi-domain applicable exploration strategy, introduced in Chapter 5 as a sophisticated solution for an intelligent query processing, should be mostly found in its applicability in highly dimensional and sparse request-attribute space. As the quality of a performed exploration directly depends on the ability of a combined distance function, being given by expression (5.4), to find usable requests, one control experiment in Section 8.3.1 will be thus used to evaluate its successfulness, and to determine optimal weights that control the influence of Euclidean distance and Jaccard index. Section 8.3.2 afterwards gives the major evaluation, being performed in order to compare the presented exploration approach to other strategies that are also capable to somehow cope with the highly dimensional spaces. Since a subjective opinion about the strengths and weaknesses of each approach always plays the significant role, several examples, which illustrate the expanding of some typical requests, will be at the end given in Section 8.3.3.

8.3.1. Combined Distance Function Evaluation

To enable an efficient comparison of filtering requests, which are defined in the highly dimensional space of attributes that have their unique both values and weights, Euclidean distance function and Jaccard index are combined. On the one hand, Euclidean distance is good for estimating the closeness of the overlapping between two requests mostly based on the encapsulated values of attributes. On the other hand, Jaccard index is excellent in utilising the weights of attributes for finding requests with really wide overlapping. The combined distance function, being introduced by expression (5.4), aims to exploit these complementary strengths, and to outperform every separate solution. As the contributions of Euclidean distance and Jaccard index are controlled by the tuning parameters δ_E and δ_J in (5.4), the remainder of this sub-section is going to deeply describe one performed experiment, where the performances of this combination are evaluated by assigning the different values to δ_E and δ_J.

The evaluation experiment is algorithmically given on Figure 8.12, showing that really the first step is the manual construction of S_A and S_S sets of requests. These two sets are generated in the attribute space, having 467 unique words, where the smaller S_A set with only 10 requests plays a role of actual requests for which the usable requests are searched for in the larger S_S with 100 requests. Needed requests are simply generated by arbitrary selecting between 3 and 21 words to form the base of each request, and by independently assigning both value and weight to every chosen word in each request. On the one hand, larger requests are not created since less than 1% of real user requests from the two year long PIA usage have actually more than 21 words. On the other hand, filtering requests,

being shorter than 3 words, are not very useful for the explanation purposes by reason of not having enough background data, which can be potentially utilised for the expansion of the actual request.

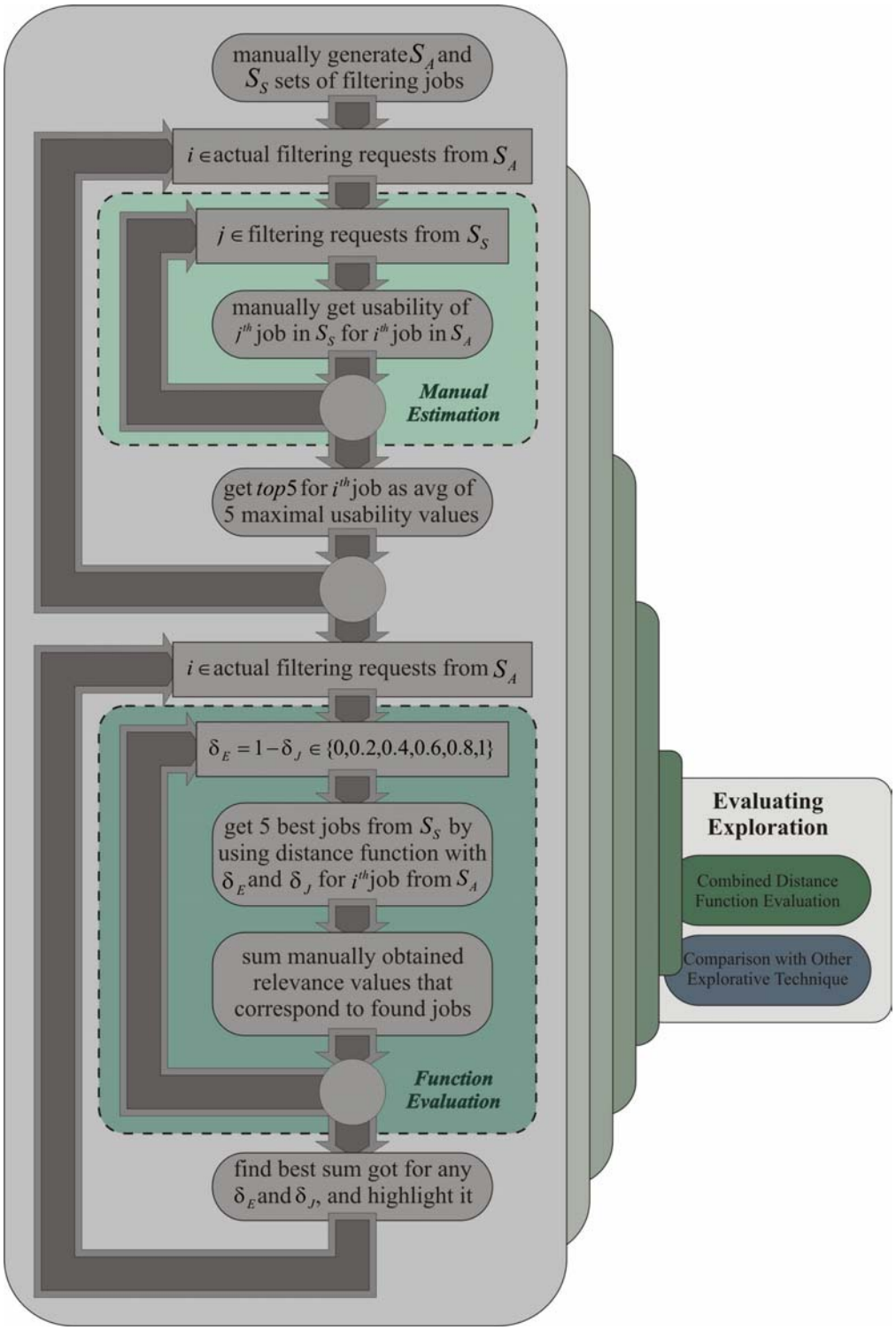

Figure 8.12: Comparing combined distance functions with different δ_E and δ_J weights

The most laborious activity, being marked on Figure 8.12 as a manual estimation, is to determine by hand the exploration usefulness of every request from larger S_S set to each request from S_A set of actual requests. As the result of these estimations, to every request from S_S set, ten real numbers from $[0,1]$ are assigned, each illustrating its usefulness for the expansion of a corresponding request from S_A set. An intuition behind the estimation of the usefulness is to assign the number that is closer to 1 when the requests have several overlapping words, which do not have so distinct values, and at the same time play the important roles in corresponding requests by reason of not having insignificant weights. The fulfilled manual estimation finally provides the needed data to find for every request from S_A the so-called $top5$ values, being the average sum of 5 largest estimations, which correspond to a particular S_A request.

What is more important, the manually determined usefulness can simulate the output of a perfect distance function in a given domain for the purpose of finding usable requests. In the function evaluation block on Figure 8.12, this ideal similarity solution can be then utilised for estimating a goodness of the realised combined distance, having the different values of δ_E and δ_J tuning parameters. Tested values for δ_E and δ_J are complementary chosen, where δ_E takes values from $\{0,0.2,0.4,0.6,0.8,1\}$ and $\delta_J = 1 - \delta_E$. Each distance function, obtained for the given δ_E and δ_J, is afterwards used for finding 5 best requests from S_S, having the largest d_C values, for every request from S_A. The manually given scores, corresponding to found requests, are finally averaged and given in Table 8.6. As an useful guideline for a comparison, average scores for top five S_S requests, which have the largest manually assigned similarity values for each request from S_A, and which have been already computed after the manual estimation has been performed, are given in the column $top5$.

Table 8.6: The average scores obtained by combined distance function having different δ_E and δ_J values, as well as the best possible manually achievable score, for any request from S_A set.

S_A request	$\delta_E = 1 - \delta_J$						$top5$
	0	**0.2**	**0.4**	**0.6**	**0.8**	**1**	
1	0.68	0.83	**0.91**	0.85	0.78	0.72	0.93
2	0.57	0.75	0.83	**0.84**	0.68	0.60	0.87
3	0.63	0.87	**0.88**	0.79	0.74	0.53	0.91
4	0.68	0.71	**0.91**	0.88	0.79	0.64	0.95
5	0.70	0.78	**0.84**	**0.84**	0.81	0.75	0.90
6	0.54	0.69	0.72	**0.73**	0.64	0.59	0.85
7	0.62	0.71	**0.78**	0.77	0.59	0.58	0.91
8	0.75	**0.83**	0.82	0.78	0.75	0.69	0.90
9	0.61	0.65	0.72	**0.79**	0.74	0.61	0.83
10	0.70	0.82	**0.84**	0.81	0.77	0.64	0.93

The results from Table 8.6 are encouraging mostly by demonstrating that practically any combination, where both δ_E and δ_J are not zero, outperforms solutions where either solely Euclidean distance or Jaccard index is utilised. The performed experiment clearly shows that while the most successful weighting is almost regularly $\delta_E = 1 - \delta_J = 0.4$, the best results are sometimes obtained for $\delta_E = 1 - \delta_J = 0.6$, and also for the 8th S_A request even with $\delta_E = 1 - \delta_J = 0.2$. Since little better results are obtained for $\delta_J > \delta_E$, the role of Jaccard index is slightly more important in a realised combination scheme. On Figure 8.13, the best combination, being marked by a bold font in Table 8.6, is separately compared to the ideal solution, given in a column $top5$, for every S_A request. While the bar with a red filling corresponds to the value of the best combination, the blue bar shows for how much $top5$ is better than the best $\delta_E = 1 - \delta_J$ result. By analysing the size of blue bars, it might be noticed that the achieved accuracy is always within 10%, with the exception of 6th and 7th S_A requests, where the obtained results are for around 15% below the top results.

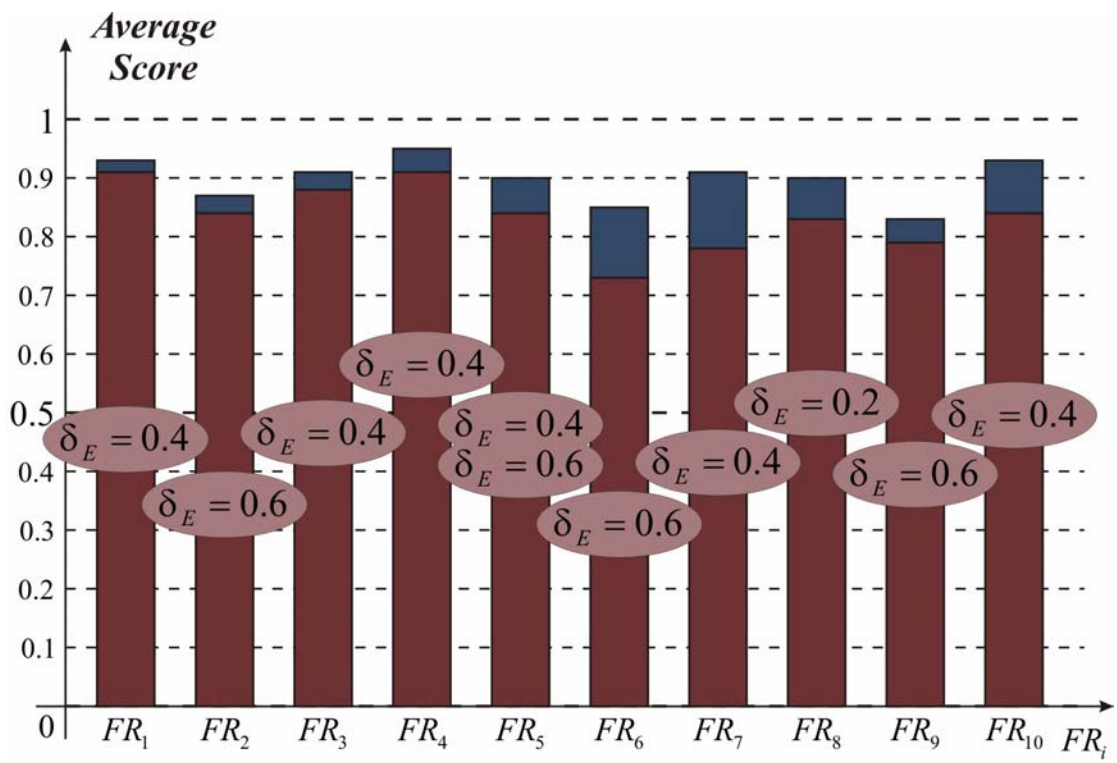

Figure 8.13: Comparison between the manually obtained average score (blue bar filling) with the best possible result that is obtained for different δ_E values (red bar filling together with δ_E winner)

Even though the very small sample of requests (S_A and S_S have 10 and 100 requests, respectively) is created in not very high dimensional space (only 467 distinct words take part in created requests), the results from Figure 8.13 and especially from Table 8.6 can at least motivate the combination between different distance functions. Since the best values for weights δ_E and δ_J cannot be universally set to always bring the best possible results,

the more sophisticated combination of distance functions can maybe take care of request properties, such as number of words, standard deviation and mean value of weights, and so on. The idea of learning during the runtime, based on the received relevance feedback, which distance function is more applicable for which types of requests, seems to be quite appealing, at least because something similar is already achieved in a complex domain of information retrieval and a selection of filtering strategies [17]. Similarly as each filtering strategy has its unique abilities towards various types of requests [15], the contribution of different distance functions can be controlled based on the properties of the actual request. The possible ways, how this analogy can be ambitiously utilised to design one intelligent self-improving distance function, is going to be certainly one of topics for future author's investigations.

8.3.2. Comparison with Other Explorative Techniques

The Section 5.2, where background knowledge for the exploration has been presented, has pointed out that an idea of performing the intelligent query processing is not new, and that many other techniques are also trying to cope with that challenge. In this sub-section, the exploration approach, presented in Chapter 5, is going to be compared with the pure collaborative filtering, as well as with mutation and crossover mechanisms, in the domain of received user feedback values. The skyline technique is not realised by reason of not being able to effectively handle requests in highly dimensional attribute space, whereas a so-called similarity versus diversity approach is not interesting because of assuming that users are always capable to perfectly articulate their thoughts about their current interests and therefore not providing the adequate support for imprecisely formulated information needs.

As a test environment for performing necessary user studies, PIA system is going to be utilised mostly by reason of currently maintaining around 500 thousand distinct words, as well as having processed almost 25 thousand various filtering requests. Since the filtering framework, which is behind PIA system, is one collection of various filtering strategies, ranging from different indexing techniques [58][59] to the wide palette of self organising map algorithms [46][88][134][148][218], somebody might speculate that obtained user feedback values are not solely the consequence of deployed exploration approaches. A deployment of the single filtering strategy is unfortunately not manageable by reason of making completely useless comprehensive coordination [13][17] mechanisms, which PIA exploits in order to achieve an optimal trade-off between the response time and quality of results for any received request in the current runtime situation. The found workaround, minimising the influence of deployed various filtering strategies, is summarised as set-up block on Figure 8.14, which formally gives the used testing scenario. As the ability of a particular community to deliver solid results critically depends on the installed filtering strategies, communities are made equally good by owning the same searching algorithms. The potential advantage of a particular community of solely having the access to the right documents is also cancelled by letting that everybody uses a whole collection. These two set-up steps have therefore made as equivalent as possible different communities, which are still necessary by reason of balancing a load, and making a whole system more robust to single failures.

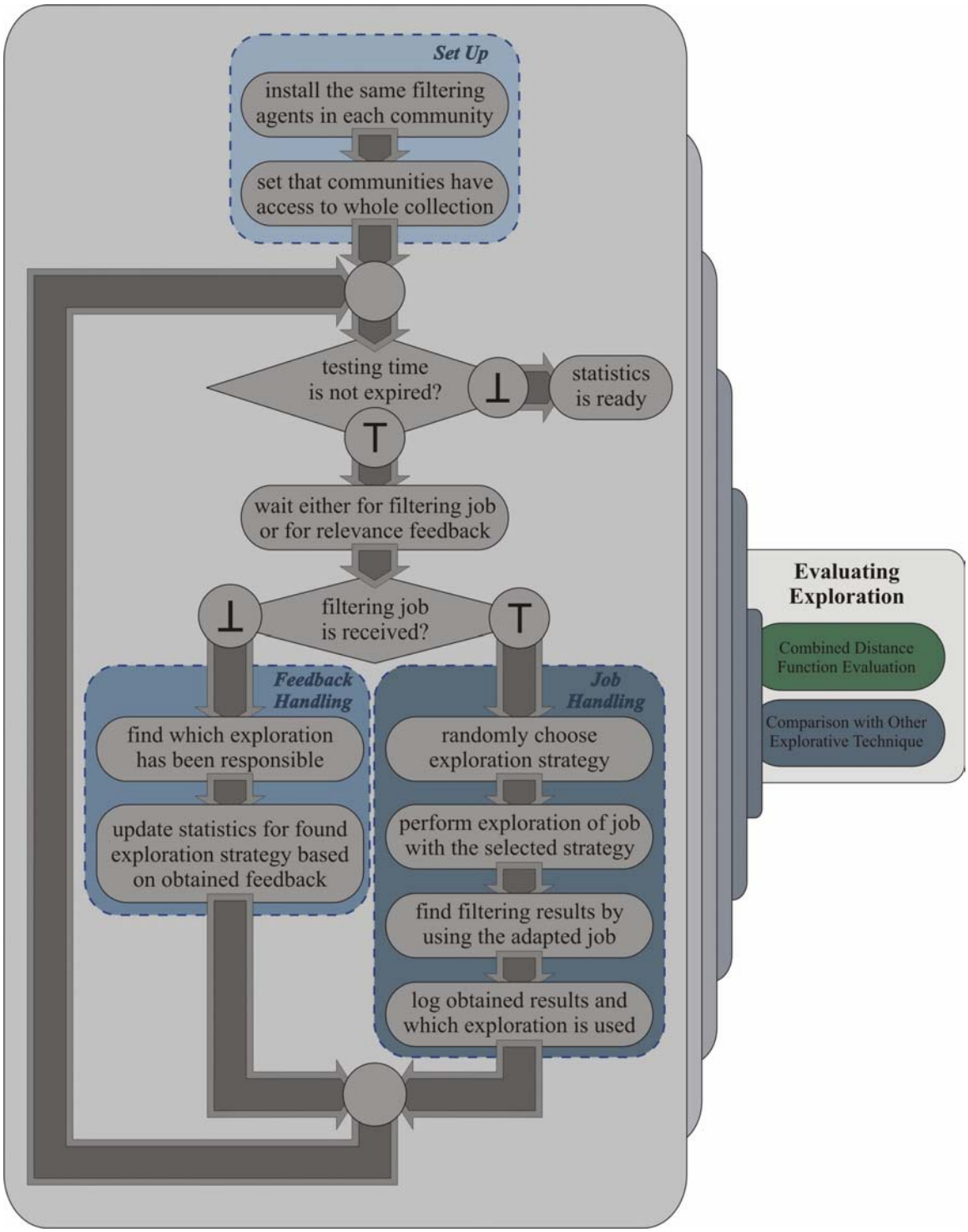

Figure 8.14: Procedure used for evaluating different exploration strategies

Since the performances of the exploration approach from this thesis depend on really many configuration parameters, such as δ_E, δ_J from expression (5.4), β_E from (5.6) and β_P from (5.7), performed analysis has been simplified by setting $\delta_E = 0.4$ and $\delta_J = 0.6$, being values for which the combined distance function has shown the best performances

in the small control experiment from the previous sub-section. The possible values for the exploration β_E and prediction β_P rate are afterwards independently chosen from a small set $\{0.25, 0.5, 0.75\}$, which is necessary in order not to risk too much the patience of users. These different strategies, obtained for any combination of β_E and β_P, are, together with collaborative filtering and mutation & crossover approaches, tested in the way that no one is continually used in the given longer period of time by reason of potentially producing the high users' dissatisfaction when a bad strategy is deployed over the several days. The optimal testing strategy is found in arbitrary selecting the exploration approach for every received request, being done in the job handling block on Figure 8.14, and afterwards in keeping track on the obtained user feedback values, which update the statistics about the responsible strategy in feedback handling block. Although the users' feedback values are highly subjective, where the same user can also inconsistently provide diverse ratings for the same article in the different points in time [29], the obtained results can provide some evidence by reason of being based on more than 8 thousands filtering requests, which are processed within the three month long testing period.

Before discussing the experimental results, presented in Table 8.7, the short summary about the collaborative filtering and mutation & crossover evolution techniques will be given. On the one hand, the realised collaborative filtering uses user ratings to first find like-minded users, and then to recommend articles being liked by them. This strategy completely ignores the actual user's preferences, and consequently does not manage to control the level of exploration. On the other hand, the mutation & crossover evolution technique is first measuring the fitness of the different requests based on the obtained feedback values, and then using the most fitted requests to exchange the words with the actual request. While such an exchange of words is actually crossover, both values and weights of words can be mutated by being randomly changed for a small amount. This technique also ignores the content of the actual request while searching for requests that will be used for exploration, and practically leads the exploration towards the most successful requests, which have produced results that have got high user ratings.

