Online Optimization:
Probabilistic Analysis and Algorithm Engineering

vorgelegt von

Dipl.-Inf. Benjamin Hiller

Von der Fakultät II – Mathematik und Naturwissenschaften
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Naturwissenschaften
– Dr. rer. nat. –

genehmigte Dissertation

Berichter:	Prof. Dr. Dr. h.c. mult. Martin Grötschel
	Dr. Tjark Vredeveld
Vorsitzender:	Prof. Dr. Fredi Tröltzsch

Tag der wissenschaftlichen Aussprache: 15. Dezember 2009

Berlin 2010
D 83

Zusammenfassung

Diese Arbeit beschäftigt sich mit *Online-Optimierung*, also der Steuerung von Systemen, bei denen die für die optimierte Steuerung relevanten Daten erst mit der Zeit, d. h. *online*, bekannt werden. Wir konzentrieren uns dabei auf *kombinatorische* Online-Optimierungsprobleme, bei denen die Steuerungsentscheidungen diskret sind.

Im ersten, praktisch orientierten Teil der Arbeit werden *Reoptimierungsalgorithmen* für die Online-Steuerung komplexer realer Systeme vorgestellt. Ein Reoptimierungsalgorithmus trifft seine Steuerungsentscheidung so, dass sie in einem bestimmten Sinn "günstig" für die aktuelle Situation ist. Wir benutzen Techniken der mathematischen Optimierung, insbesondere *ganzzahlige Optimierung*, um fortgeschrittene Reoptimierungsalgorithmen zu entwickeln. Unsere erste Anwendung ist die automatische Disposition von Pannenhilfefahrzeugen des ADAC. Hier zeigt sich, dass mathematische Optimierungsmethoden den Dispositionsprozess gegenüber der Planung mit einfachen Heuristiken verbessern und kürzere Wartezeiten für die Kunden erzielen. Für die zweite Anwendung, die Steuerung von Gruppen von Personenaufzügen in Hochhäusern, entwickeln wir ebenfalls Reoptimierungsalgorithmen. Diese sind speziell auf die Steuerung von *Zielrufsystemen*, bei denen Passagiere ihre Zieletage bereits auf der Startetage eingeben, zugeschnitten. Zusätzlich zur Steuerung der Aufzugsgruppe unter Ausnutzung der Freiheitsgrade des Systems versuchen die Algorithmen, langfristig ungünstige Steuerungen zu vermeiden, was durch eine geeignete Modellierung erreicht wird. Unsere Simulationen zeigen, dass die Anzahl der Passagiere, die mit akzeptabler Servicequalität bedient werden kann, durch die Verwendung von Zielrufsystemen um 50% gesteigert werden kann.

Im theoretischen zweiten Teil stellen wir einen neuen Ansatz zur probabilistischen Analyse von Onlinealgorithmen vor. Im Gegensatz zu bestehenden Analysen bewerten wir die Güte eines Algorithmus nicht anhand des Erwartungswertes der Zielfunktion, sondern anhand der *Verteilung der Zielfunktionswerte* auf allen möglichen Eingaben, die eine globalere Beschreibung der Güte des Algorithmus darstellt. Mithilfe des Konzepts der *stochastischen Dominanz* können wir z. B. zeigen, dass der bekannte Paging-Algorithmus LRU eine bezüglich der stochastischen Dominanz optimale Verteilung erreicht, wenn die Eingabe bestimmte praktisch häufig vorkommende Lokalitätseigenschaften aufweist. Für das Online-Bin-Coloring-Problem [KdPSR01] beweisen wir eine Aussage, die das in Simulationen beobachtete Verhalten erklärt und damit eine offene Frage aus [KdPSR01] beantwortet.

Abstract

The subject of this thesis is *online optimization*, which deals with making decisions in an environment where the data describing the process to optimize becomes available over time, i. e., *online*. In particular, we study *combinatorial* online optimization problems involving discrete decisions both from a practical and a theoretical point of view.

The first part of the thesis is devoted to *reoptimization algorithms* for the online control of complex real-world systems. A reoptimization algorithm obtains its online control decision by determining a decision that is in some sense "good" for the current state of the system. We apply rigorous mathematical modelling and optimization methods based on *Integer Progamming* to develop advanced reoptimization algorithms. Our first application concerns the automatic dispatching of a large fleet of service vehicles to serve waiting customers. We find that rigorous methods improve the performance of the dispatching process, leading to shorter waiting times for the customers. In our second application we consider the scheduling of groups of passenger elevators in high rise buildings. We suggest advanced control algorithms for *destination call systems*, in which a passenger enters his desired destination floor already at his current floor. In addition to exploiting all degrees of freedom offered by the system, our reoptimization algorithms feature means to avoid decisions that will lead to undesirable online behavior. Our simulation experiments indicate that the number of passengers that can be served with an acceptable service level increases by 50% by using a destination call system controlled by our algorithms instead of a conventional system.