Table 8.7: The distribution of obtained user feedback values for nine different configurations of the realised exploratory approach, collaborative filtering (CF) and mutation & crossover (MC) strategies

β_E	β_P	Obtained user feedback values				
		Very Good	**Good**	**Bad**	**Very bad**	**Off-topic**
0.25	**0.25**	40 (33.61%)	42 (35.29%)	15 (12.6%)	10 (8.4%)	12 (10.08%)
	0.5	47 (38.21%)	35 (28.45%)	19 (15.45%)	7 (5.69%)	15 (12.19%)
	0.75	31 (27.67%)	24 (21.43%)	21 (18.75%)	13 (11.6%)	23 (20.53%)
0.5	**0.25**	38 (37.25%)	25 (24.51%)	18 (17.65%)	11 (10.78%)	10 (9.80%)
	0.5	41 (35.96%)	30 (26.31%)	17 (14.91%)	13 (11.4%)	13 (11.40%)
	0.75	20 (18.86%)	23 (21.69%)	23 (21.7%)	15 (14.15%)	25 (23.58%)
0.75	**0.25**	19 (17.11%)	30 (27.03%)	24 (21.62%)	16 (14.41%)	22 (19.82%)
	0.5	23 (21.49%)	20 (18.69%)	21 (19.63%)	13 (12.15%)	30 (28.04%)
	0.75	28 (20.58%)	28 (20.59%)	28 (20.59%)	16 (11.76%)	36 (26.47%)
CF		23 (13.37%)	17 (9.88%)	32 (18.6%)	22 (12.79%)	78 (45.35%)
MC		18 (13.13%)	15 (10.95%)	29 (21.17%)	19 (13.87%)	56 (40.88%)

Both absolute number of ratings and percent distribution for each strategy are given in Table 8.7, where for example 40 (33.61%) in row $\beta_E = \beta_P = 0.25$ and column "very good", should be understood that in total 40 "very good" user ratings are received in the configuration $\beta_E = \beta_P = 0.25$, and that these "very good" ratings make 33.61% of all the ratings obtained when $\beta_E = \beta_P = 0.25$. While the absolute values are excellent to identify for example that collaborative filtering has got the largest number of ratings (on average each strategy has received 121.72 ratings, whereas the collaborative one has obtained 172), percentage values are more objective for speculating about a successfulness of each strategy. Even though for example both the configuration with $\beta_E = 0.75$ and $\beta_P = 0.5$, as well as collaborative filtering, have got 23 "very good" ratings, these 23 "very good" ratings make 21.49% of all ratings obtained for $\beta_E = 0.75$ and $\beta_P = 0.5$, and only 13.37% of all collaborative ratings.

In addition to usually available four rating scale (very good, good, bad, very bad), the opportunity of specifying that a particular result is off-topic, is intentionally provided not only to minimise the potential dissatisfaction of users, who have only known that various strategies have been tested and that sometimes completely unexpected or off-topic results can appear, but also to illustrate the cornerstone advantage of the presented approach for performing the intelligent exploration without making many fatal mistakes. While both collaborative filtering and mutation & crossover technique result in really many off-topic results (78 collaborative and 56 mutation & crossover results are rated as off-topic), the intelligent exploration approach from this thesis produces on average not more than 25 off-topic results for $\beta_E \leq 0.5$ and around 29 for $\beta_E = 0.75$.

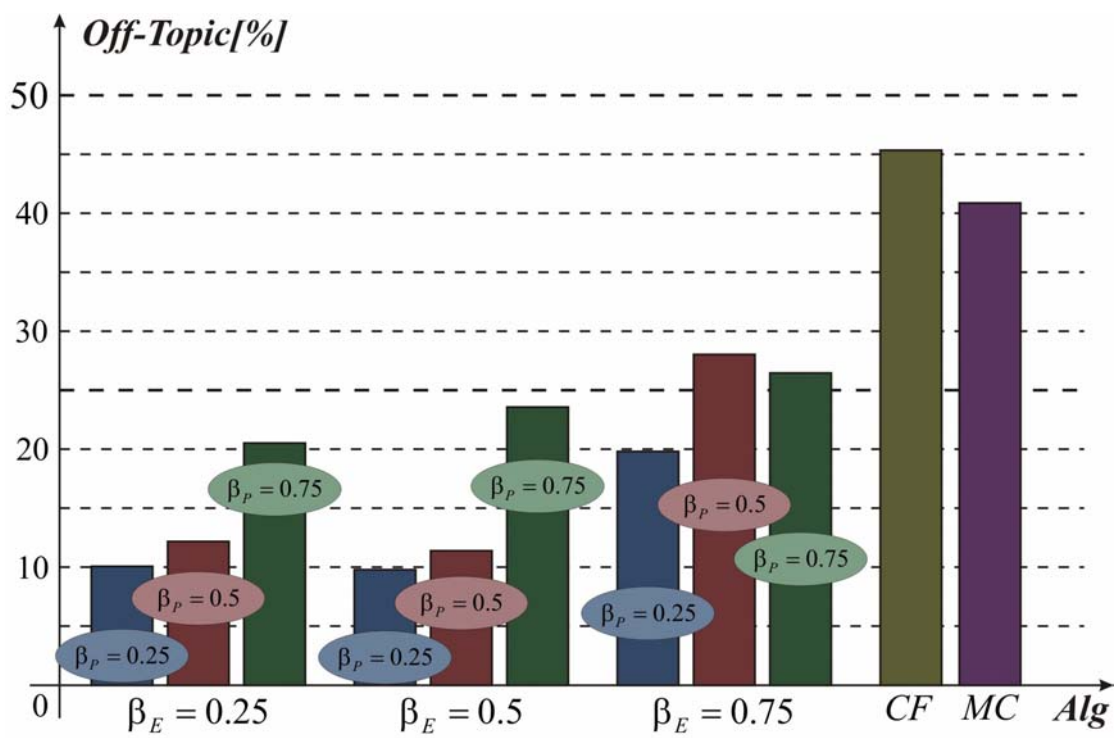

Figure 8.15: Comparison of the amount of off-topic results that are obtained by different algorithms

The maybe more expressive comparison of relative values of the got off-topic ratings for different exploration strategies is presented on Figure 8.15, where the exploration rate β_E is explicitly given on a horizontal axis, and the prediction rate β_P is drawn inside the bar corresponding to a particular configuration. A relative or percent point of view shows that while 45.35% of all collaborative and 40.88% of all mutation & crossover results are rated as the off-topic, the proposed exploration algorithm gives on average only 14.6% for $\beta_E \leq 0.5$ and 24.8% for $\beta_E = 0.75$ off-topic results. An observed number of off-topic results is even smaller when both β_E and β_P are equal or below 0.5, in which case the average number of off-topic results is only 12.5, or expressed relatively only 10.87% of results, being obtained in these four configurations, are rated as off-topic. It can be finally seen, in Table 8.7 and Figure 8.15, that even the worst observed cases, when $\beta_E = 0.75$ and $\beta_P \geq 0.5$, are at least around 15% better than CF and MC approaches.

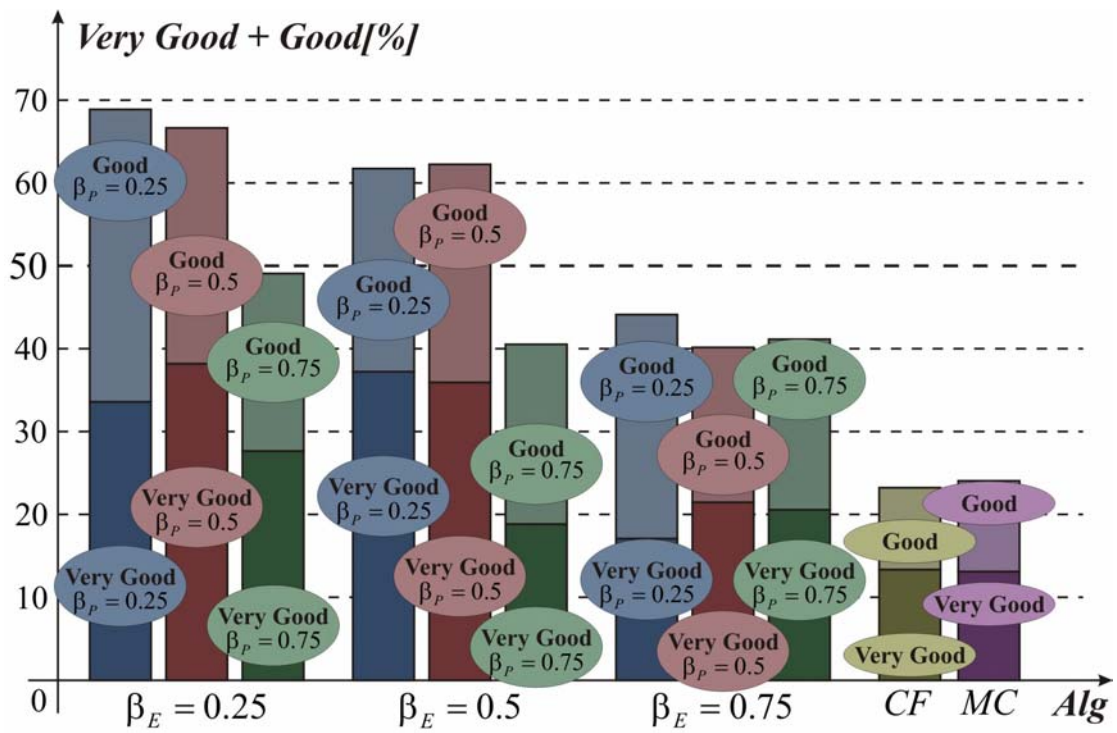

Figure 8.16: Comparisons between very good (darker bar filling) and good (lighter bar filling) ratings that are obtained for different algorithms

As far as the comparison of other obtained feedback values is concerned, more "very good" and "good" received ratings, are especially got when both $\beta_E \leq 0.5$ and $\beta_P \leq 0.5$ than for collaborative filtering and mutation & crossover strategy. These "very good" and "good" ratings are both separately and cumulatively presented on Figure 8.16, helping in getting as clear picture as possible about a superior behaviour of the exploration scheme from this thesis. Both Table 8.7 and Figure 8.16 shows that while 36.26% of "very good" and 28.64% of "good" relevance ratings are received on average in four configurations with $\beta_E, \beta_P \leq 0.5$, only 13.37% or 9.88% of all collaborative, and 13.13% or 10.95% of all mutation & crossover results, are rated as either "very good" or "good". Even the

much higher exploration rate, having $\beta_E = \beta_P = 0.75$, has managed to clearly outperform collaborative filtering and mutation & crossover techniques, by getting 20.59% of "very good" and the same amount of "good" ratings.

The obtained results for various β_E and β_P values also motivate a decision to try to vary these parameters, and to intelligently control the level of exploration. The results of the performed study show that the best user satisfaction, regarding both the sent ratings and the number of off-topic results, is generally achieved for both $0.25 \le \beta_E \le 0.5$ and $0.25 \le \beta_P \le 0.5$. The further step can be in trying to find the optimal values for these parameters not globally for all users, but separately for each user, being reasonable since various users probably prefer to get results with different levels of exploration.

Table 8.8: List of actual requests that are adapted always by adding five most promising words, being found in the similar request neighbourhood of the size three. The added words are both in the neighbourhood of the found similar old requests, as well as in adapted actual request, typed in bold.

Actual request	Three most similar old requests	Adapted actual request
{agent}	{agent, **negotiation**, **reactivity**, coordination, agent-builder}	{agent, **negotiation**, **reactivity**, **gaia** **neighbourhood** **telecommunication** }
	{agent, **telecommunication**, **neighbourhood**, network, isdn}	
	{agent, **gaia**, methodology}	
{agent, negotiation}	{agent, negotiation, **auction**, **coordination**, bdi, simulation}	{agent, negotiation, **auction**, **blackboard**, **coordination**, **bdi**, **agency**}
	{negotiation, **auction**, **blackboard**, **coordination**, jade}	
	{agent, **auction**, **agency**, jess}	
{agent, reactivity}	{agent, reactivity, scalability, **communication**, **protocol**}	{agent, reactivity, **communication**, **protocol**, **ontology** **actor**, **intelligence** }
	{agent, **communication**, **actor**, **intelligence**, message, ams}	
	{reactivity, event, **ontology**}	
{agent, neighbourhood}	{agent, neighbourhood, **tfidf**, **similarity**, **query**, feedback}	{agent, **similarity** neighbourhood, **tfidf**, **filter**, **fitness**, **query**}
	{neighbourhood, **tfidf**, **filter**, stemming, metric}	
	{agent, **filter**, **fitness**, **query**}	
{agent, gaia}	{agent, gaia, **methodology**, **aose**, technique, process}	{agent, gaia, **methodology**, **message**, **aose** **requirements**, **environment**}
	{gaia, **aose**, **requirements**, **environment**, task-oriented}	
	{agent, **message**, **methodology**}	
{agent, telecommunication}	{agent, telecommunication, **tcp**, isdn, wap, udp, fttp, wlan}	{agent, **network** telecommunication, **tcp**, **seamless**, **bluetooth**, **isdn**}
	{agent, **isdn**, **network**}	
	{telecommunication, **seamless**, **bluetooth**, **network**, motorola}	

8.3.3. Exploration Examples

The quantitative evaluations, given in the previous two sub-sections, can be strengthen by giving few examples, which demonstrate the ability of presented exploration approach to both find usable old requests and afterwards extract important words from them. Since PIA was mainly only internally used inside the author's laboratory, most of the processed filtering requests are from the agent technology domain. The following examples, being given in Table 8.8, will consequently demonstrate the adaptation of requests, which one, for the agent domain naïve, user can post to the PIA system. Giving exploration examples with requests from other domains will not be fair, because there is not a needed previous experience that is encapsulated in similar requests. As soon as the PIA system establishes its competence in other domains, it will become possible to provide such examples.

In order to make as illustrative and as simple examples as possible, only underlying words from the selected requests are given, and their values and weights are omitted. In the case where the found similar requests are analysed, instead of basing the extraction of the representative words on β_P, which depends on not presented weights, always five best words are extracted and included into the actual request. The most common request, being posted to PIA 107 times, contains only one word "agent", and currently, for one such request, PIA will behave as it is presented in the first part of the Table 8.8. Obviously, the exploration is limited because word "agent" occurs in many different contexts. The second part of the Table 8.8 illustrates the adaptation of requests, which are little better formulated. It is noticeable that requests are always expanded with related words, being the most important property of the presented intelligent exploration.

8.4. Evaluating Coordination Approach

The greatest contribution of three coordination schemes, being introduced in Chapter 6 as resource aware, job aware, and self-healing, should be found in describing the different capabilities of filtering strategies by the adequate fitness values, and then in utilising that knowledge to achieve the optimal usage of system resources and a solid user satisfaction. Since the exploitation of resource related fitness values, being introduced in Section 6.1.1, naturally requires the methodology for estimating the load of relevant system resources, the evaluation algorithms, which measure the execution time, and whish are proposed in Section 6.3.1, will be consequently tested in Section 8.4.1. The applicability of a realised coordination scheme to be applied inside large communities will be afterwards checked in Section 8.4.2. In a controlled environment Section 8.4.3 then demonstrates the abilities of resource aware coordination scheme to autonomously improve itself during a runtime by adapting its resource related fitness values. The last three sub-sections finally utilise a real PIA system to examine both the abilities to eliminate the long lasting filtering jobs (Section 8.4.4), the influences of taking care of the properties of jobs while selecting the filtering strategies (Section 8.4.5), and the capabilities to autonomously recover after the failures (Section 8.4.6).

8.4.1. Estimating CPU and Database Server Load

The load estimation technique, being proposed in Section 6.3.1, is based on giving the task to either CPU of database server, and afterwards measuring the time, needed to fulfil

it. A natural expectation is that more loaded CPU or database server will result in longer execution, as well as that the needed time will be proportional to the average load during processing. The following experiments thus have a goal to artificially load either CPU or database, and afterwards to check whether a produced load can be deduced by measuring the execution time.

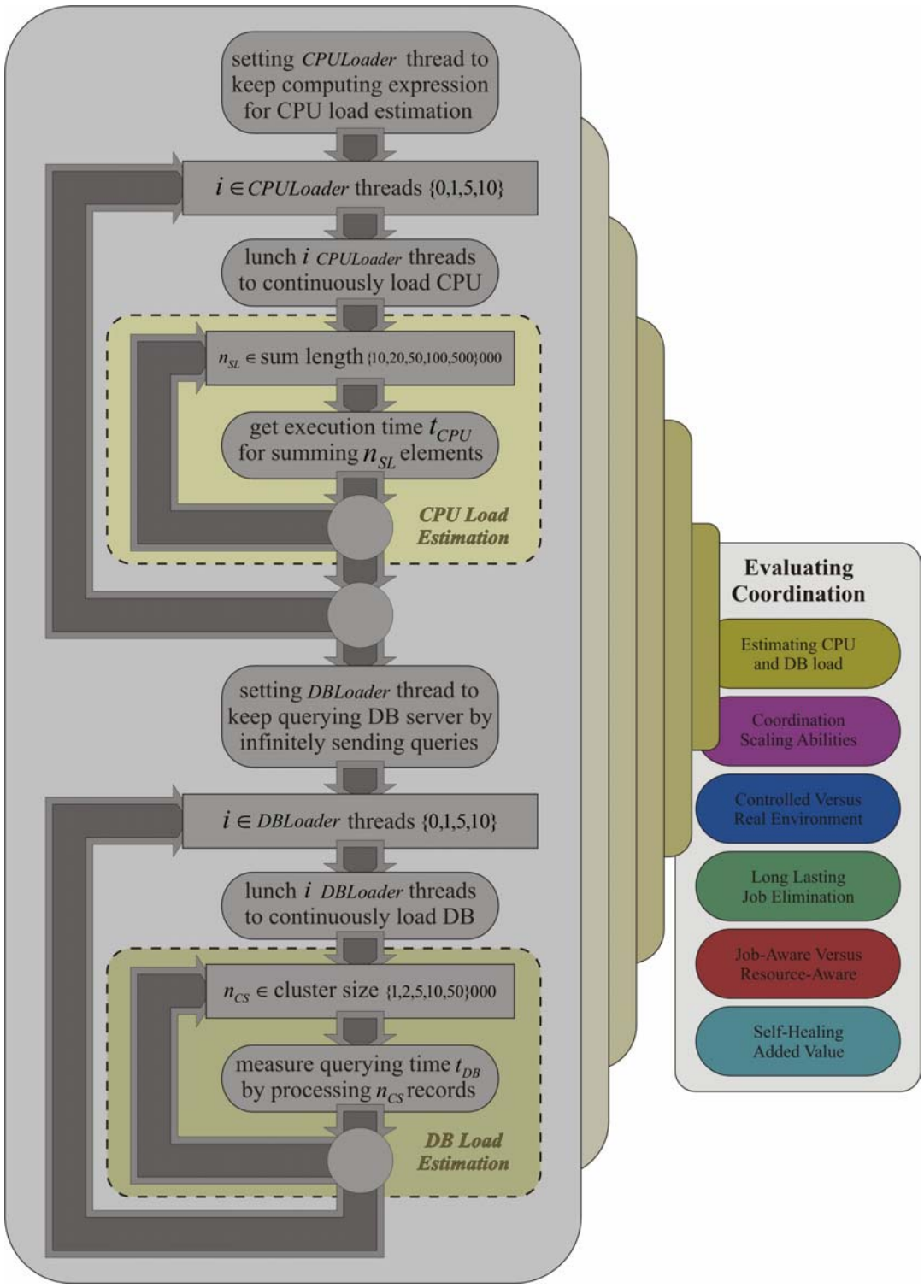

Figure 8.17: Estimation of CPU and DB server load by simulating load through extra threads

A testing algorithm is formally summarised on Figure 8.17, where the separate blocks for CPU and DB load estimation are noted. The CPU is artificially loaded by utilising so-called *CPULoader* threads, which infinitely keep computing an expression $\sum_{i=1}^{n_{SL}} f(i)$ for CPU load estimation. A level of the actual CPU load is therefore precisely controlled by lunching the given number of such *CPULoader* threads. The execution time t_{CPU} can be finally measured for a different summation length n_{SL} in order to additionally check how the sensitivity of estimation depends on n_{SL}.

An experimentally found the dependence of t_{CPU} from n_{SL}, when the different number of *CPULoader* threads have been lunched, is presented both in Table 8.9 and Figure 8.18. A measured t_{CPU}, corresponding to a duration when as CPU one Pentium IV on 2.5 GHz is utilised, shows that a grow of either $n_{CPULoaders}$ or n_{SL} has as a consequence the increase of t_{CPU}. This observation further enables that a measuring of t_{CPU} with a known n_{SL} can be used for finding the actual load of CPU.

Table 8.9: t_{CPU} dependence from a summation length n_{SL}

$n_{CPULoaders}$	n_{SL} [in thousands]	t_{CPU} [s]
0	10	0.016
	20	0.031
	50	0.078
	100	0.125
	500	0.657
1	10	0.078
	20	0.109
	50	0.281
	100	0.563
	500	2.766
5	10	0.187
	20	0.328
	50	0.844
	100	1.688
	500	8.765
10	10	0.328
	20	0.625
	50	1.547
	100	3.078
	500	15.578

The database server is examined in DB load estimation block on Figure 8.17. Its load is similarly simulated by defining the so-called *DBLoader* threads, which send all over again SQL queries to a database server. By lunching a given number of such *DBLoader* threads, a DB load can be controlled, and therefore makes feasible experiments where the response time t_{DB} is measured for different cluster size n_{CS}.

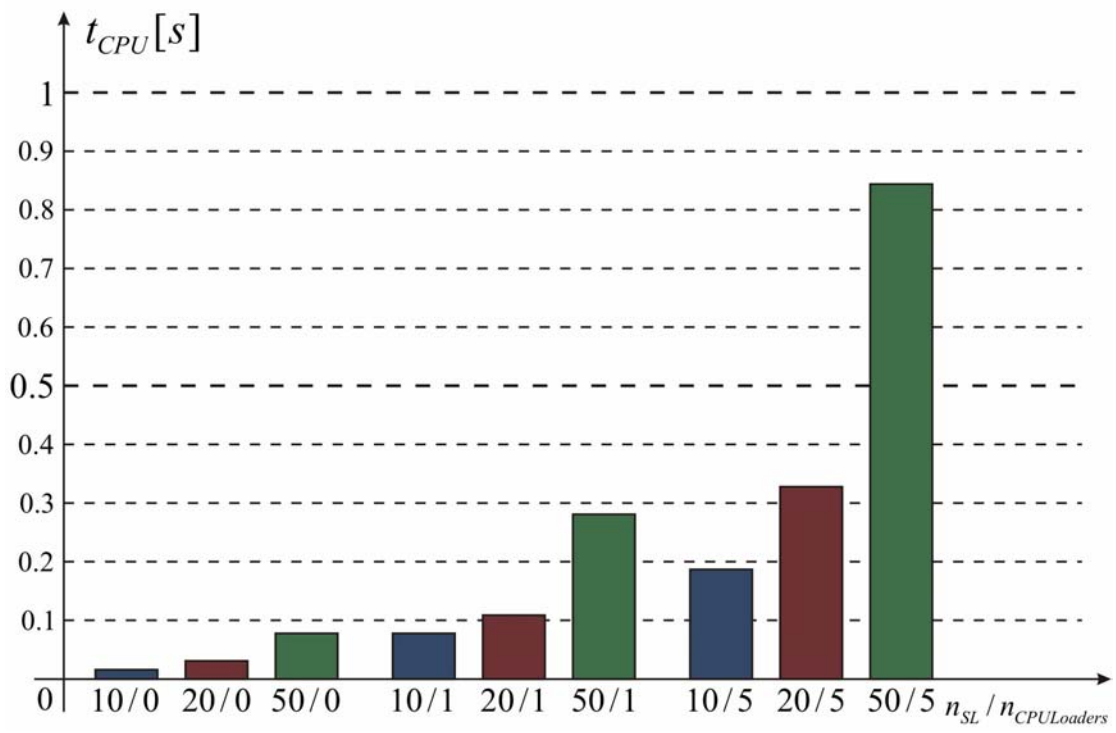

Figure 8.18: Dependence of t_{CPU} for $n_{SL}/n_{CPULoaders}$ from Table 8.9 that can be visually presented

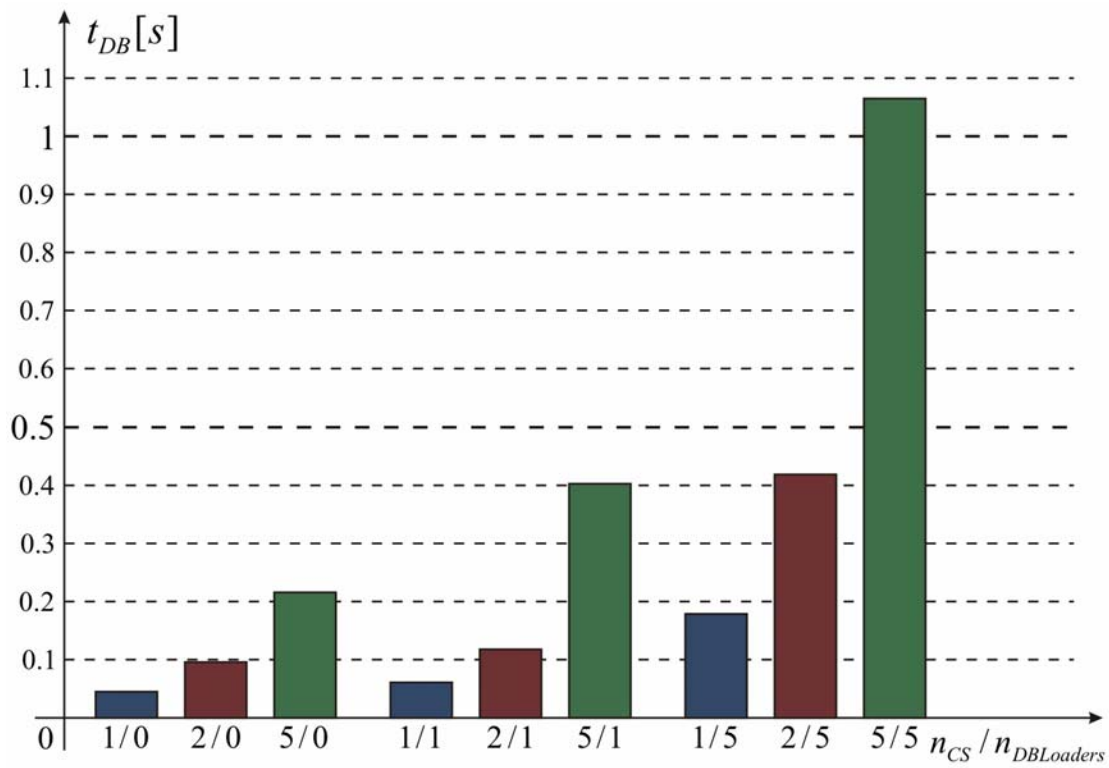

Figure 8.19: Dependence of t_{DB} for $n_{CS}/n_{DBLoaders}$ from Table 8.10 that can be visually presented

In the case where as the database server one Pentium IV on 1.6 GHz is utilised, the experimentally found dependence of t_{DB} from n_{CS} and the number of *DBLoader* threads $n_{DBLoaders}$ is given in Table 8.10 and on Figure 8.19. On the one hand, it might be noticed that for the fixed sensitivity, being defined by the cluster size n_{CS}, the t_{DB} proportionally grows with $n_{DBLoaders}$. On the other hand, larger cluster size n_{CS} also results in longer t_{DB}. A final conclusion is that the measuring of t_{DB} can provide reliable information about an actual DB load.

Table 8.10: t_{DB} dependence from a cluster size n_{CS}

$n_{DBLoaders}$	n_{CS} [in thousands]	t_{DB} [s]
0	1	0.045
	2	0.096
	5	0.216
	10	0.499
	50	2.206
1	1	0.061
	2	0.118
	5	0.403
	10	0.803
	50	4.134
5	1	0.179
	2	0.419
	5	1.065
	10	2.070
	50	10.955
10	1	0.292
	2	0.620
	5	1.587
	10	3.523
	50	16.244

Performed experiments with measuring t_{CPU} and t_{DB} have shown, as being expected, that a higher load of CPU or database results in longer t_{CPU} or t_{DB}. A proposed technique for load estimation can be thus used not only when the inbuilt system functions for giving load values are not available, but also when the average load over the particular period of time is needed.

8.4.2. Coordination Scaling Abilities

The resource aware coordination algorithm from Section 6.3 is designed to address the challenges that arise when the size of a filtering community is scaled to the hundreds or thousands of members. That is mainly achieved by establishing evaluation and selection coordination mechanisms in a way that the most time consuming activities should be the ones that compute CPU and DB server load, and that at the same time do not depend on

the number of filtering agents. In order to check the last scaling assumption, communities with a different number of filtering agents (n_{FA}) are created and to each 100 jobs are sent. As being algorithmically given on Figure 8.20, it is afterwards measured how much time is needed to compute CPU load ($t_{\omega_{CPU}}$), to estimate DB server load ($t_{\omega_{DB}}$), and to perform all other coordination activities (t_C), being the evaluation of total fitness $F_t^{(i)}$ values and the computation of selection probabilities $P^{(i)}$. Figure 8.20 shows that while neither $t_{\omega_{CPU}}$ nor $t_{\omega_{DB}}$ depends on n_{FA}, t_C should grow with n_{FA} as there is, inside a strategy selection block whose duration is represented by t_C, a loop that iterates over all installed filtering agents.

The obtained average values for $t_{\omega_{CPU}}$, $t_{\omega_{DB}}$ and t_C are presented in Table 8.11. While $t_{\omega_{CPU}}$ and $t_{\omega_{DB}}$ slightly fluctuate regarding n_{FA}, by reason of being measured in real time conditions, t_C heavily depends on n_{FA}, mostly because of being concerned with the evaluation of a total fitness F_t, which has to be done for each filtering agent. Furthermore, t_C additionally assumes the application of a proportional selection, which also depends on n_{FA}. The last column t_C[%] in Table 8.11 gives the percentage contribution of t_C in the overall time that is necessary for performing coordination. As it can be seen, for the communities, having less than 200 agents, that t_C contribution is below 1%, which is a quite encouraging observation. Even in a case of very large community with 5000 agents, t_C contribution is only around 10%, being also acceptable.

Table 8.11: Durations of coordination activities for different size n_{FA} of communities. t_C is duration of selecting a strategy after computing ω_{CPU}, ω_{DB} and ω_m (t_C includes neither $t_{\omega_{CPU}}$ nor $t_{\omega_{DB}}$)

n_{FA}	$t_{\omega_{CPU}}$ [ms]	$t_{\omega_{DB}}$ [ms]	t_C [ms]	t_C [%]
10	32.97	83.75	0.15	0.12834774
20	32.74	107.27	0.46	0.32747206
50	33.39	87.39	0.6	0.49431537
100	32.79	81.26	0.66	0.57536396
200	32.29	82.85	0.83	0.71570234
500	32.53	81.23	1.24	1.07826087
1000	32.49	84.7	1.87	1.57063665
2000	32.58	87.74	3.43	2.77171717
5000	33.03	82.61	14.2	10.9365373
10000	32.81	83.74	33.91	22.5375515
20000	32.35	82.82	61.24	34.7145853

The results presented in Table 8.11 prove that in Chapter 6 described resource aware coordination mechanisms are scaling well, and that they can be applied even inside very large filtering communities, having thousands of agents.

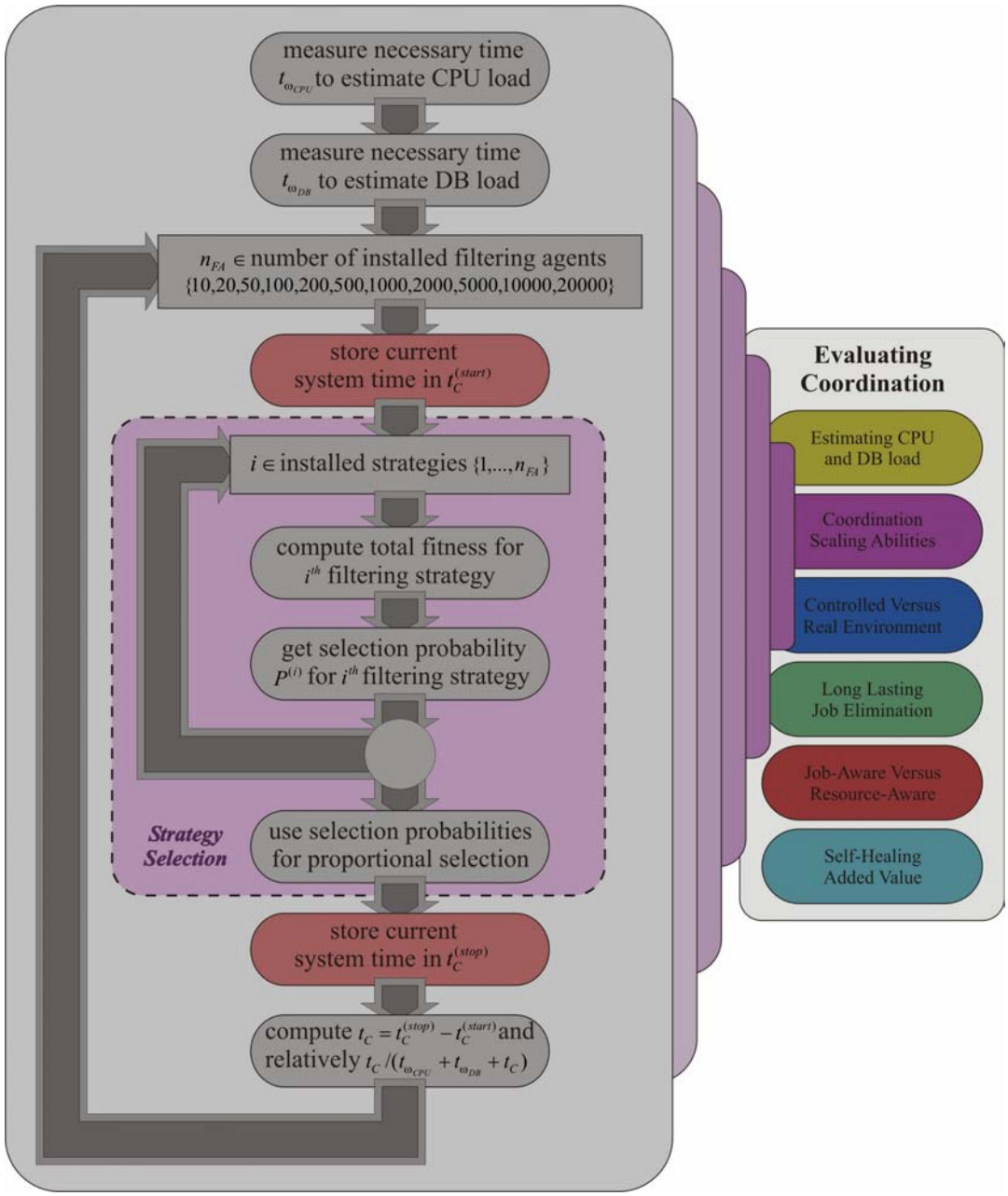

Figure 8.20: Estimating the scalability of coordination mechanisms regarding the installed agents

8.4.3. Controlled Versus Real Environment

In a real environment, load values ω_x, $x \in \{CPU, DB, m\}$, introduced in Section 6.3.1, are always more or less unpredictable. On the one hand, it is reasonable to use current ω_x values for the evaluation of resource strategy fitness value by expression (6.4), because in the selection process only the idea about the actual suitability of every strategy is needed. On the other hand, when fulfilling the adaptation of resource related fitness values, being described in Section 6.3.4, maybe it is not fair to punish strategy for its bad performance,

because in real conditions ω_x values are not fixed during the processing of a filtering job. It is possible that ω_x values have been changed in a way that is particularly inappropriate for a chosen filtering strategy, and a final consequence is the probably long response time t_r. Without any doubt, for that big t_r a particular strategy should not be so responsible, because it has not declared that it is good in that changed environment. The influence of ω_x unpredictability to the adaptation of resource related fitness values is therefore a main motivation for performing an experiment, being algorithmically presented on Figure 8.21, and having the following five steps:

[Step I] A real PIA system log is used to get statistics about ω_x values. It is found that $\omega_{CPU} \in [0.01,4.7]$, $\omega_{DB} \in [0.04,20.5]$ and $\omega_m \in [0.02,0.9]$.

[Step II] A strategy FS is asked to process 100 jobs in the controlled environment, which can change its properties only because of a FS activity. For each job, different ω_x are used as an environmental description by applying controlled threads, which will not suddenly be finished during a FS activity. A response time t_r is measured, and as output 100 pairs $(\omega_{CPU}, \omega_{DB}, \omega_m, t_r)$, describing FS behaviour in different situations, are got.

[Step III] A multiple polynomial regression on response time t_r with basis functions $\{1, \omega_{CPU}, \omega_{CPU}^2, \omega_{DB}, \omega_{DB}^2, \omega_m, \omega_m^2\}$, performed by program package Mathematica [270], gives:

$$t_r(\omega_{CPU}, \omega_{DB}, \omega_m) = a^{(0)} + \sum_{x \in \{CPU, DB, m\}} \sum_{i=1}^{2} a_x^{(i)} \omega_x^i \tag{8.2}$$

where $a^{(0)} = 9.72$, $a_{CPU}^{(1)} = 139.42$, $a_{CPU}^{(2)} = 67.53$, $a_{DB}^{(1)} = 46.8$, $a_{DB}^{(2)} = 38.9$, $a_m^{(1)} = 87.31$, $a_m^{(2)} = 78.2$. Fitness values F_x, are introduced as $a_x^{(i)} = b_x^{(i)} e^{-\beta F_x}$, $i \in \{1,2\}$, which further for $\beta = 0.01$ and $F_x = 50$ gives $b_{CPU}^{(1)} = 232.3$, $b_{CPU}^{(2)} = 112.5$, $b_{DB}^{(1)} = 72.3$, $b_{DB}^{(2)} = 64.2$, $b_m^{(1)} = 144.1$ and $b_m^{(2)} = 129$. A larger F_x means a smaller $a_x^{(i)}$ and consequently implies a minor ω_x influence on t_r, which perfectly corresponds to F_x definitions in Section 6.1.1. The obtained t_r approximation with fitness values F_x is:

$$t_r(\omega_{CPU}, \omega_{DB}, \omega_m, F_{CPU}, F_{DB}, F_m) = a^{(0)} + \sum_{x \in \{CPU, DB, m\}} e^{-\beta F_x} \sum_{i=1}^{2} b_x^{(i)} \omega_x^i \tag{8.3}$$

[Step IV] By utilising the obtained results from the previous step, a perfect history log $(\omega_{CPU}, \omega_{DB}, \omega_m, F_{CPU}, F_{DB}, F_m, t_r)$ can be synthetically created for any number of filtering agents in the community and any number of processed jobs. A table with 2000 records is created in the way that for every of 20 randomly created filtering strategy descriptions (F_{CPU}, F_{DB}, F_m), exactly 100 random environmental descriptions $(\omega_{CPU}, \omega_{DB}, \omega_m)$ are also generated, and t_r is computed by using its polynomial approximation. In order to try to

reflect a real log as good as possible, ω_x belongs to intervals that are obtained in Step I and $F_x \in [0,100]$. The idea is to utilise the created table as an ideal knowledge base for expected fitness value $F_x^{(e)}$ computation.

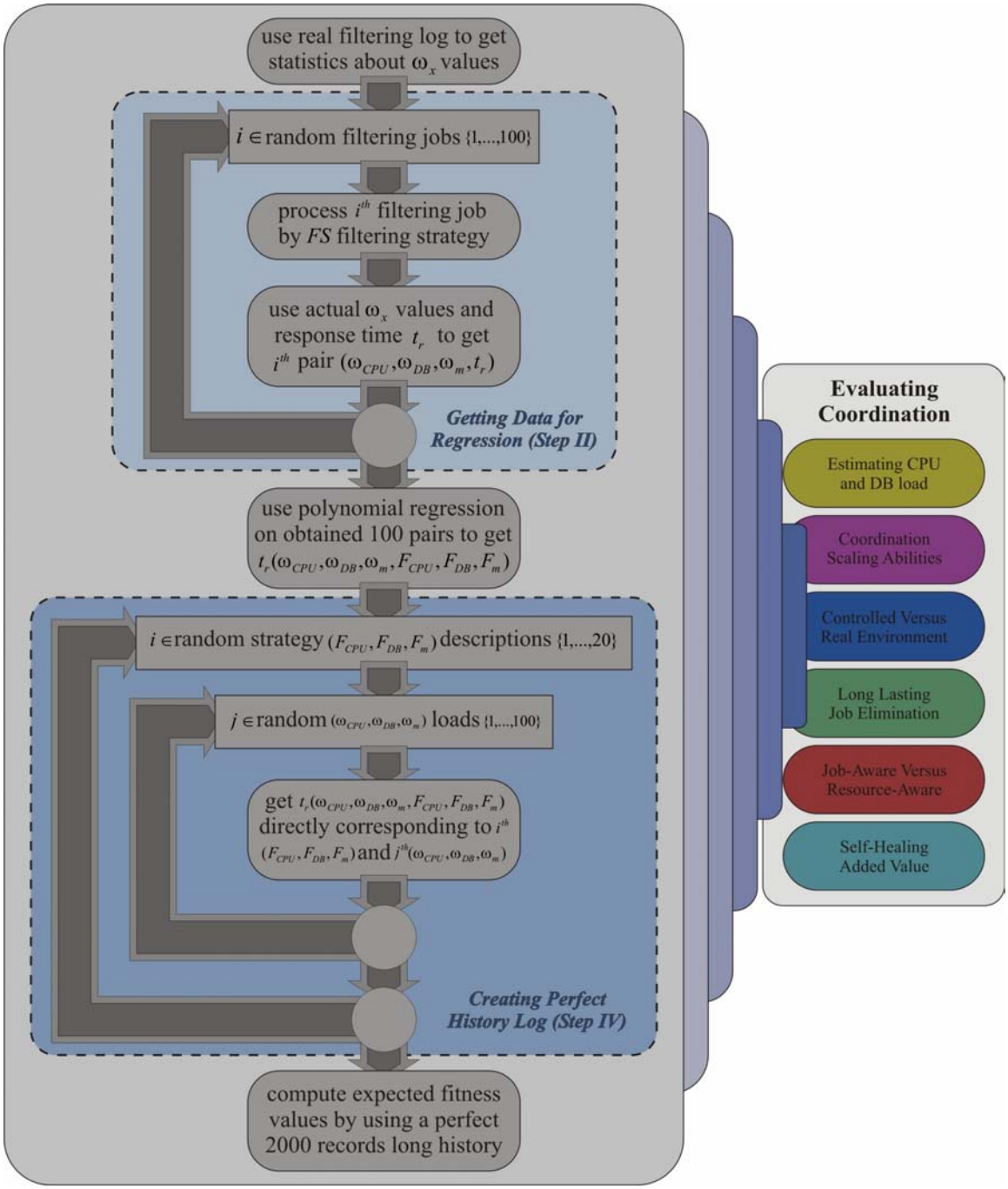

Figure 8.21: Five steps of the performed experiment for estimating the influence of the unpredictable changes of load ω_{CPU}, ω_{DB} and ω_m values when the resource related fitness values are adapted

[Step V] FS is used for processing 100 jobs in the real environment where ω_x can be unpredictably changed. Expected fitness values are computed by using the synthetically created log. Jobs are divided based on their durations t_r in four clusters C_i, and for each cluster standard deviation $\sigma_{F_x^{(e)}}$ is computed. The information about the created clusters, being the number of jobs in the given cluster, and the computed standard deviations, are finally presented in Table 8.12.