The second part introduces a novel kind of probabilistic analysis for online algorithms. In contrast to existing probabilistic analyses, we do not judge the quality of an online algorithm using the expectation of the objective. Instead, we consider the *distribution of the objective value* on all inputs which gives a more global description of the performance of the algorithm. Using the notion of *stochastic dominance*, we are able to establish that certain online algorithms obtain better objective value distributions than others. For instance, we can show that the famous paging algorithm LRU achieves a distribution that is optimal w. r. t. the stochastic dominance order if the request sequences exhibit locality of reference. We also apply this approach to the analysis of algorithms for online bin coloring [KdPSR01], obtaining a result that explains the behavior observed in simulations, thus resolving an open problem posed in [KdPSR01].

Acknowledgements

The theoretical part of this thesis grew out of the project “Combinatorial online planning” which was part of the DFG research group “Algorithms, Structures, Randomness”. I want to thank my colleagues from this research group, in particular Arie Koster and Volker Kaibel, for inspiring discussions and encouraging my own research. Special thanks go to Tjark Vredeveld, with whom I started the analysis of the bin coloring algorithms and who always motivated me to continue the work. I am very grateful to him for hosting me for several stays in the nice city of Maastricht.

Before even starting in the “Combinatorial online planning” project I did already work on the project with the ADAC together with Jörg Rambau and Sven O. Krumke. Both guided my research there, but also let me pursue own ideas.

The ADAC project was a good preparation for two research projects with Kollmorgen Steuerungstechnik on the development of control algorithms for passenger elevators. I want to thank Björn Kollmorgen and Peter Gerstenmeyer, who provided important insights in the details of elevator scheduling. Most importantly, they believed that we young researchers could deliver insights and control algorithms that are at least as good as if Kollmorgen Steuerungstechnik developed them on their own. I have to mention Martin Grötschel’s way of supporting young researchers here: He just lets them do own projects, so one quickly learns to take responsibility, but also to define own research goals. Andreas Tuchscherer, who also worked on the elevator projects, was a big help both in very valuable discussions and in sharing the overtime work to meet the project deadlines.

Finally, I have to thank my proof readers Torsten Klug, Jacint Szabó, and Andreas Tuchscherer for providing me with valuable feedback, despite the tight deadline I imposed on them. Their feedback, and in particular Andreas’ detailed comments and questions, helped a lot to improve the presentation and to wipe out errors. I also want to thank my other colleagues at ZIB for the nice working atmosphere and their encouragement during the completion of my thesis.

Last not least I want to thank my wife Petra for her patience, in particular during the last two months when I did almost nothing else than to finish this work.

Contents

Introduction and overview

The subject of this thesis is *online optimization*, which deals with making decisions in an environment where the data describing the process to optimize becomes available over time. Online optimization problems occur in many different settings, notably in the broad area of logistics, and are of high practical relevance.

The prime example of an online optimization problem is elevator control: The passengers requiring service of the elevator system arrive in the course of the day and the elevator control has no means of determining where and when a new passenger will arrive. Instead, it has to react to the elevator calls provided by the passengers as they are issued, trying to provide a good service to all passengers.

Online optimization problems are frequently control problems, but may be different, for instance there are online optimization problems concerning the organisation of dynamic data structures. We focus on *combinatorial* online optimization problems, meaning that the decisions to make are of a discrete nature, e. g., deciding which elevator answers a certain passenger call. This is in contrast to online control problems, where the decision variables are usually continuous, e. g., in controlling chemical production processes (although there discrete decisions might be involved, too). We study combinatorial online optimization problems and, most importantly, online algorithms both from a practical and a theoretical point of view.

Real-world combinatorial online problems are usually solved by *reoptimization algorithms*: Everytime the state of the system changes or new information becomes available, all currently available information is collected and a *snapshot problem* is solved to determine the reaction to the change, i. e., the online decision. When solving the snapshot problem, one usually generates solutions which are suppposed to be good w. r. t. some objective function, usually the same as the overall, end-of-day objective for the system that is optimized. It is, however, not at all clear that controlling a system in this way will yield good online performance, since the decisions made this way do not take into account future and thus may be bad in the long run. In fact, for the control of elevator systems there is some theoretical evidence [HKR00] that such a strategy might lead to very undesirable behavior in some situations. It may well be that the use of different or extended objectives gives better overall performance.