The adaptation of the actual fitness values can be successful only when computed $F_x^{(e)}$ values for a particular filtering strategy are not varying greatly. A standard deviations for C_1 and C_2, where 65% of all jobs belong, has reasonably small values, mostly because only short jobs are in C_1 and C_2. This actually means that a probability for unpredictable ω_x variations is low, and the adaptation process can work as it is expected. For longer jobs, probability that something unexpected will happen is much higher, which will result in larger $F_x^{(e)}$ deviations.

Table 8.12: Standard deviations for clusters with s records, created based on different job durations

i	t_r condition [s]	s	$\sigma_{F_{CPU}^{(e)}}$	$\sigma_{F_{DB}^{(e)}}$	$\sigma_{F_m^{(e)}}$
1	$t_r < 1$	27	2.78	3.02	2.15
2	$1 \le t_r < 10$	38	4.12	4.58	3.94
3	$10 \le t_r < 100$	15	7.98	9.34	8.28
4	$t_r \ge 100$	20	29.4	25.1	32.7

The performed experiment has shown that while for jobs with $t_r < 10s$ the adaptation can be based on in the evaluation computed ω_x values, for longer jobs the adaptation will require additional information about ω_x during job processing. The way how that can be achieved is going to be subject for the future investigations.

8.4.4. Long Lasting Filtering Job Elimination

A trial to escape long lasting filtering jobs, being the ones with the duration over 1000 seconds, and being usually the consequence of the unavailability of an important resource, has been a main motivation for the realisation of the resource aware coordination scheme, introduced in Section 6.3. Although these long lasting jobs will probably produce perfect results in the next few hours, to obtain nearly perfect results within few minutes or even faster is usually much more appropriate. As a final judgement, concerning such trade-off statements, is always given by the user, this sub-section gives comparisons between PIA systems with and without resource aware coordination mechanisms in relevance feedback and response time domains.

Before the 14th of January 2004, PIA I system was working without a resource aware coordination scheme. Between 15th of January and 9th of March 2004, PIA II was using coordination approach with only evaluation and selection mechanisms, being described in Sections 6.3.1 and 6.3.3, respectively. After 9th of March 2004, the adaptation component,

being used for learning the realistic resource related fitness values, and being presented in Section 6.3.4, is plugged in PIA III.

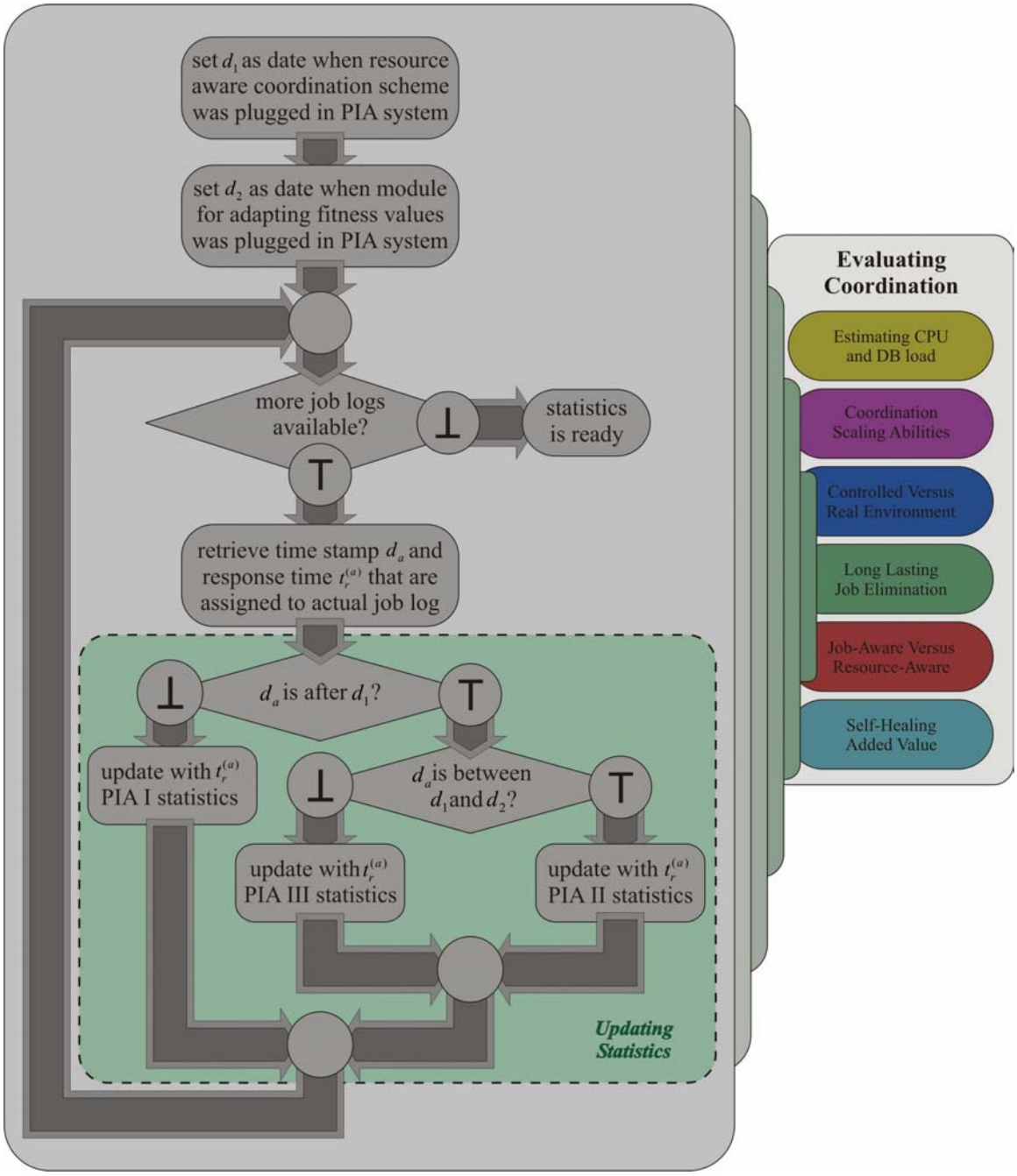

Figure 8.22: Evaluation algorithm used for getting statistics about long lasting jobs in different PIAs

The algorithm for obtaining statistics about the distribution of the durations of filtering jobs is based on mining the available logs, as being illustrated on Figure 8.22, where the mentioned 15th of January and 9th of March 2004 are marked as d_1 and d_2, respectively. Every single log, encapsulating both the duration $t_r^{(a)}$ of a corresponding job, as well as a

time stamp d_a when the particular job was processed, is analysed in the block, named as updating statistics on Figure 8.22. The got statistics about the durations of jobs is finally presented in Table 8.13.

The statistics data in Table 8.13 clearly show that resource aware coordination scheme successfully eliminates long lasting filtering jobs (6 problematic long lasting jobs having the response time that is longer than 1000 seconds, do not occur anymore in PIA II and PIA III). At the same time no noticeable difference in a received user feedback has been detected. A logical conclusion is that by optimising the usage of available resources by resource aware coordination mechanisms, which are also able to autonomously improve themselves during runtime, PIA can provide better filtering services to a significantly larger user community.

Table 8.13: Number of processed jobs by different PIA systems for which user feedback exists

t_r condition [s]	PIA I	PIA II	PIA III
$t_r < 10$	9	99	8743
$10 \le t_r < 100$	14	97	3482
$100 \le t_r < 1000$	3	2	21
$t_r \ge 1000$	6	-	-

8.4.5. Job-Aware Versus Pure Resource Aware Coordination

To exploit an assumption that each strategy also has very different capabilities towards processing the various types of filtering jobs, the job aware coordination scheme has been introduced in Section 6.4. Its major property is surely contained in making the weighted combination between resource F_r and job content F_c components while computing total fitness F_t value by expression (6.18). Since the weighting in (6.18) is fulfilled by α, the cornerstone of the performed experiments will be contained in varying α value, and in making comparisons in both user relevance feedback and response time domains.

From the 30th of August till 10th of September 2004, each two days the PIA inbuilt coordination mechanism was changed by incrementing α value for 0.25, starting from $\alpha = 0$. In order to make comparisons as fair as possible, only working days are taken into account, being done because of the minor PIA usage during weekend days by DAI Labor workers. Figure 8.23 formally presents the utilised evaluation methodology, being split in updating coordination, feedback handling and job handling blocks. On the one hand, the role of updating coordination part is to increment α value after the tested period of two working days is expired for a particular configuration. On the other hand, both feedback and job handling parts are waiting for either new filtering request or user explicit ratings. One should note that job handling part will consider only requests, being received during working days, as potential candidates, whose results might finally inspire a user to give a feedback that will be counted.

The obtained results are given in Table 8.14, where n_{fj} corresponds to the number of filtering jobs for which a feedback has been received, n_{lj} represents the number of long-

lasting jobs, having a duration of more than 1000 seconds, $avg(d_{fj})$ is the average duration of jobs with a feedback, and $avg(q_{fj})$ is the average received user feedback. The most important $avg(d_{fj})$ and $avg(q_{fj})$ are also graphically presented on Figure 8.24.

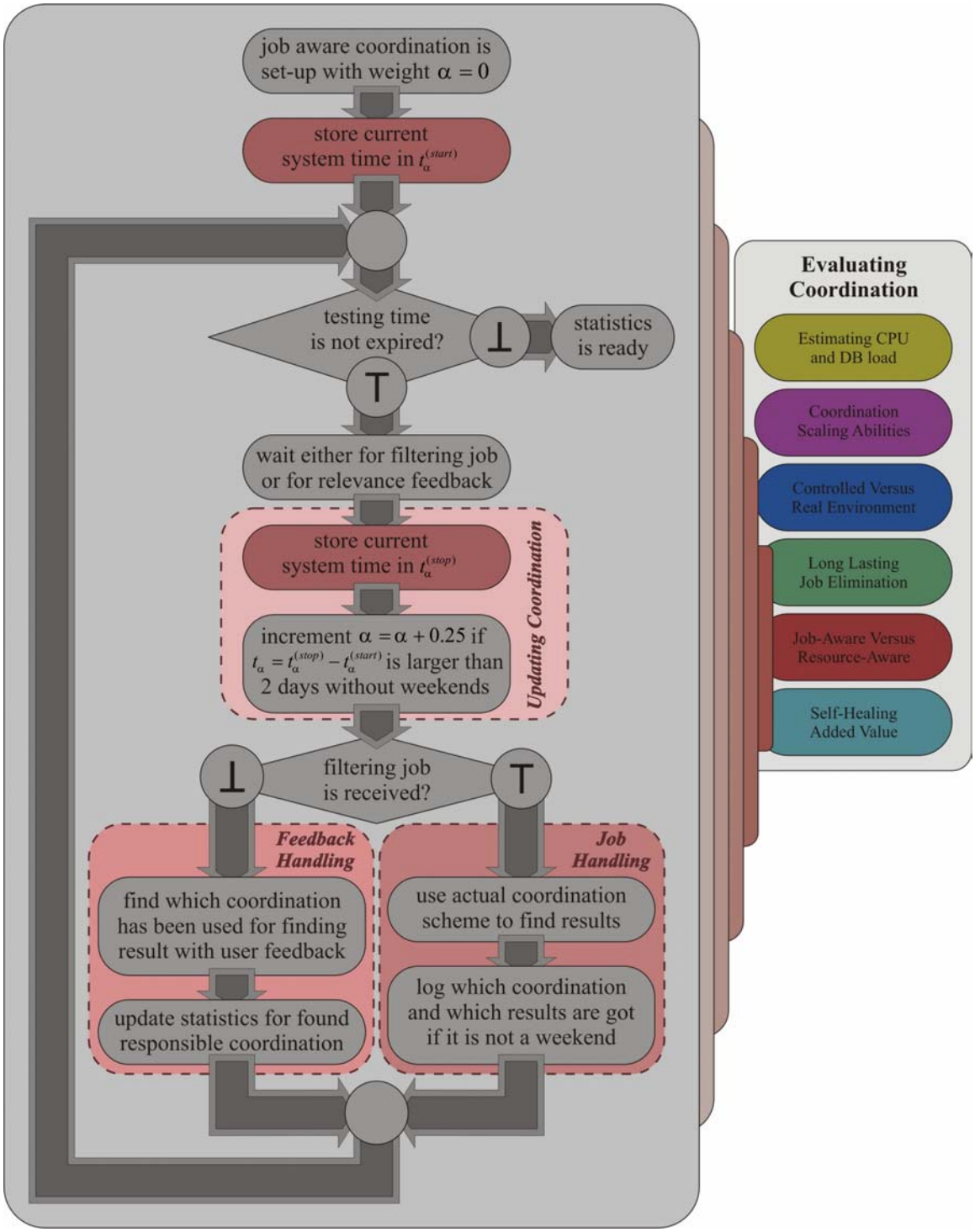

Figure 8.23: Algorithm for evaluating job aware coordination scheme with different α weights

Table 8.14: Results of the two week internal PIA usage that illustrate strengths and weaknesses of coordination mechanisms, having different influence of resource and job content fitness values (different α)

α	n_{fj}	n_{lj}	$avg(d_{fj})$ [s]	$avg(q_{fj})$ [%]
0	47	3	57.8	62.3
0.25	55	0	27.2	65.3
0.5	62	0	24.5	**70.2**
0.75	51	0	23.1	68.1
1	38	0	19.2	58.3

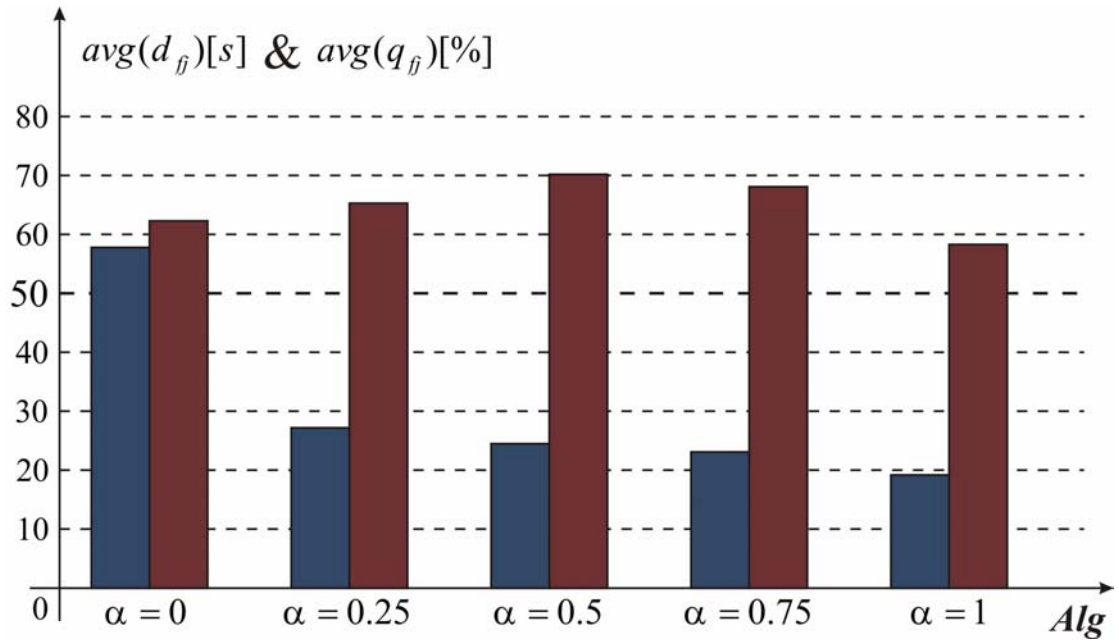

Figure 8.24: Average duration of filtering and obtained user relevance feedback values, corresponding to $avg(d_{fj})$ and $avg(q_{fj})$ columns from Table 8.14.

The results of this 10 day long experiments with the various coordination mechanisms clearly show that the exclusion of a resource fitness component ($\alpha = 0$) potentially leads to the appearance of long-lasting filtering jobs, being exactly the same conclusion as in Section 8.4.4, where the pure resource aware coordination scheme has been tested. At the opposite side, taking care only of the resources ($\alpha = 1$) will successfully eliminate long-lasting jobs, but the average obtained feedback value will also be reduced to only 58.3%. Because the user satisfaction should always be the imperative, the optimal solution is searched for $0 < \alpha < 1$. While the average durations $avg(d_{fj})$ are almost the same when $\alpha \in \{0.25, 0.5, 0.75\}$, the best average user feedback $avg(q_{fj})$ is obtained for $\alpha = 0.5$. The increase of $avg(q_{fj})\big|_{\alpha=0.5}$ is for almost 12%, when compared with $avg(q_{fj})\big|_{\alpha=1}$, whereas at the same time, $avg(d_{fj})\big|_{\alpha=0.5}$ is for around 5 seconds longer than $avg(d_{fj})\big|_{\alpha=1}$. Users are obviously not ready to pay unreasonably high price for the reduction of filtering time, and they are ready to wait little longer in order to get much better recommendations.

8.4.6. Self-Healing Added Value

The efforts to create a smart filtering engine, which will be able to autonomously heal itself, has been the major driving force for extending resource and job aware coordination approaches from Sections 6.3 and 6.4 to additionally take care of how every strategy is promising to be used for recovery. On the one hand, both resource F_r and job content F_c fitness values show how in the current runtime situation a particular strategy can process a received filtering job. While larger F_r means that for a given strategy more favourable resource situation exists, the better suited job will result in higher F_c value. On the other hand, the abilities of every single strategy to be deployed for the recovery are illustrated through healing F_h fitness value, being defined by expression (6.22). In situations where successful recovery is an imperative, large F_h should be a good guideline for selecting promising strategies.

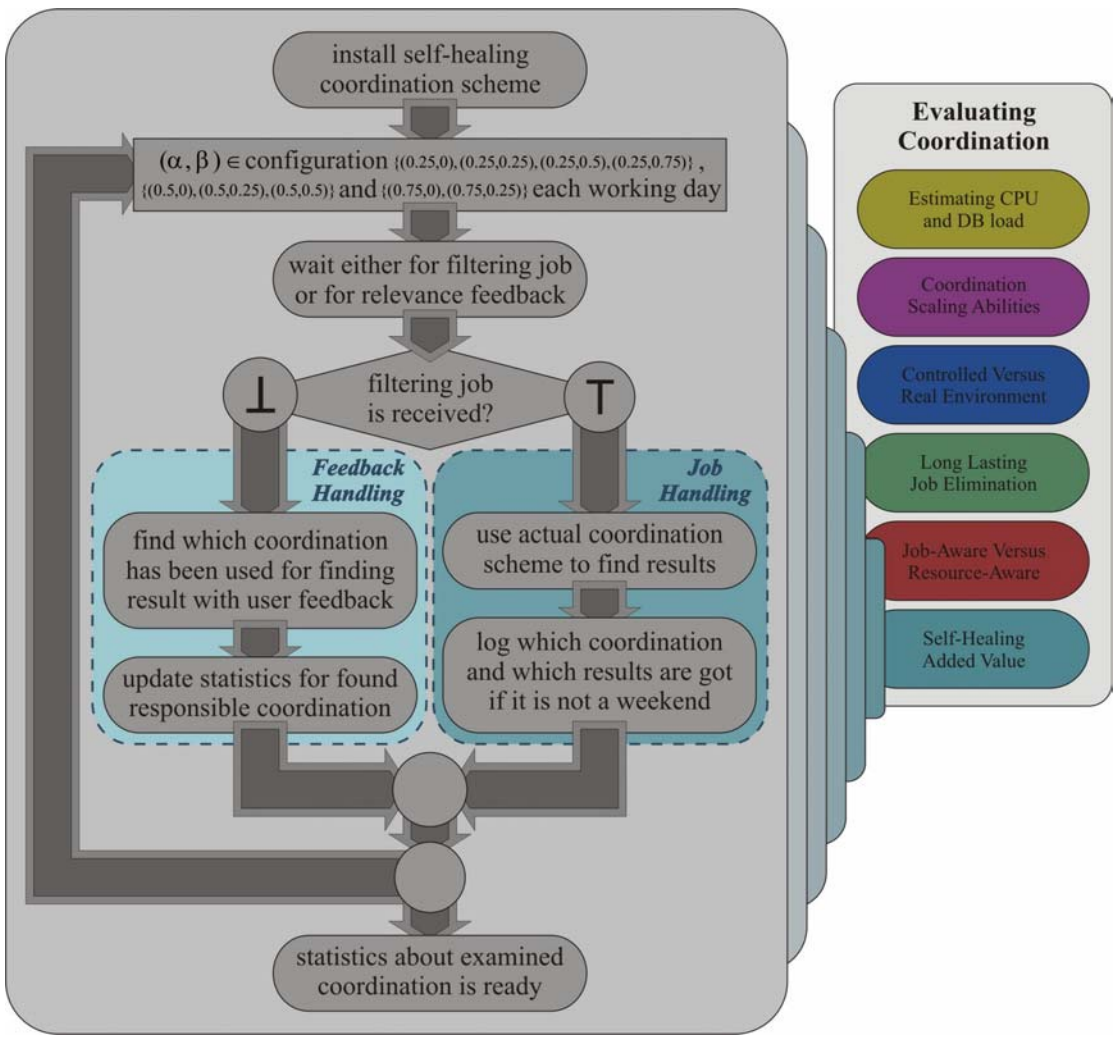

Figure 8.25: Evaluation of different self-healing coordination schemes

To estimate how each of F_r, F_c and F_h fitness values are successful in fulfilling their roles, the performed experiments will be based on varying α and β values in expression (6.23), and in afterwards making comparisons in both relevance feedback and response time domains. The robustness of the self-healing coordination algorithm will be estimated by additionally monitoring the number of corrected failures, showing how many times the requested results have been delivered in spite of problems with the initially selected filtering agent.

From 20th of April till 29th of April 2005, each day different (α,β) configuration was tested, where only working days were again taken into consideration in order to make as fair comparisons as possible. The utilised testing algorithm is summarised on Figure 8.25, having as a main point a loop, where every working day (α,β) configuration is changed to take its new value. The roles of both feedback handling and job handling blocks are the same as it has been the case on Figure 8.23.

The obtained results are given in Table 8.15, where n_{fj} corresponds to the number of filtering jobs for which a feedback has been received, n_{lj} represents the number of long-lasting jobs, having a duration of more than 1000 seconds, n_f is the number of both detected and corrected failures, $avg(d_{fj})$ is the average duration of jobs with a user feedback, and $avg(q_{fj})$ is the average received feedback.

Table 8.15: Results of the nine day internal PIA usage that illustrate the strengths and weaknesses of self healing coordination mechanisms, having different influence of F_r, F_c and F_h fitness values (different α and β)

α	β	n_{fj}	n_{lj}	n_f	$avg(d_{fj})$ [s]	$avg(q_{fj})$ [%]
0.25	0	69	1	9	34.1	67.2
	0.25	82	0	8	32.5	72.4
	0.5	75	0	5	33.2	69.3
	0.75	67	0	4	31.1	54.8
0.5	0	64	0	10	32.1	65.4
	0.25	71	0	7	30.3	**70.1**
	0.5	69	0	6	28.9	52.3
0.75	0	82	0	2	28.2	57.2
	0.25	73	0	8	28.3	55.7

The results of this 9 day long experiments with the different coordination mechanisms clearly show that the exclusion of a job content fitness component, being the case always when $\alpha+\beta=1$, i.e. $(\alpha,\beta)=(0.25,0.75)$, $(\alpha,\beta)=(0.5,0.5)$ and $(\alpha,\beta)=(0.75,0.25)$, leads to a reduction of a user satisfaction. That is the same conclusion as in Section 8.4.5 where the lowest user satisfaction has been obtained in the configuration where only the resource fitness component has been taken into account. As far as the elimination of long lasting jobs is concerned, only one is detected in a configuration $(\alpha,\beta)=(0.25,0)$, which can lead to a conclusion that assigning a weight $\alpha=0.25$ to the F_r is maybe not enough.

More long lasting filtering jobs are not detected by reason of not testing the problematic configurations where $\alpha = 0$. These configurations are intentionally omitted as the results from Section 8.4.4, and especially from Section 8.4.5, have already clearly shown that the resource fitness component has to be taken into consideration in order to even think about the elimination of long lasting jobs.

The number of both detected and recovered failures n_f proves that it has been quite reasonable to improve resource and job aware coordination schemes, being presented in Sections 6.3 and 6.4, in a direction of integrating the automatic healing functionality. The recovery is successful due to both the presence of multiple iterations in a coordination approach, as well as the integration of healing fitness component. Even though a recovery is possible in configurations where $\beta = 0$, the obtained relevance feedback shows that users are more satisfied with filtering results, being obtained when a healing component has influence in coordination process ($\beta > 0$). A difference of more than 20% between the lowest ($avg(q_{fj})\big|_{(\alpha,\beta)=(0.5,0.5)} = 52.3\%$) and the highest ($avg(q_{fj})\big|_{(\alpha,\beta)=(0.25,0.25)} = 72.4\%$) average user feedback values, being detected in spite of the longer duration of filtering ($avg(d_{fj})\big|_{(\alpha,\beta)=(0.25,0.25)} = 32.5s$ and $avg(d_{fj})\big|_{(\alpha,\beta)=(0.5,0.5)} = 28.9s$), definitely proves that the system performances can be significantly tuned by finding the optimal trade-off between F_r, F_c and F_h values. Users are obviously ready to wait little longer in order to get better recommendations, and the well tuned filtering system should know how much that longer can be.

8.5. Conclusion

Cooperation, exploration, and coordination mechanisms, being introduced in Chapters 4, 5 and 6, have been evaluated in both the simulated settings and the real time conditions in this Chapter. Always when being necessary, a solid test environment has been found in the PIA system, being locally developed to serve as one intelligent personal information assistant.

The cooperation approach has been evaluated in Section 8.2 by using both the small-scale simulated environment, as well as the real PIA system. On the one hand, centralised and distributed systems have been compared in Section 8.2.1 by computing precision and recall values. It has been found that a distributed system with 5 cooperative communities has resulted in the precision and recall values that are 6% and 4% smaller than when the centralised solution has been used. On the other hand, the elimination of the long filtering jobs (Section 8.2.2), updating descriptions (Section 8.2.3) and utilising the inaccurate descriptions (Section 8.2.4) have shown in real PIA settings that there are no more jobs with a duration over 1000 seconds, as well as that a distributed system is able to learn and track the dynamics of the underlying domains.

A combined distance function has been analysed in the course of the evaluation of the exploration approach in Section 8.3.1, and it has been got that the combination between Euclidean distance and Jaccard index provides better results than any separate similarity solution. The comparison with the pure collaborative filtering and mutation & crossover

techniques has shown in Section 8.3.2 that a proposed exploration approach has superior behaviour especially when the number of off-topic results is taken into account.

The most important coordination approach has been evaluated in Section 8.4, first by finding that the load of system resources can be reliably measured by a proposed method, based on a measurement of the execution time (Section 8.4.1). The evaluation of scaling abilities in Section 8.4.2 has demonstrated that proposed approaches scale quite well, and that can be applied even inside communities with 5000 agents. The most important tests have been reported in Section 8.4.3, where it has been shown that jobs with durations that are shorter than 10 seconds can be efficiently used as the past experience while adapting resource related fitness values. The evaluation of coordination approach has been finished by proving the abilities to eliminate the long lasting jobs due to not choosing the filtering agent for which unfavourable resource situation exists (Section 8.4.4), the reasonability of taking care of the job properties (Section 8.4.5), and finally the increased robustness due to the ability to recover after failures (Section 8.4.6).

Chapter 9
Conclusion

This thesis has aimed to provide an agent-based solution for information filtering and retrieval problems, and to make at the same time the abstraction from algorithms that any particular searching strategy uses in the course of finding requested results. The presented cooperation, exploration and coordination mechanisms have been therefore realised to be completely independent from the utilised filtering strategies. Due to the usage of JIAC IV for developing the underlying filtering framework, whose foundation has been made from these mechanisms, novel searching heuristics can be easily integrated as new agents even during the runtime.

A distributed information retrieval has been addressed as cooperation among multiple filtering communities, being compactly represented by their descriptions that contain the chosen set of the most important words from underlying collections. While communities find each other by simply exchanging their descriptions, the largest contribution has been achieved by learning when the particular description becomes obsolete due to the changes of a supported collection, and consequently the new one should be requested. An optimal usage of network resources has been reached, being very important for every distributed system.

A contribution of proposed cooperation mechanisms has been also found in resolving the information fusion. Instead of performing too costly re-examination of results, which will unacceptably extend the delivery of final recommendations, the results are re-ranked based on their locally predicted relevance and the reliability of a responsible community. Since such reliability represents the successfulness of the given community in finding the results that users have liked, its adaptation routine has been developed to utilise available user relevance feedback values.

The solution for intelligently handling imperfect queries has been found in exploration mechanisms, which are able to fix human mistakes and to increase chances that the good results will be delivered. These corrections have been reached by finding the most similar old requests and in using them while adapting the actual one. The needed adaptation has been fulfilled not only by adding the attributes that are missing in the actual request and that are important for the found similar ones, but also by correcting suspicious attributes again under the guidance of similar requests. The most important added value has been found in controlling the level of adaptation, being the insurance that the provided results will not be far away from the manually specified preferences.

The comparison of requests in the highly dimensional space, being typical in a domain of text documents where every distinct word plays the role of a separate attribute, might be a quite challenging task. Since different distance functions have usually their strengths and weaknesses, the solution has been found in combining two of them. While Euclidean distance has been chosen due to its good behaviour when both values and weights of an attribute should be taken into account, Jaccard index has happened to be quite important by reason of its solid performances in sparse spaces. The fulfilled small experiment has shown that such combined distance outperforms any single solution, and has also opened a challenging field of somehow learning the optimal combination weights.

The biggest contribution of this thesis, which has also attracted the greatest attention, relates to three coordination mechanisms, being used for making a benefit from multiple strategies. All these mechanisms have been based on the usage of proportional or roulette wheel selection scheme for deciding which filtering strategies should fulfil the actual job. Since the probability of being selected is proportional to the estimation of an applicability of a given strategy, a roulette wheel approach gives fair chances to everybody. A system is finally capable to learn about the real capabilities of a chosen strategy, either based on a measurement of the response time or utilising user feedback values. Such an adaptation of knowledge about available strategies has finally completed a picture about a trade-off between exploiting well-known strategies and exploring the novel ones.

The first coordination scheme has been named resource aware by reason of taking care of the available resources before deciding which strategy will do filtering. The motivation has been found in the reasoning that even the brilliant filtering strategy will fail to deliver the expected results in the case where the needed resources are not enough available. As the accurate estimation of resource load values has been found to be very important, the specially designed expressions are executed, and the load is computed by measuring the execution time. By not basing the load estimation on integrated system functions, which are maybe not available on all platforms, the proposed approach has become independent from the supporting operating system.

The idea to take into account the properties of jobs, while estimating the applicability of every filtering strategy, has mainly motivated a development of job aware coordination mechanisms. The currently used job properties are rather very simple, such as the number of words and phrases, mostly because there are strategies, which index the positions of words and therefore can explicitly search for sought phrases, but there are also techniques, which apply a dimensionality reduction and consequently are excellent for requests with many words. Never the less, the proposed approach is fully open for the integration of the additional properties as soon as available filtering strategies manage to prove the reasons for doing that.

The major coordination achievement has been realised as a self-healing scheme, which is able to autonomously recover after the failures of the selected filtering strategies. The proposed approach is able to formally represent the seriousness of a particular situation, regarding the lost time and the number of currently available results. By using a previous experience about other strategies, a system has been consequently capable to optimally recover itself, and to deliver as good filtering results as possible.

References

[1] Albayrak Sahin, *Agent-oriented Technology for Telecommunications: Introduction*, Communications of the ACM (CACM), Volume 44, Number 4, pp. 30-33, 2001.

[2] Albayrak Sahin, Bsufka Karsten, Kirchwitz Andreas, Schlenker Hans, Seidler Kai, Sesseler Ralf, *Agent-Based Traffic Telematics Service*, In Intelligent Agents in Telecommunications Applications – Basics, Tools, Languages and Applications, Albayrak Sahin (Ed.), IOS Press, Netherlands, pp. 148-170, 1998.

[3] Albayrak Sahin, Haase Raik, Riegel Holger, *The Unified Messaging Agent Service (UniMAS)*, In Intelligent Agents in Telecommunications Applications – Basics, Tools, Languages and Applications, Albayrak Sahin (Ed.), IOS Press, Netherlands, pp. 130-147, 1998.

[4] Albayrak Sahin, *Introduction to Agent Oriented Technology for Telecommunication*, In Intelligent Agents in Telecommunications Applications – Basics, Tools, Languages and Applications, Albayrak Sahin (Ed.), IOS Press, Netherlands, pp. 1-20, 1998.

[5] Albayrak Sahin, Kirsch Dietrich, *Agent-Based Travel Information Service*, In Intelligent Agents in Telecommunications Applications – Basics, Tools, Languages and Applications, Albayrak Sahin (Ed.), IOS Press, Netherlands, pp. 171-185, 1998.

[6] Albayrak Sahin, Milosevic Dragan, *Collaborative Filtering Guided Exploration in Multi Agent Filtering Framework*, IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE-05), Hong Kong, China, pp. 760-767, 2005.

[7] Albayrak Sahin, Milosevic Dragan, *Component Based Local Optimisation Filtering Agent*, 1st World Congress on Lateral Computing (WCLC 04) in cooperation with IEEE and World Federation on Soft Computing (WFSC), Bangalore, India, 2004.

[8] Albayrak Sahin, Milosevic Dragan, *Component Based Local Optimisation Filtering Agent*, International Journal of Lateral Computing (IJLC), Volume 1, Number 2, pp. 16-23, 2005.

[9] Albayrak Sahin, Milosevic Dragan, *Cooperative Community Selection in Multi Agent Filtering Framework*, IEEE/WIC/ACM International Joint Conference on Intelligent Agent technology (IAT 05) and Web Intelligence (WI 05), Compiegne University of Technology, France, 2005 (to appear)

[10] Albayrak Sahin, Milosevic Dragan, *Distributed Community Cooperation In Multi Agent Filtering Framework*, 7th International Conference on Enterprise

Information Systems (ICEIS 05) in the cooperation with IEEE and ACM, Miami, Florida, USA, pp. 400-407, 2005.

[11] Albayrak Sahin, Milosevic Dragan, *Flexible Strategy Coordination Approach for Supporting Resource Awareness in Multi Agent Filtering Framework*, 4th International Conference on Machine Learning and Cybernetics (ICMLC 2005) in cooperation with IEEE, Guangzhou, China, pp. 5615-5623, 2005.

[12] Albayrak Sahin, Milosevic Dragan, *Generic Intelligent Personal Information Agent*, International Conference on Advances in Internet, Processing, Systems, and Interdisciplinary Research (IPSI-2004S) in cooperation with Yu IEEE Computer Chapter, Serbia, 2004.

[13] Albayrak Sahin, Milosevic Dragan, *Multi-Domain Strategy Coordination Approach for Optimal Resource Usage in Agent Based Filtering Framework*, International Journal of Web Intelligence and Agent Systems (WIAS), IOS Press, Amsterdam, 2005, (to appear)

[14] Albayrak Sahin, Milosevic Dragan, *Response Time Aware Coordination in Multi Agent Filtering Framework*, 18th International FLAIRS Conference (FLAIRS 05) in the cooperation with The American Association of Artificial Intelligence (AAAI), Florida Artificial Intelligence Research Society, Clearwater Beach, Florida, USA, pp. 806-807, 2005.

[15] Albayrak Sahin, Milosevic Dragan, *Resource And Job Aware Coordination In Multi Agent Filtering Framework*, Artificial Intelligence And Applications (AIA 2005), 23rd IASTED International Multi-Conference on Applied Informatics, Innsbruck, Austria, pp. 539-544, 2005.

[16] Albayrak Sahin, Milosevic Dragan, *Resource Load Predictable Coordination in Multi Agent Filtering Framework*, International Conference on Advances in Intelligent Systems - Theory and Applications in the cooperation with IEEE Computer Society (AISTA 04), Luxembourg-Kirchberg, Luxembourg, 2004.

[17] Albayrak Sahin, Milosevic Dragan, *Self Improving Coordination in Multi Agent Filtering Framework*, IEEE/WIC/ACM International Joint Conference on Intelligent Agent technology (IAT 04) and Web Intelligence (WI 04), Beijing, China, pp. 239-245, 2004.

[18] Albayrak Sahin, Milosevic Dragan, *Situation-aware Coordination in Multi Agent Filtering Framework*, 19th International Symposium on Computer and Information Sciences (ISCIS 04) in cooperation with Lecture Notes in Computer Science by Springer-Verlag (LNCS), Antalya, Turkey, pp. 480-493, 2004.

[19] Albayrak Sahin, Wollny Stefan, Varone Nicolas, Lommatzsch Andreas, Milosevic Dragan, *Agent Technology for Personalized Information Filtering: The PIA-System*, The 20th ACM Symposium on Applied Computing, Agents, Interactions, Mobility, and Systems (AIMS) track, Santa Fe, New Mexico, USA, pp. 54-59, 2005.

[20] Albayrak Sahin, Wieczorek Dirk, *JIAC – An Open and Scalable Agent Architecture for Telecommunication Applications*, In Intelligent Agents in Telecommunications Applications – Basics, Tools, Languages and Applications, Albayrak Sahin (Ed.), IOS Press, Netherlands, pp. 189-202, 1998.

[21] Albayrak Sahin, Wieczorek Dirk, *JIAC – A Toolkit for Telecommunication Applications*, 3rd International Workshop on Intelligent Agents for Telecommunication Applications (IATA 99), Albayrak Sahin (Ed.), Stockholm, Sweden, pp. 1-18, 1999.

[22] Allan James, *Incremental Relevance Feedback for Information Filtering*, 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 96), Zurich, Switzerland, pp. 270-278, 1996.

[23] Amato Giuseppe, Straccia Umberto, *User Profile Modeling and Applications to Digital Libraries*, 3rd European Conference on Research and Advanced Technology for Digital Libraries (ECDL 99), Paris, France, pp. 184-197, 1999.

[24] Andreevskaia Alina, Abi-Aad Rony, Radhakrishnan Thiruvengadam, *Agent-Mediated Knowledge Acquisition for User Profiling*, In Intelligent Agents for Data Mining and Information Retrieval, Idea Group Publishing, pp. 164-187, 2004.

[25] Appian Personalization Consulting, *Appian Web Personalization Report*, Online at http://www.appiancorp.com/report/

[26] Arens Yigal, Knoblock Craig, *Intelligent Caching: Selecting, Representing, and Reusing Data in an Information Server*, 3rd International Conference on Information and Knowledge Management (CIKM 1994), Gaithersburg, Maryland, pp. 433-438, 1994.

[27] Armstrong Robert, Dayne Freitag, Thorsten Joachims, Mitchell Tom, *WebWatcher: A Learning Apprentice for the World Wide Web*, AAAI Spring Symposium on Information Gathering from Heterogeneous, Distributed Environments, Stanford, 1995.

[28] Balabanovic Marko, *An Adaptive Web Page Recommendation Service*, 1st International Conference on Autonomous Agents (Agents 97), Marina del Rey, California, USA, pp. 378-385, 1997.

[29] Balabanovic Marko, *Learning to Surf: Multiagent Systems for Adaptive Web Page Recommendation*, Ph.D. Thesis, Stanford University, 1998.

[30] Balabanovic Marko, Shoham Yoav, *Fab: Content-Based Collaborative Recommendation*, Communication of the ACM, Volume 40, Number 3, pp. 66-72, 1997.

[31] Balabanovic Marko, Shoham Yoav, *Learning information retrieval agents: Experiments with automated Web browsing*, AAAI Spring Symposium on Information Gathering from Heterogeneous, Distributed Environments, Stanford, 1995

[32] Balabanovic Marko, Shoham Yoav, Yun Yeogirl, *An Adaptive Agent for Automated Web Browsing*, Technical Report, Department of Computer Science, Stanford University, 1997.

[33] Baldwin Mark, *A New Factor in Evolution*, The American Naturalist 30, June 1896, pp. 441-451, 536-553.

[34] Bauer Travis, Leake David, *Real Time User Context Modeling for Information Retrieval Agents*, ACM CIKM International Conference on Information and

Knowledge Management (CIKM 2001), Atlanta, Georgia, USA, pp. 568-570, 2001.

[35] Bayardo Roberto, *Efficiently Mining Long Patterns from Databases*, ACM SIGMOD International Conference on Management of Data (SIGMOD Conference 1998), Seattle, USA, pp. 85-93, 1998.

[36] Bäck Thomas, *Selective Pressure in Evolutionary Algorithms: A Characterization of Selection Mechanisms*, 1st IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence (ICEC 1994), Orlando, USA, pp. 57-62, 1994.

[37] Beenen Gerard, Ling Kimberly, Wang Xiaoqing, Chang Klarissa, Frankowski Dan, Resnick Paul, Kraut Robert, *Using Social Psychology to Motivate Contributions to Online Communities*, ACM CSCW 2004 Conference on Computer Supported Cooperative Work, Chicago, pp. 212-221, 2004.

[38] Beran Tomas, Macek Tomas, *Text Processing by Binary Neural Networks*, In Intelligent Agents for Data Mining and Information Retrieval, Idea Group Publishing, pp. 125-136, 2004.

[39] Berenci Ezio, Carpineto Claudio, Giannini Vittorio, Mizzaro Stefano, *Effectiveness of Keyword-based Display and Selection of Retrieval Results for Interactive Searches*, International Journal on Digital Libraries, Volume 3, Number 3, pp. 249-260, 2000.

[40] Billsus Daniel, Pazzani Michael, *A Hybrid User Model for News Story Classification*, 17th International Conference on User Modeling (UM 1999), Banff, Canada, pp. 97-108, 1999.

[41] Billsus Daniel, Pazzani Michael, *A Personal News Agent That Talks, Learns and Explains*, 3rd Annual Conference on Autonomous Agents (Agents 1999), Seattle, USA, pp. 268-275, 1999.

[42] Billsus Daniel, Pazzani Michael, Chen James, *A Learning Agent for Wireless News Access*, International Conference on Intelligent User Interfaces (IUI), New Orleans, USA, pp. 33-36, 2000.

[43] Billsus Daniel, Pazzani Michael, *Learning Probabilistic User Models*, Workshop notes on Machine Learning for User Modeling held during the 6th International Conference on User Modeling, Sardinia, Italy, 1997.

[44] Billsus Daniel, Pazzani Michael, *Revising User Profiles: The Search for Interesting Web sites*, 3rd International Multi-Strategy Learning Conference, Harpers Ferry, USA, pp. 232-243, 1996.

[45] Billsus Daniel, Pazzani Michael, *User Modeling for Adaptive News Access*, User Modeling and User-Adapted Interaction, Volume 10, Number 2-3, pp. 147-180, 2000.

[46] Blackmore Justine, Miikkulainen Risto, *Incremental Grid Growing: Encoding High-Dimensional Structure into a Two-Dimensional Feature Map*, IEEE International Conference on Neural Networks (ICNN 93), San Francisco, USA, pp. 450-455, 1993.

[47] Blake Catherine, Pratt Wanda, *Better Rules, Fewer Features: A Semantic Approach to Selecting Features from Text*, 2001 IEEE International Conference on Data Mining (ICDM 2001), San Jose, USA, pp. 59-66, 2001.

[48] Blok Henk Ernst, Vries Arjen, Blanken Henk, *Top N MM Query Optimization: The Best of Both IR and DB Worlds*, Information Science Conference (CIW 99), Amsterdam, pp. 53-69, 1999.