Moreover, the reoptimization algorithms used by practioners are usually heuristics only and are not based on a rigorous mathematical model describing the choices available. Thus they cannot exploit the full potential of the system and it is unclear how much performance is lost this way. An advantage of having a mathematical model is that it allows to employ mathematical optimization methods that can exploit the degree of freedom offered by the system. Using these methods it is possible to investigate how much potential is wasted by using weak optimization only. One can then study how the model,

e. g., the objective function, might be changed in order to obtain an improved online performance. Frequently, mathematical optimization methods require long computation times and seem therefore not to be suitable for real online control systems that require real-time decisions. Therefore, important questions in this context are

1. Does it pay off to use computationally expensive mathematical optimization methods, given the future is unknown anyway?
2. If mathematical optimization improves performance, how can the algorithms be made real-time compliant?

The results of this thesis indicate that exact mathematical optimization methods indeed help to improve the online performance, despite neglecting the future. However, they can be improved further by studying effects of the decisions on future performance and adjusting the mathematical model appropriately. Moreover, for practically relevant cases they can also be implemented to be real-time compliant.

In the theoretical part of the thesis, we strive to improve the analysis of existing online algorithms, bringing the analysis more in line with the empirically observed behavior. By now, the standard tool to analyze online algorithms from a theoretical point of view is *competitive analysis*.

In the simplest theoretical model [BEY98] of online optimization, the input is represented as a *request sequence* that reveals the information (abstracted as a *request*) piece by piece. An online algorithm processes the request sequence by making for each request an (irrevocable) decision, trying to minimize some overall objective. While processing a request, the online algorithm has no access to later, i. e., future, requests. One way to evaluate the quality of an online algorithm is *competitive analysis*, which measures how much the performance of the algorithm degrades w. r. t. what would have been possible if one knew the future. One can think of this as the "price of not knowing the future".

Comparing with the optimal decisions when knowing the future is often not a suitable yardstick. There are several interesting online optimization problems where competitive analysis fails in the sense that it does not allow to discriminate between algorithms that perform well in practice and ones that do not. In a sense, in these problems there is a "bad future" for any algorithm, i. e., no matter how you make your decisions. To address this issue, many extensions of and alternatives to competitive analysis have been suggested.

We suggest a new approach of probabilistic analysis, i. e., we assume that the request sequence is generated by some random process. An important feature of our approach is that we do not compare an algorithm to some hypothetical optimum, but we compare two algorithms directly. We also do not consider a single number (e. g., an average or a maximum) describing the performance of an online algorithm, but rather consider the distribution of the objective values on *all* request sequences. To compare the objective value distributions of two online algorithms we use *stochastic orders* which are partial orders on probability distributions. In particular, we suggest to compare online algorithms A and B by establishing that the distribution of objective values of A is smaller w. r. t. the *stochastic dominance order* than the objective value distribution of B.

A frequent argument against probabilistic analysis is that often the probabilistic assumptions or the probabilistic model are not justified and considered for reasons of tractability only. Our approach has the interesting property that a stochastic dominance result obtained for the uniform distribution admits a strong deterministic interpretation, stating that A achieves small objective values on at least as many request sequences as B. Put in other words, A is at least as good as B on many possible futures, which is a very desirable property for an online algorithm.

Using our stochastic-dominance-based approach, we are able to prove that for the well-studied paging problem, the LRU algorithm is an optimal online algorithm for several classes of interesting request sequences. Most of these results were known before, but our approach provides a simplifying, unifying, and also strengthening view on those results. In an application to the online bin coloring problem [KdPSR01], we are able to show that the algorithm GreedyFit is better w. r. t. stochastic dominance than the algorithm OneBin, thus explaining the superior performance of GreedyFit over OneBin observed in simulations.

Outline of the thesis The first part of the thesis deals with the design and application of reoptimization algorithms in real-world environments. In Chapter 1 we experimentally study reoptimization algorithms for dispatching the service vehicles of the ADAC, a big German automobile club. In particular, we address the question whether better reoptimization algorithms can be constructed by sacrificing modelling accuracy to obtain simpler models that are more focused on the near future. The contents of this chapter is taken from [HKR06].

Chapters 2 and 3 consider algorithms for controlling groups of passenger elevators in high rise buildings. Most of the elevator systems currently installed use conventional 2-button control, which means that at each floor there are two input buttons for specifying whether passengers want to go up or down. The major drawback of such systems is that the elevator control algorithm learns of the desired destination floors only after an elevator is sent to the floor. This destination floor information can, however, be crucial for an efficient scheduling of the elevators, especially in high rise buildings. The elevator industry thus developed *destination call systems*, in which the passengers can enter the desired destination floor already at their current floor. We investigate how advanced algorithms can take advantage of this additional information to increase the performance of an elevator system. Chapter 2 provides an thorough introduction in the fundamentals of controlling passenger elevators and devises a general mathematical model describing the schedule of the elevator, taking into account the relevant constraints. We also describe a best-insertion type heuristic for scheduling elevators that has been the basis of the real-world implementation of the algorithm that is currently employed by our industry partner, Kollmorgen Steuerungstechnik GmbH.