[49] Boger Zvi, Kuflik Tsvi, Shapira Bracha, Shoval Peretz, *Information Filtering and Automatic Keyword Identification by Artificial Neural Networks*, 8th European Conference on Information Systems (ECIS 2000), Vienna, Austria, pp. 379-385, 2000.

[50] Boley Daniel, *Principal Direction Divisive Partitioning*, Data Mining and Knowledge Discovery, Volume 2, Number 4, pp. 325-344, 1998.

[51] Bollacker Kurt, Lawrence Steve, Giles Lee, *A System for Automatic Personalized Tracking of Scientific Literature on the Web*, 4th ACM conference on Digital Libraries (ACM DL 1999), Berkeley, USA, pp. 105-113, 1999.

[52] Bollacker Kurt, Lawrence Steve, Giles Lee, *CiteSeer: An Autonomous Web Agent for Automatic Retrieval and Identification of Interesting Publications*, 2nd International Conference on Autonomous Agents (Agents 1998), Minneapolis, USA, pp. 116-123, 1998.

[53] Bollacker Kurt, Lawrence Steve, Giles Lee, *Discovering Relevant Scientific Literature on the Web*, IEEE Intelligent Systems, Volume 15, Number 2, pp. 42-47, 2000.

[54] Borchers Al, Herlocker Jonathan, Riedl John, *Ganging up on Information Overload*, IEEE Computer, Volume 31, Number 4, pp. 106-108, 1998.

[55] Borzsonyi Stephan, Kossmann Donald, Stocker Konrad, *The Skyline Operator*, 17th IEEE International Conference on Data Engineering (ICDE 01), Heidelberg, Germany, pp. 421-430, 2001.

[56] Bridge Derek, *Defining and Combining Symmetric and Asymmetric Similarity Measures*, 4th European Workshop on Advances in Case-Based Reasoning (EWCBR 98), Barry Smyth and Padraig Cunningham (Eds.), Dublin, Ireland, pp. 52-63, 1998.

[57] Brin Sergey, Motwani Rajeev, Ullman Jeffrey, Tsur Shalom, *Dynamic Itemset Counting and Implication Rules for Market Basket Data*, ACM SIGMOD International Conference on Management of Data (SIGMOD Conference 1997), Tucson, USA, pp. 255-264, 1997.

[58] Brin Sergey, Page Lawrence, *The anatomy of a large-scale hyper textual (Web) search engine*, Proc. 7th International World Wide Web Conference on Computer Networks, Volume 30, Numbers 1-7, pp. 107-117, 1998.

[59] Brown Eric, *Execution Performance Issues in Full-Text Information Retrieval*, University of Massachusetts, Ph.D. Thesis, 1995.

[60] Byoung-Tak Zhang, Young-Woo Seo, *Personalized Web-Document Filtering Using Reinforcement Learning*, Applied Artificial Intelligence, Volume 15 Number 7, pp. 665-685, 2001.

[61] Callan James, *Learning While Filtering Documents*, 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 1998), Melbourne, Australia, pp. 224-231, 1998.

[62] Camacho David, Aler Ricardo, Cuadrado Juan, *Rule-Based Parsing for Web Data Extraction*, In Intelligent Agents for Data Mining and Information Retrieval, Idea Group Publishing, pp. 65-86, 2004.

[63] Carpineto Claudio, Mori de Renato, Romano Giovanni, Bigi Brigitte, *An Information-Theoretic Approach to Automatic Query Expansion*, ACM Transactions on Information Systems (TOIS), Volume 19, Number 1, pp. 1-27, 2001.

[64] Carpineto Claudio, Romano Giovanni, Adamo Paolo, *Inferring Dependencies from Relations: A Conceptual Clustering Approach*, Computational Intelligence, Volume 15, Number 4, pp. 415-441, 1999.

[65] Carpineto Claudio, Romano Giovanni, *A Lattice Conceptual Clustering System and Its Application to Browsing Retrieval*, Machine Learning, Volume 24, Number 2, pp. 95-122, 1996.

[66] Carpineto Claudio, Romano Giovanni, *Exploiting the Potential of Concept Lattices for Information Retrieval with CREDO*, Journal of Universal Computer Science, Volume 10, Number 8, pp. 985-1013, 2004.

[67] Carpineto Claudio, Romano Giovanni, Giannini Vittorio, *Improving Retrieval Feedback with Multiple Term-Ranking Function Combination*, ACM Transactions on Information Systems (TOIS), Volume 20, Number 3, pp. 259-290, 2002.

[68] Carpineto Claudio, Romano Giovanni, *Information Retrieval through Hybrid Navigation of Lattice Representations*, International Journal of Human Computer Studies, Volume 45, Number 5, pp. 553-578, 1996.

[69] Castellanos Malu, *HotMiner-Discovering hot Topics from Dirty Text*, In Survey of Text Mining: Clustering, Classification, and Retrieval, Michael Berry (Ed.), Springer-Verlag, pp. 123-158, 2003.

[70] Cavnar William, Trenkle John, *N-Gram based text categorization*, 3rd Annual Symposium on Document Analysis and Information Retrieval, Las Vegas, NV, pp. 161-175, 1994.

[71] Charikar Moses, Chekuri Chandra, Feder Tomas, Motwani Rajeev, *Incremental clustering and dynamic information retrieval*, In Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, 29th STOC, El Paso, Texas, USA, pp. 626-635, 1997.

[72] Chatfield Chris, Koehler Anne, Ord Keith, Snyder Ralph, *A new look at models for exponential smoothing*, The Statistician (Journal of the Royal Statistical Society, Series D), Volume 50, pp. 147-159, 2001.

[73] Chau Rowena, Yeh Chung-Hsing, *Multilingual Web Content Mining: A User-Oriented Approach*, In Intelligent Agents for Data Mining and Information Retrieval, Idea Group Publishing, pp. 88-100, 2004.

[74] Chen Liren, Sycara Katia, *WebMate: A Personal Agent for Browsing and Searching*, 2nd International conference on Autonomous agents, Minneapolis, USA, pp. 132-139, 1998.

[75] Claypool Mark, Brown David, Le Phong, Waseda Makoto, *Inferring User Interest*, IEEE Internet Computing, Volume 5, Number 6, pp. 32-39, 2001.

[76] Claypool Mark, Gokhale Anula, Miranda Tim, Murnikov Pavel, Netes Dmitry, Sartin Natthew, *Combining Content-Based and Collaborative Filters in an Online Newspaper*, ACM SIGIR Workshop on Recommender Systems, Berkeley, CA, 1999.

[77] Cornelson Michael, Greenglass Ed, Grossman Robert, Karidi Ron, Shnidman Daniel, *Combining Families of Information Retrieval Algorithms Using Metalearning*, In Survey of Text Mining: Clustering, Classification, and Retrieval, Michael Berry (Ed.), Springer-Verlag, pp. 159-169, 2003.

[78] Cotter Paul, Smyth Barry, *PTV: Intelligent Personalised TV Guides*, 17th National Conference on Artificial Intelligence and 12th Conference on Innovative Applications of Artificial Intelligence (AAAI/IAAI 2000), Austin, USA, pp. 957-964, 2000.

[79] Cutting Douglas, Karger David, Pedersen Jan, *Constant Interaction-Time Scatter/Gather Browsing of Very Large Document Collections*, 16th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval (SIGIR 1993), Pittsburgh, USA, pp. 126-134, 1993.

[80] Cutting Douglas, Pedersen Jan, Karger David, Tukey John, *Scatter/Gather: A Cluster-based Approach to Browsing Large Document Collections*, 15th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 1992), Copenhagen, Denmark, pp. 318-329, 1992.

[81] Decker Keith, Sycara Katia, Williamson Mike, *Middle-Agents for the Internet*, In Proceedings of the 15th International Joint Conference on Artificial Intelligence, Nagoya, Japan, 1997.

[82] Delgado Joaquin, *Agent based information filtering and recommender systems*, Ph.D. Thesis, Nagoya Institute of technology, Japan, 2000.

[83] Delgado Joaquin, Davidson Richard, *Knowledge bases and user profiling in travel and hospitality recommender systems*, In Proceedings of the ENTER 2002 Conference, Innsbruck, Austria, January 22-25 2002., Springer Verlag, pp. 1-16.

[84] Deters Ralph, *Scalability and Information Agents*, Applied Computing Review, Volume 9, Number 3, pp. 13-20, 2001.

[85] Deters Ralph, *Scalable Multi-Agent Systems*, ACM 2001 Java Grande Conference (Java Grande 2001), Stanford University, California, USA, p 182, 2001.

[86] Dhillon Inderjit, Kogan Jacob, Nicholas Charles, *Feature Selection and Document Clustering*, In Survey of Text Mining: Clustering, Classification, and Retrieval Michael Berry (Ed.), Springer-Verlag, pp. 73-100, 2003.

[87] Dittenbach Michael, Merkl Dieter, Berger Helmut, *What customers really want to know from tourism information systems but never dared to ask*, 5th

International Conference on Electronic Commerce Research (ICECR-5), Montreal, Canada.

[88] Dittenbach Michael, Merkl Dieter, Rauber Andreas, *Growing hierarchical self-organizing map*, In Proceedings of the International Joint Conference on Neural Networks, volume 6, Piscataway, NJ, Technical University Wien, IEEE, pp. 15–19, 2000.

[89] Dodero Juam Manuel, Diaz Paloma, Aedo Ignacio, *A Multi-agent Approach to Collaborative Knowledge Production*, In Intelligent Agents for Data Mining and Information Retrieval, Idea Group Publishing, pp. 31-46, 2004.

[90] El-Baltagy Samhaa, Rafea Ahmed, Abdelhamid Yasser, *Using Dynamically Acquired Background Knowledge For Information Extraction And Intelligent Search*, In Intelligent Agents for Data Mining and Information Retrieval, Idea Group Publishing, pp. 196-207, 2004.

[91] English Thomas, *Evaluation of Evolutionary and Genetic Optimizers: No Free Lunch*, 5th Annual Conference on Evolutionary Programming Evolutionary Programming V), San Diego, CA, USA, pp. 163-169, 1996.

[92] Eppstein David, *Fast Hierarchical Clustering and Other Applications of Dynamic Closest Pairs*, ACM Journal of Experimental Algorithms, Volume 5, Electronic Edition, 2000.

[93] Ester Martin, Kriegel Hans-Peter, Sander Jörg, Xu Xiaowei, *A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise*, In Proc. 2nd Int. Conf. on Knowledge Discovery and Data Mining (KDD'96), AAAI Press, Menlo Park, CA, pp. 226-231, August 1996.

[94] Faloutsos Christos, Oard Douglas, *A Survey of Information Retrieval and Filtering Methods*, University of Maryland, Technical Report CS-TR-3514, August 1995.

[95] Fang Wang, *Self-organizing Communities Formed by Middle Agents*, 1st International Joint Conference on Autonomous Agents and Multi-agent Systems (AAMAS 2002), Bologna, Italy, pp. 1333-1339, 2002.

[96] Fan Weiguo, Gordon Michael, Pathak Praveen, *Personalization of search engine services for effective retrieval and knowledge management*, In Proceedings of the 2000 International Conference on Information Systems, pp. 20-34.

[97] Farnstrom Fredrik, Lewis James, Elkan Charles, *Scalability for Clustering Algorithms Revisited*, ACM SIGKDD Explorations, Volume 2, Number 1, pp. 51-57, August 2000.

[98] Fayyad Usama, Piatetsky-Shapiro Gregory, Smyth Padhraic, *From Data Mining to Knowledge Discovery in Databases*, AI Magazine, Volume 17, Number 3, pp. 37-54, 1996.

[99] Fink Josef, Kobsa Alfred, *A review and analysis of commercial user modelling servers for personalization on the World Wide Web*, User Modelling and User-Adapted Interaction, Volume 10, Numbers 2-3, pp. 209-249, 2000.

[100] Flake Stephan, Geiger Christian, Lehrenfeld Georg, Mueller Wolfgang, Paelke Volker, *Agent-Based Modeling for Holonic Manufacturing Systems with Fuzzy*

Control, 18th International Conference of the North American Fuzzy Information Processing Society (NAFIPS'99), New York, 1999.

[101] Fogel David, *Evolutionary Entertainment with Intelligent Agents*, IEEE Computer, Volume 36, Number 6, pp. 106-108, 2003.

[102] Franconi Enrico, Kuper Gabriel, Lopatenko Andrei, Zaihrayeu Ilya, *A Distributed Algorithm for Robust Data Sharing and Updates in P2P Database Networks*, EDBT Workshops 2004, Heraklion, Crete, Greece, pp. 446-455, 2004.

[103] Franconi Enrico, Kuper Gabriel, Lopatenko Andrei, Zaihrayeu Ilya, *Queries and Updates in the coDB Peer to Peer Database System*, 30th International Conference on Very Large Data Bases (VLDB 2004), Toronto, Canada, pp. 1277-1280, 2004.

[104] Fricke Stefan, Bsufka Karsten, Keiser Jan, Schmidt Torge, Sesseler Ralf, Albayrak Sahin, *Agent-based Telematic Services and Telecom Applications*, Communications of ACM, Volume 44, Number 4, pp. 43-48, April 2001.

[105] Frigui Hichem, Nasraoui Olfa, *Simultaneous clustering and dynamic keyword weighting for text documents*, In Survey of Text Mining Clustering, Classification, and Retrieval, Berry Michael (Ed.), Springer-Verlag, pp. 45-72, 2004.

[106] Frigui Hichem, Nasraoui Olfa, *Simultaneous Clustering and Attribute Discrimination*, Proceedings of the Ninth IEEE International Conference on Fuzzy Systems, San Antonio, TX, pp. 158-163, May 2000.

[107] Frigui Hichem, Nasraoui Olfa, *Simultaneous Clustering and Dynamic Keyword Weighting for Text Documents*, In Survey of Text Mining: Clustering, Classification, and Retrieval, Michael Berry (Ed.), Springer-Verlag, pp. 45-72, 2003.

[108] Gaede Volker, Günther Oliver, *Multidimensional Access Methods*, ACM Computing Surveys (CSUR), Volume 30, Number 2, pp. 170-231, June 1998.

[109] Gaustad Tanja, Gosse Bouma, *Accurate Stemming of Dutch for Text Classification*, Computational Linguistics in the Netherlands, 2002.

[110] Gelati Gionata, Guerra Francesco, Vincini Maurizio, *Agents Supporting Information Integration: The MIKS Framework*, 2nd AI*IA TABOO Joint Workshop From Objects to Agents, Evolutive Trends of Software Systems (WOA 2001), Modena, Italy, pp. 109-112, 2001.

[111] Gerber Christian, *Self-Adaptation and Scalability in Multi-Agent Societies*, PhD thesis, University of the Saarland, 2000.

[112] Giegerich Robert, Kurtz Stefan, *A Comparison of Imperative and Purely Functional Suffix Tree Constructions*, Science of Computer Programming, Volume 25, Numbers 2-3, pp. 187-218, 1995.

[113] Giles Lee, Bollacker Kurt, Lawrence Steve, *CiteSeer: An Automatic Citation Indexing System*, 3rd ACM International Conference on Digital Libraries (ACM DL 1998), Pittsburgh, USA, pp. 89-98, 1998.

[114] Giunchiglia Fausto, Zaihrayeu Ilya, *Implementing Database Coordination in P2P Networks*, 2nd Workshop on Semantics in Peer-to-Peer and Grid Computing (SemPGrid'04), New York, USA, 2004.

[115] Giunchiglia Fausto, Zaihrayeu Ilya, *Making peer databases interact - a vision for an architecture supporting data coordination*, 6th International Workshop on Cooperative Information Agents (CIA-2002), Universidad Rey Juan Carlos, Madrid, Spain, September 18 -20, 2002.

[116] Goldberg Ken, Theresa Roeder, Dhruv Gupta, Chris Perkins, *Eigentaste: A Constant Time Collaborative Filtering Algorithm*, UCB ERL Technical Report M00/41, August 2000.

[117] Good Nathaniel, Schafer Ben, Konstan Joseph, Borchers Al, Sarwar Badrul, Herlocker Jonathan, Riedl John, *Combining Collaborative Filtering with Personal Agents for Better Recommendations*, 16th National Conference on Artificial Intelligence and 11th Conference on Innovative Applications of Artificial Intelligence (AAAI/IAAI 1999), Orlando, USA, pp. 439-446, 1999.

[118] Grabler Klaus, Zins Andreas, *Vacation Trip Decision Styles as Basis for an Automated Recommendation System: Lessons from Observational Studies*, ENTER 2002 Conference, Innsbruck, Austria, pp. 458-469, 2002.

[119] Graps Amara, *An Introduction to Wavelets*, IEEE Computational Science & Engineering, Volume 2, Number 2, pp. 50-61, 1995.

[120] Gribble Steven, Halevy Alon, Ives Zachary, Rodrig Maya, Suciu Dan, *What Can Database Do for Peer-to-Peer?*, 4th International Workshop on the Web and Databases, Santa Barbara, California, USA, WebDB 2001, pp. 31-36, 2001.

[121] Halevy Alon, *Data Integration: A Status Report*, 10th BTW Conference, Leipzig, pp. 24-29, 2003.

[122] Halevy Alon, *Structures, Semantics and Statistics*, 13th International Conference on Very Large Data Bases, Toronto, Canada, Morgan Kaufmann, VLDB 2004, pp. 4-6, 2004.

[123] Halevy Alon, Ives Zachary, Madhavan Jayant, Mork Peter, Suciu Dan, Tatarinov Igor, *The Piazza Peer Data Management System*, IEEE Transactions on Knowledge and Data Engineering, Volume 16, Number 7, pp. 787-798, 2004.

[124] Halevy Alon, Ives Zachary, Mork Peter, Tatarinov Igor, *Piazza: Data management infrastructure for semantic web applications*, 12th International World Wide Web Conference, WWW2003, Budapest, Hungary, pp. 556-567, 2003.

[125] Han Eui-Hong, Boley Daniel, Gini Maria, Gross Robert, Hastings Kyle, Karypis George, Kumar Vipin, Mobasher Bamshad, Moore Jerome, *WebACE: A Web Agent for Document Categorization and Exploration*, 2nd International Conference on Autonomous Agents (Agents 1998), Katia Sycara and Michael Wooldridge (Eds.), Minneapolis, USA, pp. 408-415, 1998.

[126] Han Eui-Hong, Karypis George, Kumar Vipin, Mobasher Bamshad, *Clustering Based on Association Rule Hypergraphs*, Workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD 97), held together with ACM

SIGMOD International Conference on Management of Data (ACM SIGMOD 97), Tucson, USA, 1997.

[127] Han Jiawei, Kamber Micheline, *Data Mining: Concepts and Techniques*, Morgan Kaufmann Publishers, 2001.

[128] Hasenfratz Jean-Marc, Damez Cyrille, Sillion François, Drettakis George, *A Practical Analysis of Clustering Strategies for Hierarchical Radiosity*, Computer Graphics Forum, Volume 18, Number 3, pp. 221-232, 1999.

[129] Hearst Marti, Pedersen Jan, *Re-examining the Cluster Hypothesis: Scatter/Gather on Retrieval Results*, 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 1996), Zurich, Switzerland, pp. 76-84, 1996.

[130] Hellerstein Joseph, Naughton Jeffrey, Pfeffer Avi, *Generalized Search Trees for Database Systems*, 21th International Conference on Very Large Data Bases (VLDB 1995), Zurich, Switzerland, pp. 562-573, 1995.

[131] Herlocker Jonathan, Konstan Joseph, Terveen Loren, Riedl John, *Evaluating collaborative filtering recommender systems*, ACM Transactions on Information Systems (TOIS), Volume 22, Number 1, pp. 5-53, 2004.

[132] Herlocker Jonathan, Konstan Joseph, Riedl John, *Explaining collaborative filtering recommendations*, ACM 2000 Conference on Computer supported cooperative work (CSCW 2000), Philadelphia, Pennsylvania, United States, pp. 241-250, 2000.

[133] Howland Peg, Park Haesun, *Cluster-Preserving Dimension Reduction Methods for Efficient Classification of Text Data*, In Survey of Text Mining: Clustering, Classification, and Retrieval, Michael Berry (Ed.), Springer-Verlag, pp. 3-24, 2003.

[134] Hung Chihli, Wermter Stefan, *A Self-Organising Hybrid Model for Dynamic Text Clustering*, Proceedings of the The Twenty-third SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence, Cambridge, UK, 2003.

[135] Jagadish HV, *Linear Clustering of Objects with Multiple Atributes*, ACM SIGMOD International Conference on Management of Data, Atlantic City, USA, pp. 332-342, 1990.

[136] Kantrowitz Mark, Mohit Behrang, Mittal Vibhu, *Stemming and its effects on TFIDF ranking*, 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2000), Athens, Greece, pp. 357-359, 2000.

[137] Karypis George, *Evaluation of Item-Based Top-N Recommendation Algorithms*, Proceedings of the 2001 ACM CIKM International Conference on Information and Knowledge Management, Atlanta, Georgia, USA, pp. 247-254, 2001.

[138] Kaski Samuel, Kangas Jari, Kohonen Teuvo, *Bibliography of Self-Organizing Map (SOM) Papers: 1981-1997*, Neural Computing Surveys, 1, pp. 102-350, 1998.

[139] Kephart Jeffrey, Hanson James, Levine David, Grosof Benjamin, Sairamesh Jakka, Segal Richard, White Steve, *Dynamics of an Information-Filtering*

Economy, 2nd International Workshop on Cooperative Information Agents II, Learning, Mobility and Electronic Commerce for Information Discovery on the Internet (CIA1998) , Paris, France, pp. 160-171, 1998.

[140] Khrouf Kais, Soule-Dupuy Chantal, *A Textual Warehouse Approach: a Web Data Repository*, In Intelligent Agents for Data Mining and Information Retrieval, Idea Group Publishing, pp. 101-124, 2004.