In Chapter 3 we study more advanced elevator scheduling algorithms, using current state-of-the-art techniques to obtain provably optimal schedules in each reoptimization run. Based on the general model from Chapter 2, we develop a novel algorithm to control destination call systems. The heart of this algorithm is a Branch & Bound method for which we study three different lower bound techniques, two of which have been developed in this work. In our computational experiments it turns out that one of our lower bound methods leads to short running times of the overall exact reoptimization algorithm. Thus our algorithm becomes real-time compliant, at least for low to medium load traffic. Improving the algorithm further is an ongoing research project. We also provide data from extensive simulation experiments based on real-world buildings, allowing us to conclude that destination control systems can serve up to 50% more traffic than conventional systems without degrading the service quality. To achieve this, we extended the reoptimization model by additional constraints that may degrade the quality of the snapshot solution, but lead to a better online performance in the long run.

In the second part of the thesis, we develop our novel approach to the analysis of online algorithm and apply it to several online optimization problems.

Our approach is introduced and discussed in Chapter 4, which also contains a detailed review of related work. At first sight, a stochastic dominance relation between two online algorithms seems to be a too strong result to hope for. We establish using computer simulations of Markov chains that this phenomenon shows up, however, between interesting bin packing and bin coloring algorithms. We conclude the chapter with a series of simple proofs for the optimality of the paging algorithm LRU in the presence of locality of reference. The stochastic dominance approach allows us to unify and strengthen several results that were known before.

Chapter 5 is devoted to a detailed analysis of online bin coloring algorithms. It turns out that an analysis of the stochastic dominance relation between the algorithms GreedyFit and OneBin suggested by our simulation results from Chapter 4 is harder than the analysis of the paging algorithm LRU. We show that techniques based on the monotonicity of stochastic matrices cannot easily be used to derive this result. Using a novel proof technique, we are nevertheless able to establish that GreedyFit is better w. r. t. stochastic dominance than OneBin. The last section of the chapter containing the proof of this result is based on [HV08].

Part I.

Design of reoptimization algorithms

Chapter 1.

Dispatching service vehicles for the ADAC under high load

This chapter is based on joint work with Sven O. Krumke and Jörg Rambau. The text is a slightly adjusted version of our *Discrete Applied Mathematics* article [HKR06].

1.1. Issues and motivation

The ADAC (Allgemeiner Deutscher Automobilclub) is the major German automobile association. In Germany, ADAC is best known for its "yellow angels", a fleet of service vehicles operated by ADAC. The "yellow angels" provide help service to people experiencing car breakdowns. The fleet of service vehicles is controlled by five service centers spread over Germany. Customers can report their breakdowns by phone, the service center responsible for the region of the breakdown will then ensure service, either by sending an appropriate ADAC service vehicle or by assigning the customer to an external contractor.

In each service center, dispatchers coordinate the service vehicles of a certain region by assigning each vehicle the customer to serve next. All over Germany there are some 1600 service vehicles with different capabilities plus thousands of contractors available to provide the required service. The task of the dispatchers is to do the assignments such that a high service quality, i. e., short waiting times for customers, is achieved at low operational cost. Operational cost arise e. g., due to driving, working time, and contractor cost. Observe that although the service vehicles are assigned to at most one customer at a time, it may be beneficial if the dispatcher is actually planning a tour for each vehicle that serves a sequence of customers.

Until 2004, dispatching was done manually with no other support than a graphical display of vehicle and customer locations. With increasing traffic, the need for more advanced computer support arose. Thus ADAC and a group at Zuse Institute Berlin developed an automated dispatching algorithm called ZIBDIP [KRT02]. The ZIBDIP algorithm has been incorporated into the automated dispatching system for service vehicles (units) and service contractors (contractors). It is based on exact cost-reoptimization, meaning that a current dispatch, which is near optimal on the basis of the current data, is maintained. The current dispatch is a set of tours for each vehicle that covers all known yet unserved requests; whenever a unit becomes idle its next request is read from the current dispatch. At each event (new request, finished service, etc.) the dispatch is updated by a reoptimization run, generating again a near-optimal current dispatch.