[141] Klein Peter, Reiterer Harald, Mueller Frank, Limbach Tobias, *Metadata Visualisation with VisMeB*, 7th International Conference on Information Visualization (IV 2003), London, pp. 600-605, 2003.

[142] Klose Aljoscha, Nürnberger Andreas, Kruse Rudolf, Hartmann Gerd, Richards Michael, *Interactive Text Retrieval Based on Document Similarities*, Physics and Chemistry of the Earth, Part A: Solid Earth and Geodesy, Volume 25, Number 8, pp. 649-654, 2000.

[143] Kim Jin Sung, *Customized Recommendation Mechanism Based on Web Data Mining and Case-Based Reasoning*, In Intelligent Agents for Data Mining and Information Retrieval, Idea Group Publishing, pp. 47-64, 2004.

[144] Knorr Edwin, Ng Raymond, *Algorithms for Mining Distance-Based Outliers in Large Datasets*, 24th International Conference on Very Large Data Bases (VLDB 1998), New York City, USA, pp. 392-403, 1998.

[145] Kobayashi Mei, Aono Masaki, *Vector space models for search and cluster mining*, In Survey of Text Mining: Clustering, Classification, and Retrieval, Michael Berry (Ed.), Springer-Verlag, pp. 103-122, 2003.

[146] Kobsa Alfred, *Generic User Modeling Systems*, User Modeling and User-Adapted Interaction, Volume 11, Number 1-2, pp. 49-63, 2001.

[147] Kobsa Alfred, Koenemann Jürgen, Pohl Wolfgang, *Personalised hypermedia presentation techniques for improving online customer relationships*, The Knowledge Engineering Review, Cambridge University Press, United Kingdom, Volume 16, Number 2, pp. 111-155, 2001.

[148] Kohonen Teuvo, Kaski Samuel, Lagus Krista, Salojärvi Jarkko, Honkela Jukka, Paatero Vesa, Saarela Antti, *Self organization of massive document collection*, IEEE Transactions on Neural Networks, Volume 11, Number 3, pp. 574-585, May 2000.

[149] Koller Daphne, Sahami Mehran, *Toward Optimal Feature Selection*, 13th International Conference on Machine Learning (ICML 1996), Bari, Italy, pp. 284-292, 1996.

[150] Konstan Joseph, Miller Bradley, Maltz David, Herlocker Jonathan, Gordon Lee, Riedl John, *GroupLens: Applying Collaborative Filtering to Usenet News*, Communications of the ACM (CACM), Volume 40, Number 3, pp. 77-87, 1997.

[151] Kontostathis April, Galitsky Leon, Pottenger William, Roy Soma, Phelps Daniel, *A Survey of Emerging Trend Detection in Textual Data Mining*, In Survey of Text Mining: Clustering, Classification, and Retrieval, Michael Berry (Ed.),
Springer-Verlag, pp. 185-224, 2003.

[152] Kossmann Donald, Ramsak Frank, Rost Steffen, *Shooting Stars in the Sky: An Online Algorithm for Skyline Queries*, Proceedings of the 28th VLDB Conference, Hong Kong, China, pp. 275-286, 2002.

[153] Kraaij Wessel, Pohlmann Renee, *Evaluation of a Dutch Stemming Algorithm*, The New Review of Document & Text Management, Jennifer Rowley (Ed.), Volume 1, Taylor Graham Publishing, London, pp. 25-43, 1995.

[154] Kummamuru Krishna, Lotlikar Rohit, Roy Shourya, Singal Karan, Krishnapuram Raghu, *A Hierarchical Monothetic Document Clustering Algorithm for Summarization and Browsing Search Results*, 13th International conference on World Wide Web (WWW 2004) New York, USA, pp. 658-665, 2004.

[155] Kuropka Dominik, Serries Thoams, *Personal Information Agent*, Informatik Jahrestagung 2001, pp. 940-946, 2001.

[156] Kurumatani Koichi, *Social Coordination with Architecture for Ubiquitous Agents – CONSORTS*, In Intelligent Agents for Data Mining and Information Retrieval, Idea Group Publishing, pp. 154-163, 2004.

[157] Laerhoven Kristof, Aidoo Kofi Asante, *Teaching Context to Applications*, Personal and Ubiquitous Computing, Volume 5, Number 1, pp. 46-49, 2001.

[158] Lai Wei, Huang Maolin, Zhang Kang, *Generating and Adjusting Web Sub-Graph Displays for Web Navigation*, In Intelligent Agents for Data Mining and Information Retrieval, Idea Group Publishing, pp. 241-253, 2004.

[159] Lam Shyong, Riedl John, *Shilling Recommender Systems for Fun and Profit*, 13th international conference on World Wide Web (WWW2004), New York, pp. 393-402, 2004.

[160] Lang Ken, *NewsWeeder: Learning to Filter Netnews*, 12th International Conference on Machine Learning (ICML 1995), Tahoe City, USA, pp. 331-339, 1995.

[161] Langdon William, *Natural Language Text Classification and Filtering with Trigrams and Evolutionary Nearest Neighbour Classifiers*, CWI Report, National research institute for Mathematics and Computer Science in the Netherlands, 2000.

[162] Lawrence Steve, Bollacker Kurt, Giles Lee, *Indexing and Retrieval of Scientific Literature*, ACM CIKM International Conference on Information and Knowledge Management (CIKM 1999), Kansas City, USA, pp. 139-146, 1999.

[163] Lawrence Steve, Giles Lee, Bollacker Kurt, *Autonomous Citation Matching*, 3rd Annual Conference on Autonomous Agents (Agents 1999), Seattle, USA, pp. 392-393, 1999.

[164] Lawrence Steve, Giles Lee, Bollacker Kurt, *Digital Libraries and Autonomous Citation Indexing, IEEE Computer*, Volume 32, Number 6, pp. 67-71, 1999.

[165] Leake David, Sooriamurthi Raja, *When Two Case Bases are Better Than One: Exploiting Multiple Case Bases*, 4th International Conference on Case-Based Reasoning (ICCBR 01), Berlin, Springer-Verlag, pp. 321-335, 2001.

[166] Leake David, Sooriamurthi Raja, *Automatically Selecting Strategies for Multi-Case-Base Reasoning*, 6th European Conference (ECCBR 2002), Susan Craw, Alun Preece (Eds.), Aberdeen, Scotland, pp. 204-233, 2002.

[167] Lenzerini Maurizio, *Principles of P2P Data Integration*, 3rd International Workshop on Data Integration over the Web (DIWeb 2004), Riga, Latvia, pp. 7-21, 2004.

[168] Lieberman Henry, *Autonomous Interface Agents*, Human Factors in Computing Systems (CHI 97), Atlanta, Georgia, pp. 67-74, 1997.

[169] Lieberman Henry, Fry Christopher, Weitzman Louis, *Exploring the Web with reconnaissance agents*, Communications of the ACM, Volume 44, Number 8, pp. 69-75, 2001.

[170] Lieberman Henry, *Letizia: An Agent That Assists Web Browsing*, International Joint Conference on Artificial Intelligence, Montreal, August 1995.

[171] Lieberman Henry, Neil Van Dyke, Vivacqua Adriana, *Let's Browse: A Collaborative Browsing Agent*, Knowledge-Based Systems, Volume 12, Number 8, pp. 427–431, 1999.

[172] Lieberman Henry, Neil Van Dyke, Vivacqua Adriana, *Let's Browse: A Collaborative Web Browsing Agent*, International Conference on Intelligent User Interfaces (IUI 1999), Los Angeles, USA, pp. 65-68, 1999.

[173] Madhavan Jayant, Halevy Alon, *Composing Mappings Among Data Sources*, 29th International Conference on Very Large Data Bases (VLDB 2003), Berlin, Germany, pp. 572-583, 2003.

[174] Madsen Kaj, Zertchaninov Serguei, *A New Branch-and-Bound Method for Global Optimization*, IMM-Report-1998-05, 1998.

[175] Mann Thomas, Reiterer Harald, *Case Study: A Combined Visualization Approach for WWW-Search Results*, IEEE Symposium on Information Visualization (InfoVis'99), San Francisco, USA, pp. 59-62, 1999.

[176] Mann Thomas, Reiterer Harald, *Evaluation of Different Visualizations of Web Search Results*, 11th International Workshop on Database and Expert Systems Applications (DEXA 2000), Greenwich, London, UK, pp. 586-590, 2000.

[177] Mark Ackerman, Billsus Daniel, Gaffney Scott, Hettich Seth, Khoo Gordon, Joon Kim Dong, Klefstad Raymond, Lowe Charles, Ludeman Alexius, Muramatsu Jack, Omori Kazuo, Pazzani Michael, Semler Douglas, Starr Brian, Yap Paul, *Learning Probabilistic User Profiles: Applications for Finding Interesting Web Sites, Notifying Users of Relevant Changes to Web Pages, and Locating Grant Opportunities*, AI Magazine, Volume 18, Number 2, pp. 47-56, 1997.

[178] Mark Ackerman, Starr Brian, Pazzani Michael, *The Do-I-Care Agent: Effective Social Discovery and Filtering on the Web*, Conference on Computer-Assisted Searching on the Internet (RIAO 97), Montreal, Quebec, Canada, 1997.

[179] Mayfield James, McNamee Paul, *Indexing Using Both N-Grams and Words*, 7th Text Retrieval Conference (TREC-7), National Institute of Standards and Technology (NIST), Gaithersburg, Maryland, pp. 361-365, 1998.

[180] Meinköhn Jens, Albayrak Sahin, *Future IN-Platforms with Agent Technology*, In Intelligent Agents in Telecommunications Applications-Basics, Tools, Languages and Applications, Albayrak Sahin (Ed.), IOS Press, Netherlands, pp. 80-93, 1998.

[181] Michalewicz Zbigniew, Fogel David, *How to Solve It: Modern Heuristics*, Springer-Verlag New York, Inc., New York, NY, 2000.

[182] Miikulainen Risto, *Script Recognition with Hierarchical Feature Maps*, Connection Science 2, pp. 83-101, 1990.

[183] Miller Brad, Goldberg David, *Genetic Algorithms, Selection Schemes, and the Varying Effects of Noise*, Evolutionary Computation, Volume 4, Number 2, pp. 113-131, 1996.

[184] Miller Bradley, Riedl John, Konstan Joseph, *Experiences with GroupLens: Making Usenet useful again*, 1997 Usenix Winter Technical Conference, January 1997.

[185] McSherry David, *Diversity-Conscious Retrieval*, 6th European Conference on Advances in Case-Based Reasoning (ECCBR 2002), Aberdeen, Scotland, UK, pp. 219-233, 2002.

[186] Moffat Alistair, Zobel Justin, *Self-Indexing Inverted Files for Fast Text Retrieval*, ACM Transactions on Information Systems (TOIS), Volume 14, Number 4, pp. 349-379, 1996.

[187] Mohammadian Masoud, Jentzsch Ric, *Computational Intelligence Techniques Driven Intelligent Agents for Web Data Mining and Information Retrieval*, In Intelligent Agents for Data Mining and Information Retrieval, Idea Group Publishing, pp. 15-29, 2004.

[188] Moukas Alexandros, *Amalthaea: Information Discovery and Filtering using a Multi agent Evolving Ecosystem*, In Practical Application of Intelligent Agents & Multi-Agent Technology, London 1998.

[189] Mussler Gabriela, Reiterer Harald, Mann Thomas, *INSYDER - Information Retrieval Aspects of a Business Intelligence System*, 7th International Symposium on Informatics (ISI 2000), Darmstadt, pp. 127-144, 2000.

[190] Nagano Shinichi, Tahara Yasuyuki, Hasegawa Tetsuo, Ohsuga Akihiko, *Development of Agent-Based Electronic Catalog Retrieval System*, In Intelligent Agents for Data Mining and Information Retrieval, Idea Group Publishing, pp. 188-195, 2004.

[191] Navarro Gonzalo, Baeza-Yates Ricardo, Arcoverde Joao Marcelo Azevedo, *Matchsimile: a Flexible Approximate Matching Tool for Searching Proper Name*, Journal of the American Society for Information Science and Technology (JASIST), Volume 54, Number 1, pp. 3-15, 2003.

[192] Nichols David, *Implicit Rating and Filtering*, 5th DELOS Workshop on Filtering and Collaborative Filtering (ERCIM), Budapest, Hungary, pp. 31-36, 1997.

[193] Nielsen Jakob, *Personalization is Over-Rated*, Online at http://www.useit.com/alertbox/981004.html

[194] Ng Wee Siong, Ooi Beng Chin, Tan Kian-Lee, Zhou Aoying, *PeerDB: A P2P-based System for Distributed Data Sharing*, 19th International Conference on Data Engineering (ICDE 2003), Bangalore, India, pp. 633-644, 2003.

[195] Oard Douglas, Kim Jinmook, *Implicit Feedback for Recommender Systems*, AAAI Workshop on Recommender Systems, Madison, Wisconsin, pp. 81-83, 1998.

[196] Oard Douglas, Marchionini Gary, *A Conceptual Framework for Text Filtering*, University of Maryland, College Park, MD 20742, May, 1996.

[197] Oja Merja, Kaski Samuel, Kohonen Teuvo, *Bibliography of Self-Organizing Map (SOM) Papers: 1998-2001 Addendum*, Neural Computing Surveys, 3, pp. 1-156, 2003.

[198] Osborne Hugh, Bridge Derek, *Similarity Metrics: A Formal Unification of Cardinal and Non-Cardinal Similarity Measures*, 2nd International Conference on Case-based Reasoning, Providence, Rhode Island, 1997.

[199] Papadias Dimitris, Mantzourogiannis Marios, Kalnis Panos, Mamoulis Nikos, Ahmad Ishfaq, *Content-Based Retrieval Using Heuristic Search*, 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 1999), Berkeley, USA, pp. 168-175, 1999.

[200] Papaioannou Thanasis, Stamoulis George, *Effective Use of Reputation in Peer-to-Peer Environments*. In Proceeding of IEEE/ACM CCGRID 2004 (Workshop on Global P2P Computing), April 2004.

[201] Pazzani Michael, Billsus Daniel, *Adaptive Web Site Agents*, 3rd Annual Conference on Autonomous Agents (Agents 1999), Seattle, USA, pp. 394-395, 1999.

[202] Pazzani Michael, Billsus Daniel, *Adaptive Web Site Agents*, Autonomous Agents and Multi-Agent Systems, Volume 5, Number 2, pp. 205-218, 2002.

[203] Pazzani Michael, Billsus Daniel, *Evaluating adaptive web site agents*, 22nd International Conference on Research and Development in Information Retrieval, Workshop on Recommender Systems: Algorithms and Evaluation, Berkeley, California, USA, 1999.

[204] Pazzani Michael, Billsus Daniel, *Learning and Revising User Profiles: The Identification of Interesting Web Sites*, Machine Learning, Volume 27, Number 3, pp. 313-331, 1997.

[205] Pazzani Michael, Muramatsu Jack, Billsus Daniel, *Syskill & Webert: Identifying Interesting Web Sites*, 13th National Conference on Artificial Intelligence and 8th Innovative Applications of Artificial Intelligence Conference (AAAI/ IAAI 96) Portland, USA, pp. 54-61, 1996.

[206] Porter Martin, *An algorithm for suffix stripping*, Program (Automated Library and Information Systems), Volume 14, Number 3, pp. 130-137, 1980.

[207] Quah Jon, Chen Youming, Leow Winnie, *Networking E-learning Hosts Using Mobile Agent*, In Intelligent Agents for Data Mining and Information Retrieval, Idea Group Publishing, pp. 263-294, 2004.

[208] Radig Bernd, *MINT - Management Information from the Internet*, Bavarian Research Centre for Knowledge-Based Systems

[209] Rana Omer, Stout Kate, *What is Scalability in Multi-Agent Systems*, 4th International conference on Autonomous agents (Agents 2000), Barcelona, Spain, pp. 56-63, 2000.

[210] Reiterer Harald, Mussler Gabriela, Mann Thomas, *A visual information seeking system for Web search*, Mensch & Computer 2001, Bad Honnef, Germany, 2001.

[211] Reiterer Harald, Mussler Gabriela, Mann Thomas, Handschuh Siegfried, *INSYDER - an information assistant for business intelligence*, 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2000), Athens, Greece, pp. 112-119, 2000.

[212] Reiterer Harald, Tullius Gabriela, Mann Thomas, *INSYDER: a content-based visual-information-seeking system for the Web*, International Journal on Digital Libraries, Volume 5, Number 1, pp. 25-41, 2005.

[213] Resnick Paul, Iacovou Neophytos, Suchak Mitesh, Bergstrom Peter, Riedl John, *GroupLens: An open architecture for collaborative filtering of net news*, ACM Conference on Computer-Supported Cooperative Work (CSCW 1994), Chapel Hill, USA, pp. 175-186, 1994.

[214] Resnick Paul, Varian Hal, *Recommender Systems*, Communications of the ACM (CACM), Volume 40, Number 3, pp. 56-58, 1997.

[215] Ricci Francesco, Arslan Bora, Mirzadeh Nader, Venturini Adriano, *ITR: A Case-Based Travel Advisory System*, Proceeding of ECCBR, 6th European Conference on Case Based Reasoning, Aberdeen, Scotland, pp. 613-627, 2002.

[216] Ricci Francesco, Blaas Dennis, Mirzadeh Nader, Venturini Adriano, Werthner Hannes, *Intelligent Query Management for Travel Products Selection*, ENTER 2002 Conference, Innsbruck, 22-25 January, 2002.

[217] Rivero Daniel, Rabunal Juan, Dorado Julian, Pazos Alejandro, Pedreira Nieves, *Extracting Knowledge from Databases and Artificial Neural Networks with Genetic Programming: Iris Flower Classification Problem*, In Intelligent Agents for Data Mining and Information Retrieval, Idea Group Publishing, pp. 137-153, 2004.

[218] Roussinov Dmitri, Chen Hsinchun, *A scalable self-organizing map algorithm for textual classification: a neural network approach to thesaurus generation*, CC-AI, The Journal for the Integrated Study of Artificial Intelligence, Cognitive Science and Applied Epistemology, Volume 15, Numbers 1-2, pp. 81-111, 1998.

[219] Ruzzi Marco, *Data Integration: State of the art, new issues and research plan*, University of Rome, Italy, October 2004.

[220] Sakagami Hidekazu, Kamba Tomonari, *Learning Personal Preferences on Online Newspaper Articles from User Behaviors*, Computer Networks, Volume 29, Numbers 8-13, pp. 1447-1455, 1997.

[221] Sarwar Badrul, Karypis George, Konstan Joseph, Riedl John, *Analysis of recommendation algorithms for e-commerce*, Proceedings of the 2nd ACM Conference on Electronic Commerce (EC-00), Minneapolis, USA, pp. 158-167, 2000.

[222] Sarwar Badrul, Konstan Joseph, Borchers Al, Herlocker Jonathan, Miller Bradley, Riedl John, *Using Filtering Agents to Improve Prediction Quality in the GroupLens Research Collaborative Filtering System*, ACM 1998 Conference on Computer Supported Cooperative Work (CSCW 1998), Seattle, USA, pp. 345-354, 1998.

[223] Saurabh Singhal, Eui-Hong Han, Vipin Kumar, *Web Search Engine Technologies: A Survey*, University of Minnesota, Minneapolis, USA.

[224] Schafer Ben, Konstan Joseph, Riedl John, *E-Commerce Recommendation Applications*, Journal of Data Mining and Knowledge Discovery, pp. 115-153, January 2001.

[225] Senellart Pierre, Blondel Vincent, *Automatic discovery of similar words*, In Survey of Text Mining: Clustering, Classification, and Retrieval, Michael Berry (Ed.), Springer-Verlag, pp. 25-44, 2003.

[226] Seo Jinwook, Shneiderman Ben, *Understanding Hierarchical Clustering Results by Interactive Exploration of Dendrograms: A Case Study with Genomic Microarray Data*, Techical Report, University of Maryland Computer Science Department, 2001.

[227] Sesseler Ralf, Albayrak Sahin, *Agent-Based Marketplaces for Electronic Commerce*, International Conference on Artificial Intelligence (IC-AI 2001), Las Vegas, USA, 2001.

[228] Sesseler Ralf, Albayrak Sahin, JIAC IV – *An Open, Scalable Agent Architecture for Telecommunications Applications*, 1st International NAISO Congress on Autonomous Intelligent Systems (ICAIS 2002), Geelong, Australia, 2002.

[229] Sesseler Ralf, Albayrak Sahin, *Service-Ware Framework for Developing 3G Mobile Services*, 16th International Symposium on Computer and Information Sciences (ISCIS XVI), Berlin, Germany, 2001.

[230] Sesseler Ralf, *Building Agents for Service Provisioning out of Components*, 5th International conference on Autonomous agents (Agents 2001), Montreal, Canada, pp. 218-219, 2001.

[231] Sesseler Ralf, Keiblinger Alexander, Varone Nicolas, *Software Agent Technology in Mobile Service Environments*, Workshop on M-Services, Concepts, Approaches and Tools held at 13th International Symposium on Methodologies for Intelligent Systems (ISMIS 2002), Lyon, France, 2002.

[232] Shardanand Upendra, Maes Pattie, *Social Information Filtering: Algorithms for Automating "Word of Mouth"*, Conference on Human Factors in Computing Systems (CHI 1995), Denver, USA, pp. 210-217, 1995.

[233] Shardanand Upendra, *Social Information Filtering for Music Recommendation*, M.S. Thesis, MassachusettsInstitute of Technology, 1994.

[234] Sheikholeslami Gholamhosein, Chatterjee Surojit, Zhang Aidong, *WaveCluster: A Multi-Resolution Clustering Approach for Very Large Spatial Databases*, The Proceedings of the 24th International Conference on Very Large Data Bases (VLDB'98), pp. 428-439, New York City, August 1998.