A feasible current dispatch for all known requests and available service vehicles is a partition of the requests into tours for units and contractors such that each request is in exactly one tour and each unit drives exactly one tour (maybe directly to its home position) so that the cost function is minimized. Cost contributions come from driving costs for units, fixed service costs per requests for contractors, and a strictly convex lateness cost for the violation of soft time windows at each request (currently quadratic). The latter cost structure is chosen so as to avoid large individual waiting times for customers.

It is not a-priori clear that such a rigorous reoptimization yields the best, or even a good, long-term cost, as a "good" decision at a certain point in time may turn out to be a bad one in view of the unanticipated future; this is the *online issue* of the dispatching problem. Indeed, at times in the literature it is claimed that exact reoptimization (i. e., with small optimality gap) does not pay in practice because of the unknown future requests [ZO00, p. 5]. In the case of this particular application, however, the results of exact reoptimization are satisfying [GKRT02], in concordance with [BSL96, Sec. 8.4].

Although the reoptimization problem, which is modeled as a set partitioning problem for tours, has an astronomical number of variables, it can be solved by a dynamic column generation procedure. An effective method to obtain provably good solutions in ten seconds (the *real-time aspect* of the dispatching problem) is *dynamic pricing control*, which is the main feature of our ZIBDIP algorithm (a thorough description of the algorithm and computational results can be found in [KRT02]).

As it turns out, the fixed costs for service by contractors bound the dual values of requests. Thus, contractors substantially contribute to the success of ZIBDIP. The contractor, however, may in practice decline to serve suggested requests, in which case this request has to be manually reentered into the system, with the additional constraint that it must not be assigned to this contractor again. This is a time consuming process. In metropolitan areas, contractors decline so often that the ADAC decided to remove contractors from the model.

In simulations on ADAC production data without contractors, derived from three days in December 2002 with high load, we encountered significant reoptimization gaps. For 2002/12/13, e.g., Figure 1.1 shows the gap of the reoptimization result to the respective lower bound coming from the optimal solution of the LP relaxation computed a-posteriori for each reoptimization. The reoptimization still works well in most cases, but under high load the solutions – delivered after ten seconds – exhibit optimality gaps around 3% on average but up to 10% in peak load situations.

One way to overcome this problem is to consider *simplified reoptimization models* that stem from the following considerations: In principle, for each unit we only have to determine the next request to work on. The complete dispatch is computed only to pick up future synergies by considering more than one request per unit. Synergies that are implemented only very far in the future will be disturbed by new requests anyway; therefore, an exact pre-calculation of the best decisions in, say, two hours may not really be necessary; consequently, one can try to cover only *a subset of requests* in a reoptimization step.

The issue of this experimental work is: should one stick to the complete model and accept occasional substantial reoptimization gaps, or is it better to simplify the reoptimization

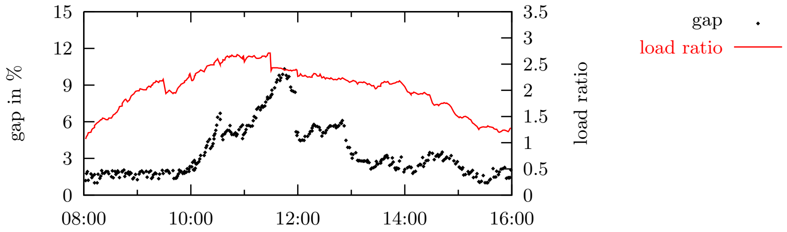


Figure 1.1.: *Optimality gap over time of ZIBDIP (the load ratio is the number of requests per unit in a reoptimization problem).*

model so as to eliminate the reoptimization gap? This question is answered on the basis of simulation studies, performed on the aforementioned ADAC production data: we first compare the original ZIBDIP reoptimization to several methods to select subsets of requests that have to be covered by any solution of the reoptimization run. Then ZIBDIP competes with two simple online heuristics for the ZIBDIP model in order to estimate how even larger reoptimization gaps harm in the long run.

1.2. Simplified models

We developed and evaluated the following strategies for the selection of requests to be covered in a reoptimization run. In the sequel, we describe the original and each simplified model in more detail.

We will use R and U to denote the set of requests and units, respectively. In all our models, there is a binary selection variable x_T for each feasible tour T. Such a tour is given by a unit u and a sequence of requests to be served by u in the given order. We call the set of all feasible tours $\mathcal{T}$ and the set of all feasible tours for unit u is written as $\mathcal{T}_u$.

We denote by c_T the cost coefficient of tour T. This is a weighted sum of strictly convex lateness costs, linear drive costs, and strictly convex overtime costs. Lateness costs in the reoptimization are incurred whenever a request is served after a waiting time of more than 15 min. The true target for the waiting time is higher. The 15 min deadline in the reoptimization problem was derived from the following consideration: the true waiting time for a request should lead to the same lateness costs as the fixed contractor costs for serving that request. This is motivated by the wish that requests that can not be served inside the true time window by a unit should be served by a contractor in order to reach the true target time. The exact formula including the numerical values of the coefficients of the cost function can not be disclosed here.

Let (a_{vT}) be the incidence matrix of requests and tours.

1.2.1. The original ZIBDIP model

The original reoptimization problem solved by ZIBDIP without contractors reads as follows.

$$\begin{aligned}
&\min \sum_{T\in\mathcal{T}} c_T x_T \quad \text{s.t.} && && \\
&\sum_{T\in\mathcal{T}} a_{vT} x_T = 1 && \forall v \in R && \text{(partitioning requests)}\\
&\sum_{T\in\mathcal{T}_u} x_T = 1 && \forall u \in U && \text{(partitioning units)}\\
&x_T \in \{0,1\} && \forall T \in \mathcal{T} &&
\end{aligned}$$

In contrast to the following simplifications this model guarantees that, after every reoptimization, each request is assigned to exactly one unit because of the set partitioning constraint. Every unit has to drive exactly one tour, where the direct move to its home position is also a feasible tour, the *drive-home tour*. The details of the column generation based solution method ZIBDIP are described in [KRT02]. It is relatively easy to adjust ZIBDIP to solve the following simplified models.

1.2.2. The simplified model 4-ZIBDIP

Select those requests that are among the four closest to some unit. This can be generalized to k-ZIBDIP. In the following, *k-close requests* are requests that are among the k closest to some unit, denoted by $R_k \subseteq R$. In formulae, we obtain the following model:

$$\begin{aligned}
&\min \sum_{T\in\mathcal{T}} c_T x_T \quad \text{s.t.} && && \\
&\sum_{T\in\mathcal{T}} a_{vT} x_T = 1 && \forall v \in R_k && \text{(partitioning k-close requests)}\\
&\sum_{T\in\mathcal{T}_u} x_T = 1 && \forall u \in U && \text{(partitioning units)}\\
&x_T \in \{0,1\} && \forall T \in \mathcal{T} &&
\end{aligned}$$

1.2.3. The simplified model PTC (Prescribed Total Cover)

Relax the set partitioning condition to set packing, and require that a request set of cardinality twice the number of units is covered by tours of units. This leads to the following model, where n_T is the number of requests in tour T:

$$\begin{aligned}
&\min \sum_{T\in\mathcal{T}} c_T x_T \quad \text{s.t.} && && \\
&\sum_{T\in\mathcal{T}} a_{vT} x_T \le 1 && \forall v \in R && \text{(packing requests)}\\
&\sum_{T\in\mathcal{T}} n_T x_T \ge 2|U| && && \text{(cardinality)}\\
&\sum_{T\in\mathcal{T}_u} x_T = 1 && \forall u \in U && \text{(partitioning units)}\\
&x_T \in \{0,1\} && \forall T \in \mathcal{T} &&
\end{aligned}$$

Note that if we replace the cardinality constraint by

$$\sum_{T \in \mathcal{T}} n_T x_T \geq \min\{2|U|, |R|\}$$

the PTC model is equivalent to the original ZIBDIP model if there are at most two requests per unit on average.

1.2.4. The simplified model ShadowPrice

Solve the LP relaxation of ZIBDIP. To find an integral solution, relax the set partitioning condition to set packing and change the cost of each tour to its reduced cost from the hopefully near optimal LP solution. In the following, the new cost coefficient $\tilde{c}^T$ of a tour T is the *reduced cost* of T w. r. t. the best LP solution that can be found in time. Because the LP solution algorithm works by dynamic column generation, this solution is an optimal solution to the last RLP that could be solved in time. The resulting model reads as follows:

$$\begin{aligned}
\min \sum_{T \in \mathcal{T}} \tilde{c}_T x_T \quad \text{s.t.} & & & \\
\sum_{T \in \mathcal{T}} a_{vT} x_T \leq 1 & \qquad & \forall v \in R & \qquad \text{(packing requests)} \\
\sum_{T \in \mathcal{T}_u} x_T = 1 & \qquad & \forall u \in U & \qquad \text{(partitioning units)} \\
x_T \in \{0,1\} & \qquad & \forall T \in \mathcal{T} &
\end{aligned}$$

In this model, requests are assigned to units only if their LP dual prices together with the drive-home cost of a unit pay enough to weigh out the primal costs of their service. This requires that the LP relaxation can be solved fast, since the LP is not simplified at all.