[235] Sheth Beerud Dilip, *A Learning Approach to Personalized Information Filtering*, M.S. Thesis, MIT- EECS dept, USA, 1994.

[236] Shi Hong, Zhang Ji-Fu, *An Algorithm of Pattern Match being Fit for Mining Association Rules*, In Intelligent Agents for Data Mining and Information Retrieval, Idea Group Publishing, pp. 254-262, 2004.

[237] Siaw Weng Ngu Daniel, Wu Xindong, *SiteHelper: A Localized Agent That Helps Incremental Exploration of the World Wide Web*, Computer Networks and ISDN Systems, Volume 29, Numbers 8-13, pp. 1249-1255, 1997.

[238] Smyth Barry, Cotter Paul, *Enabling Technologies: A Personalized Television Listings Service*, Communications of the ACM (CACM), Volume 43, Number 8, pp. 107-111, 2000.

[239] Smyth Barry, Cotter Paul, Hare Gregory, *Let's Get Personal: Personalised TV Listings on the Web*, 9th Irish Conference on Artificial Intelligence and Cognitive Science, Dublin, Ireland, 1998.

[240] Smyth Barry, Cotter Paul, *Personalized Electronic Program Guides for Digital TV*, AI Magazine Volume 22, Number 2, pp. 89-98, 2001.

[241] Smyth Barry, Cotter Paul, *Surfing the Digital Wave*, 3rd International Conference on Case-Based Reasoning and Development (ICCBR 1999), Monastery, Germany, pp. 561-571, 1999.

[242] Smyth Barry, Cotter Paul, *The Sky's the Limit: A personalised TV Listings Service for the Digital TV Age*, 19th SGES International Conference on Knowledge-Based and Applied Artificial Intelligence (ES 99), Cambridge, UK, 1999.

[243] Smyth Barry, McClave Paul, *Similarity vs. Diversity*, Case-Based Reasoning Research and Development, LNAI, Volume 2080, Springer-Verlag, Berlin Heidelberg, pp. 347-361, 2001.

[244] Sosna Dieter, *Document Version Management Using an Adapted Segment Tree*, Technical Report, Leipzig University, 1997.

[245] Spears, William, *Simulated Annealing for Hard Satisfiability Problems*, In Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, David S. Johnson and Michael A. Trick (eds.), DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Volume 26, American Mathematical Society, pp. 533-558, 1996.

[246] Starr Brian, Ackerman Mark, Pazzani Michael, *Do-I-Care: A Collaborative Web Agent*, ACM Conference on Human Factors in Computing Systems (CHI 96), pp. 273-274, 1996.

[247] Steinbach Michael, Karypis George, Kumar Vipin, *A comparison of document clustering technique*, Technical Report, Department of Computer Science and Engineering, University of Minnesota.

[248] Sutton Richard, Barto Andrew, *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, 1998.

[249] Tao Guo, Michalewicz Zbigniew, *Inver-over Operator for the TSP*, 5th International Conference on Parallel Problem Solving from Nature (PPSN 1998), Amsterdam, The Netherlands, pp. 803-812, 1998.

[250] Tauritz Daniel, *Adaptive Information Filtering as a Means to overcome Information Overload*, M.S. Thesis, Leiden University, 1996.

[251] Tauritz Daniel, *Adaptive Information Filtering: Concepts and Algorithms*, Ph.D. Thesis, Leiden University, 2002.

[252] Thijssen Baas, *Adaptive Genetic Algorithms with Multiple Subpopulations and Multiple Parents*, M.S. Thesis, Leiden University, 1997.

[253] Thorsten Joachims, Dayne Freitag, Tom Mitchell, *WebWatcher: A Tour Guide for the World Wide Web*, 15th International Joint Conference on Artificial Intelligence (IJCAI), Nagoya, Japan, pp. 770-777, 1997.

[254] Thorsten Joachims, Mitchell Tom, Dayne Freitag, Armstrong Robert, *WebWatcher: Machine Learning and Hypertext*, 7th Workshop on Machine Learning, Dortmund, Germany, 1995.

[255] Toivonen Hannu, *Sampling Large Databases for Association Rules*, 22nd International Conference on Very Large Data Bases (VLDB 1996), Bombay, India, pp. 134-145, 1996.

[256] Turney Peter, *Learning Algorithms for Keyphrase Extraction*, Information Retrieval, Volume 2, Number 4, pp. 303-336, 2000.

[257] Turner Phillip, Jennings Nicholas, *Improving the Scalability of Multi-Agent Systems*, International Workshop on Infrastructure for Multi-Agent Systems held together with International Conference on Autonomous Agents (Agents 2000), Thomas Wagner, Omer Rana (Eds.), Barcelona, Spain, pp. 246-262, 2000.

[258] Ungar Lyle, Foster Dean, *Clustering Methods for Collaborative Filtering*, Workshop on Recommender Systems at the 15th National Conference on Artificial Intelligence, 1998.

[259] Vrettos Spyros, Stafylopatis Andreas, *Taxonomy Based Fuzzy Filtering of Search Results*, In Intelligent Agents for Data Mining and Information Retrieval, Idea Group Publishing, pp. 226-240, 2004.

[260] Wang Peiling, Berry Michael, Bownas Jennifer, *Trend and Behaviour Detection from Web Queries*, In Survey of Text Mining: Clustering, Classification, and Retrieval, Michael Berry (Ed.), Springer-Verlag, pp. 173-183, 2003.

[261] Wasfi Ahmad, *Collecting User Access Patterns for Building User Profiles and Collaborative Filtering*, International Conference on Intelligent User Interfaces, Los Angeles, CA, USA, pp. 57-64, 1999.

[262] Wei Yan Zheng, Moreau Luc, Jennings Nicholas, *Market-Based Recommender Systems: Learning Users' Interests by Quality Classification*, 6th International Workshop on Agent-Oriented Information Systems (AOIS 2004), New York, USA, pp. 119-133, 2004.

[263] Wexelblat Alan, *History-based Tools for Navigation*, 32nd Annual Hawaii International Conference on System Sciences (HICSS 1999), Maui, Hawaii, 1999.

[264] Wexelblat Alan, *History-Rich Tools for Social Navigation*, Conference summary on Human factors in computing systems (CHI 98) Los Angeles, USA, pp. 359-360, 1998.

[265] Wexelblat Alan, Maes Pattie, *Footprints: History-Rich Tools for Information Foraging*, Conference on Human Factors in Computing Systems: The CHI is the Limit (CHI 1999), Pittsburgh, USA, pp. 270-277, 1999.

[266] Wexelblat Alan, Maes Pattie, *Footprints: History-Rich Web Browsing*, 5th Conference of Computer-Assisted Information Retrieval (RIAO), Montreal, Canada, pp. 75-84, 1997.

[267] White Ryen, Jose Joemon, Ruthven Ian, *Comparing Explicit and Implicit Feedback Techniques for Web Retrieval: TREC-10 Interactive Track Report*, 10th Text Retrieval Conference (TREC 2001), National Institute of Standards and Technology (NIST), Gaithersburg, Maryland.

[268] Wilson Randall, Martinez Tony, *Improved Heterogeneous Distance Functions*, Journal of Artificial Intelligence Research, Volume 6, pp. 1-34, 1997.

[269] Winiwarter Werner, *PEA - A Personal Email Assistant with Evolutionary Adaptation*, International Journal of Information Technology, Volume 5, Number 1, 1999.

[270] Wolfram Stephen, *The Mathematica Book,* Wolfram Media, Cambridge University Press, Fourth Edition, 1999.

[271] Wolpert David, William Macrea, *No Free Lunch Theorems for Optimization*, IEEE Transactions on Evolutionary Computation, Volume 1, Number 1, pp. 67-82, 1997.

[272] Wu Xindong, *The HCV Induction Algorithm*, ACM Conference on Computer Science (CSC 1993), Indianapolis, USA, pp. 168-175, 1993.

[273] Wu Wensheng, Yu Clement, Meng Weiyi, Liu King-Lup, *Text Database Selection for Longer Queries*, Technical Report, University of Illinois, 2001.

[274] Wu Wensheng, Yu Clement, Meng Weiyi, *Database Selection for Longer Queries*, In Proceedings of the 2004 Meeting of International Federation of Classification Societies, Chicago, Illinois, 2004.

[275] Yang Beverly, Garcia-Molina Hector, *Comparing Hybrid Peer-to-Peer Systems*, 27th International Conference on Very Large Data Bases (VLDB 2001), Rom, Italy, pp. 561-570, 2001.

[276] Yang Hui, Zhang Minjie, *Potential Cases, Database Types, and Selection Methodologies for Searching Distributed Text Databases*, In Intelligent Agents for Data Mining and Information Retrieval, Idea Group Publishing, pp. 1-14, 2004.

[277] Yang Yiming, Liu Xin, *A Re-Examination of Text Categorization Methods*, 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 1999), Berkeley, USA, pp. 42-49, 1999.

[278] Young-Woo Seo, Byoung-Tak Zhang, *A Reinforcement Learning Agent for Personalized Information Filtering*, 5th International conference on intelligent user interfaces, New Orleans, Louisiana, USA, pp. 248-251, 2000.

[279] Young-Woo Seo, Byoung-Tak Zhang, *Learning user's preferences by analyzing Web-browsing behaviours*, 4th International conference on Autonomous agents (Agents 2000), Barcelona, Spain, pp. 381-387, 2000.

[280] Yu Kai, Wen Zhong, Xu Xiaowei, Ester Martin, *Feature Weighting and Instance Selection for Collaborative Filtering*, 2nd International Workshop on Management of Information on the Web, Web Data and Text Mining (MIW 01), held together with 12th International Conference on Database and Expert Systems Applications (DEXA 01), Munich, Germany, pp. 285-290, 2001.

[281] Zaihrayeu Ilya, Giunchiglia Fausto, *Coordinating Mobile Databases: A System Demonstration*, 1st Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous 2004), 2004.

[282] Zamir Oren, *Clustering Web Documents: A Phrase-Based Method for Grouping Search Engine Results*, Ph.D. Thesis, University of Washington, 1999.

[283] Zamir Oren, Etzioni Oren, *Grouper: A Dynamic Clustering Interface to Web Search Results*, Computer Networks, Volume 31, Numbers 11-16, pp. 1361-1374, 1999.

[284] Zamir Oren, Etzioni Oren, Madani Omid, Karp Richard, *Fast and Intuitive Clustering of Web Documents*, Knowledge Discovery and Data Mining (KDD 1997), pp. 287-290, 1997.

[285] Zamir Oren, Etzioni Oren, *Web Document Clustering: A Feasibility Demonstration*, 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 1998), Melbourne, Australia, pp. 46-54, 1998.

[286] Zhang Lingling, Li Jun, Pan Jingui, *An Intelligent Interface Agent for Web-Based Information Retrieval*, IEEE Workshop on Internet Applications (WIAPP 99), San Jose, USA, 1999.

[287] Zhu Shanfeng, Deng Xiaotie, Fang Qizhi, Zheng Weimin, *A Study on Web Searching: Overlap and Distance of the Search Engine Results*, In Intelligent Agents for Data Mining and Information Retrieval, Idea Group Publishing, pp. 208-225, 2004.

[288] Zobel Justin, Moffat Alistair, Sacks-Davis Ron, *An Efficient Indexing Technique for Full Text Databases*, 18th International Conference on Very Large Data Bases (VLDB 1992), Vancouver, Canada, pp. 352-362, 1992.

Curriculum Vitae

Personal Data

Name: Dragan Milosevic
Date of birth: May 6, 1975
Place of birth: Jagodina, Yugoslavia (YU)
Family Status: Married from November 8, 2002 to Dr. Violeta Simic-Milosevic
Address: Mainzer Str. 5, 10715 Berlin, Germany

Education

Primary: 1982-1990, Elementary School, Nis, YU (5.00/5.00)
Secondary: 1990-1994, Gymnasium, Nis, YU, (5.00/5.00)
Basic studies: 1994-1999, Faculty of Electronic Engineering, Nis, YU, (10.00/10.00)
Master studies: 1999-2001, Faculty of Electronic Engineering, Nis, YU, (10.00/10.00)
PhD studies: 2002-, Technical University Berlin, Germany

Awards

June 1990: The best pupil of Elementary School
November 1997: The best student of Faculty of Electronic Engineering in 1997
September 1999: The best diploma work at Faculty of Electronic Engineering in 1999
October 1999: The Silver Gain as the best student of University of Nis in 1998/99 class

Specific Knowledge

Foreign languages: English (Toefl score 577/650), German (passive)
Programming languages: Java, C, C++, Mathematica, Fortran, Pascal, Lisp
Research interests: Information retrieval, Data mining, Clustering techniques, Filtering strategies, Decision diagrams, Symbolic programming

Working Experience

October 1998 – January 1999: Assisting as a student in laboratory for data mining at Faculty of Electronic Engineering, University of Nis, Yugoslavia
July 1999 – August 1999: Practical training in Hydro Power Plant in Agrinion, Greece
October 2000 – December 2000, November 2001 – December 2001: Researching about Decision Diagrams under DAAD fellowship at University of Dortmund, Germany
April 2000 – October 2001: Assistant for Logical design and Programming techniques at Faculty of Electronic Engineering, University of Nis, Yugoslavia
July 2000 – June 2001: Part-time work on developing a script based language for testing C programs for embedded systems, RistanCase, Zurich, Switzerland
January 2002 – : Research assistant in DAI Labor, Technical University Berlin, Germany

Publication List

Journal Papers

[1] Albayrak Sahin, **Milosevic Dragan**, *Component Based Local Optimisation Filtering Agent*, International Journal of Lateral Computing (IJLC), Volume 1, Number 2, pp. 16-23, 2005.

[2] Albayrak Sahin, **Milosevic Dragan**, *Multi-Domain Strategy Coordination Approach for Optimal Resource Usage in Agent Based Filtering Framework*, International Journal of Web Intelligence and Agent Systems (WIAS), IOS Press, Amsterdam (to appear)

[3] Albayrak Sahin, **Milosevic Dragan**, *Multi-Domain Collaborative Exploration Mechanisms for Query Expansion in Agent Based Filtering Framework*, Special Issue on Intelligence in e-Application, Journal of E-commerce Research and Application (to appear)

[4] Albayrak Sahin, **Milosevic Dragan**, *Strategy Coordination Approach for Safe Learning about Novel Filtering Strategies in Multi Agent Framework*, Lecture Notes in Artificial Intelligence (LNAI 3930) , pp. 30-42, 2006.

[5] Djordjevic Lazar, **Milosevic Dragan**, Milovanovic Gradimir, Srivastava Hari, *Some Finite Summation Formulas Involving Multivariable Hypergeometric Polynomials*, Integral Transforms and Special Functions, Volume 14, Number 4, pp. 349–361, 2003.

Conference Papers

[6] Albayrak Sahin, **Milosevic Dragan**, *Collaborative Filtering Guided Exploration in Multi Agent Filtering Framework*, IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE-05), Hong Kong, China, pp. 760-767, 2005.

[7] Albayrak Sahin, **Milosevic Dragan**, *Component Based Local Optimisation Filtering Agent*, 1st World Congress on Lateral Computing (WCLC 04) in cooperation with IEEE and World Federation on Soft Computing (WFSC), Bangalore, India, 2004.

[8] Albayrak Sahin, **Milosevic Dragan**, *Cooperative Community Selection in Multi Agent Filtering Framework*, IEEE/WIC/ACM International Joint Conference on Intelligent Agent technology (IAT 05) and Web Intelligence (WI 05), Compiegne University of Technology, France, pp. 527-535, 2005.

[9] Albayrak Sahin, **Milosevic Dragan**, *Distributed Community Cooperation In Multi Agent Filtering Framework*, 7th International Conference on Enterprise Information Systems (ICEIS 05) in the cooperation with IEEE and ACM, Miami, Florida, USA, pp. 400-407, 2005.

[10] Albayrak Sahin, **Milosevic Dragan**, *Flexible Strategy Coordination Approach for Supporting Resource Awareness in Multi Agent Filtering Framework*, 4th

International Conference on Machine Learning and Cybernetics (ICMLC 2005) in cooperation with IEEE, Guangzhou, China, pp. 5615-5623, 2005.

[11] Albayrak Sahin, **Milosevic Dragan**, *Generic Intelligent Personal Information Agent*, International Conference on Advances in Internet, Processing, Systems, and Interdisciplinary Research (IPSI-2004S) in cooperation with Yu IEEE Computer Chapter, Serbia, 2004.

[12] Albayrak Sahin, **Milosevic Dragan**, *Response Time Aware Coordination in Multi Agent Filtering Framework*, 18th International FLAIRS Conference (FLAIRS 05) in the cooperation with The American Association of Artificial Intelligence (AAAI), Florida Artificial Intelligence Research Society, Clearwater Beach, Florida, USA, pp. 806-807, 2005.

[13] Albayrak Sahin, **Milosevic Dragan**, *Resource And Job Aware Coordination In Multi Agent Filtering Framework*, Artificial Intelligence And Applications (AIA 2005), 23rd IASTED International Multi-Conference on Applied Informatics, Innsbruck, Austria, pp. 539-544, 2005.

[14] Albayrak Sahin, **Milosevic Dragan**, *Resource Load Predictable Coordination in Multi Agent Filtering Framework*, International Conference on Advances in Intelligent Systems - Theory and Applications in the cooperation with IEEE Computer Society (AISTA 04), Luxembourg-Kirchberg, Luxembourg, 2004.

[15] Albayrak Sahin, **Milosevic Dragan**, *Self Improving Coordination in Multi Agent Filtering Framework*, IEEE/WIC/ACM International Joint Conference on Intelligent Agent technology (IAT 04) and Web Intelligence (WI 04), Beijing, China, pp. 239-245, 2004.

[16] Albayrak Sahin, **Milosevic Dragan**, *Situation-aware Coordination in Multi Agent Filtering Framework*, 19th International Symposium on Computer and Information Sciences (ISCIS 04) in cooperation with Lecture Notes in Computer Science by Springer-Verlag (LNCS), Antalya, Turkey, pp. 480-493, 2004.

[17] Albayrak Sahin, Muller Achim, Scheel Christian, **Milosevic Dragan**, *Combining Self-Organizing Map Algorithms for Robust and Scalable Intrusion Detection*, International Conference on Computational Intelligence for Modelling Control and Automation (CIMCA 2005), Vienna, Austria (to appear)

[18] Albayrak Sahin, Wollny Stefan, Varone Nicolas, Lommatzsch Andreas, **Milosevic Dragan**, *Agent Technology for Personalized Information Filtering: The PIA-System*, The 20th ACM Symposium on Applied Computing, Agents, Interactions, Mobility, and Systems (AIMS) track, Santa Fe, New Mexico, USA, pp. 54-59, 2005.

Workshop Papers

[19] **Milosevic Dragan**, Stankovic Radomir, Moraga Claudio, *Calculation of Dyadic Autocorrelation Through Decision Diagrams*, Workshop on Computational Intelligence and Informational Technologies, Nis, pp. 129-134, June 2001.

[20] **Milosevic Dragan**, Stankovic Radomir, Moraga Claudio, *Match Counting Driven DD Minimization*, Workshop on Computational Intelligence Theory and Applications, Nis, pp. 129-140, February 2001.

Technical Reports

[21] **Milosevic Dragan**, Stankovic Radomir, Moraga Claudio, *Approaching the Wiener-Khinchin Theorem Performance in the Computation of Dyadic Autocorrelation*, Technical Report, University of Dortmund, December 2001.

[22] **Milosevic Dragan**, Stankovic Radomir, Moraga Claudio, *Fast Reordering Strategies for BDDs*, Technical Report, University of Dortmund, December 2000.

Theses

[23] **Milosevic Dragan**, *Automatic Proving of one Type of Binomial Equations by Using Program System Mathematica*, Best Diploma Work Award, Faculty of Electrical Engineering, University of Nis, 1999.

[24] **Milosevic Dragan**, *Program Generator and Verifier of Symbolic Orthogonal Polynomials and Recursive Relations*, M.S. Thesis, Faculty of Electrical Engineering, University of Nis, 2001.

Acknowledgement

I should truthfully gratitude to many people, who have kindly supported me during my dissertation research. First of all I would like to thank my thesis adviser Prof. Dr. Sahin Albayrak, not only for providing these research opportunities and environment to me, but also for raising my interests in information filtering and guiding my academic career over more than four years.

Valuable perspectives and suggestions have been also provided by Prof. Dr. Klaus Obermayer and Andre Paus in a course of working together with me on Active Learning Framework (ALF) project proposal that has many common aspects with my dissertation work.

The indebted gratitude deserves the anonymous reviewers of every author's scientific publication for their insightful comments, which greatly helped to strengthen the thesis. Special thanks also go to Dr. Verena Vanessa Hafner and Dr. Benjamin Hirsch, whose suggestions and proofreading helped in making the presented matter easier to understand.

I would like to thank Dr. Athanassios Kelemis and Georgios Dimitriou from German-Hellenic Chamber of Commerce and Industry, as well as Marziyeh Kangarlou, Burak Simsek and Piotr Jacak from DAI Labor for extensively testing PIA and providing the valuable explicit user relevance feedback values.

For helping me solve numerous technical problems, I am especially grateful to Nuri Kayaoglu, Jenny Schuster, Kathy Schwartz, Eberhard Riedel, Stephan Schier and many other DAI Labor colleagues.

A special gratitude deserves Dr. Brijnesh-Johannes Jain for supporting me, during his leadership of PIA project, and having the great understanding for my research efforts while developing cooperation, exploration and coordination schemes. I am also grateful to all other PIA team members, who have made possible every single user evaluation.

I am finally deeply indebted to my parents for their encouragement, and especially to my wife Violeta, who has tolerated my working long hours and week-ends which left us with less time to spend together.