This model is motivated by the fact that not only the column generation process is slowed down by the absence of contractors but also the IP-solution process. This can be explained as follows: In the presence of contractors, for each request there is an elementary column covering exactly that request. That way, each set packing solution using cheap tours through suitable requests can be augmented to a feasible set partitioning solution by adding such elementary columns, each at the fixed cost of the corresponding contractor. When there are no contractors, such elementary columns may become much more expensive than the price for a contractor, and for this reason they may even be overseen in the column generation process. From the remaining columns it may be difficult to augment a set of nice tours to a feasible set partitioning solution at reasonable costs. Relaxing the set partitioning condition to set packing on the model-level by-passes this problem completely and may lead to a faster IP-solution process.

1.2.5. The simplified model ZIBDIP$_{\text{dummy}}$

Introduce a dummy contractor. This contractor can be assigned arbitrarily many requests at the same time at no extra cost, i. e., in reality, these requests are unassigned for the moment. In order to enforce a cost for the assignment to the dummy contractor, its arrival time at any request is a fixed time, the *dummy contractor delay.* In our case, 135 min were chosen. In the following, d_v is the dummy contractor delay, i. e., the lateness cost for 135 min additional delay at v (on top of the current age of v). By using decision variables y_v to indicate whether request v should be served by the dummy contractor, we obtain the following model:

$$\begin{aligned}
\min \sum_{T\in\mathcal{T}} c_T x_T + \sum_{v\in\text{requests}} d_v y_v \quad \text{s.t.} & & & \\
\sum_{T\in\mathcal{T}} a_{vT} x_T + \sum_{v\in R} y_v &= 1 & \forall v \in R & \quad \text{(partitioning requests)}\\
\sum_{T\in\mathcal{T}_u} x_T &= 1 & \forall u \in U & \quad \text{(partitioning units)}\\
x_T &\in \{0,1\} & \forall T \in \mathcal{T} & \\
y_v &\in \{0,1\} & \forall v \in R &
\end{aligned}$$

This model implies that, in an optimal solution, for any request in a tour of a unit, service will start after at most 135 minutes after reoptimization; otherwise, the request would have been assigned to the dummy contractor.

We remark that all simplified models, including ZIBDIP$_{\text{dummy}}$, can be augmented to accommodate real contractors as soon as this might be reasonable again.

1.3. Simplified reoptimization algorithms

We furthermore evaluated two heuristics for the original model, which were used in the reoptimization process as replacements for ZIBDIP. One should mention that in each reoptimization with either model, the solutions of the previous reoptimization are reused as start solutions – a simple but essential technique to stabilize the dispatching process in case of occasional suboptimal reoptimization.

1.3.1. The simplified algorithm BestInsert

A new dispatch is obtained by taking the dispatch of the previous reoptimization, removing all requests that have been served in the meantime, and inserting new requests at minimal additional cost w. r. t. to the original ZIBDIP-model.

1.3.2. The simplified algorithm 2-Opt

A first tentative dispatch is computed by BestInsert. This dispatch is then improved by successively exchanging two requests between distinct time slots in the dispatch if this

instance	*requests*	*units*	*requests per unit*
2002/12/07	2123	125	16.98
2002/12/13	2537	146	17.38
2002/12/14	1731	131	13.21

Table 1.1.: *Sizes of high load instances used for simulation.*

decreases the cost. It has to be noted that the complicated cost function for tours leads to quite some computational effort for the calculation of the 2-Opt solutions.

1.4. Computational results

The simulation data stems from three days of production at ADAC in December 2002; instance sizes are given in Table 1.1. Depending on the instance, between 1700 and 2100 reoptimization runs were triggered.

The software ran on a standard Linux PC equipped with a 2.4 GHz Pentium 4 CPU, 4 GB RAM, distribution SuSE 9.0 using kernel 2.4.21-202-smp. It was compiled with gcc 3.3.1 and used the LP / MIP solver CPLEX 8.0. Each reoptimization run was interrupted after 10 seconds run-time.

1.4.1. Simplified models

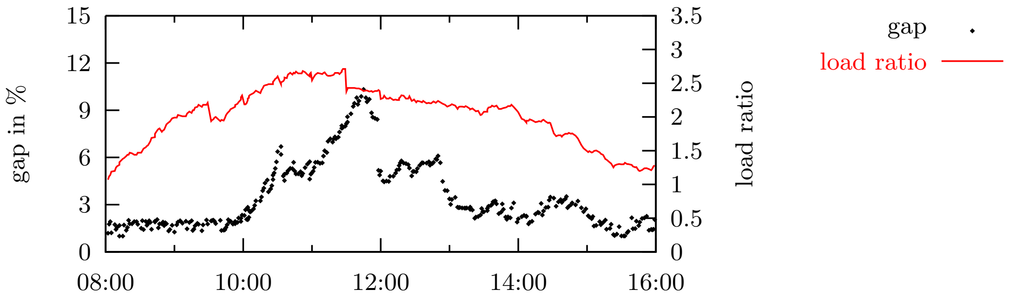
Since all our simplified models by design do not guarantee service for all requests under low load, we evaluated them in the following way: If the load ratio was less or equal to 2.0, reoptimization was performed using the original ZIBDIP model. If the load ratio exceeded 2.0 we employed the respective simplified model (this is natural since these models were designed for high load situations).

First of all, we checked whether the simplified models can reduce the optimality gaps of the reoptimization solutions that could be computed in 10 s (see Figure 1.2). It can be seen that all models reduce the gap significantly, i. e., the corresponding optimization problems are easier to solve in 10 s.

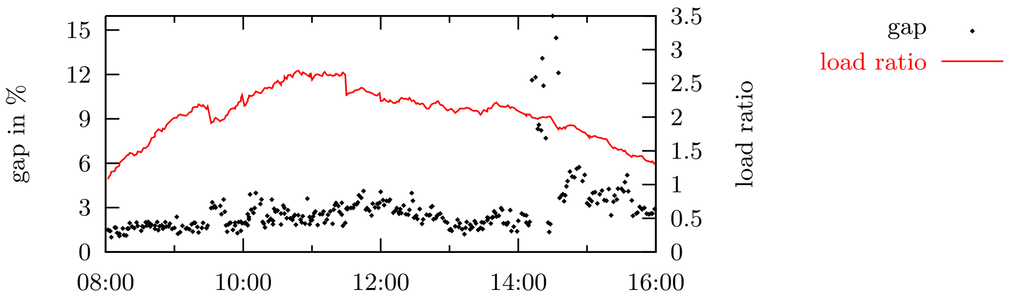
We think that some single large optimality gaps for 4-ZIBDIP and PTC stem from switching back to ZIBDIP if the load ratio drops temporarily below 2.0. The switches are particularly "unsmooth" for these two models, since ZIBDIP has to run essentially without a feasible start solution. This discontinuity in operation is certainly a drawback of 4-ZIBDIP and PTC.

Next, we investigated the cost over time w. r. t. the reoptimization cost function, designed in cooperation with ADAC (see Figure 1.3 on page 17).

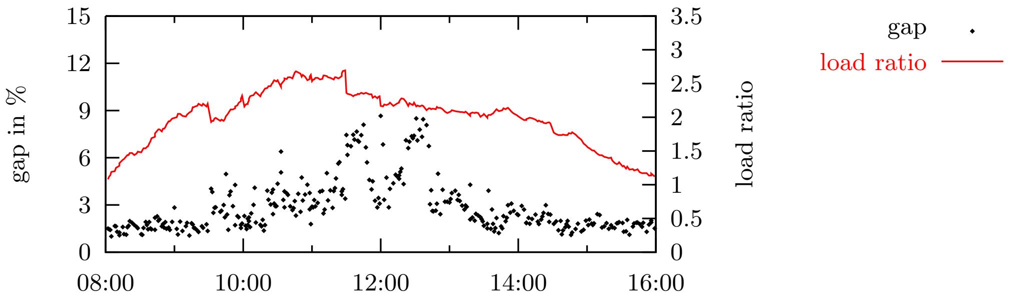
The results: only ShadowPrice and $\text{ZIBDIP}_{\text{dummy}}$ are competitive against ZIBDIP, although ShadowPrice seems to degrade in performance in the largest instance (b). In two out of three instances, ShadowPrice and $\text{ZIBDIP}_{\text{dummy}}$ have even slightly lower long-term cost than ZIBDIP, though by a small margin. In the largest instance with the most



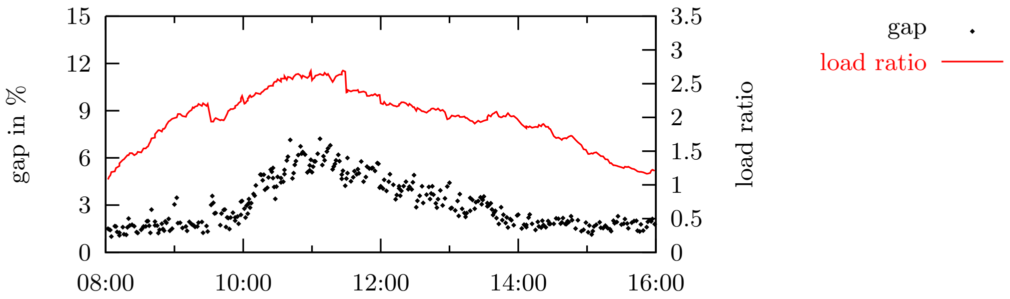
(a) ZIBDIP



(b) 4-ZIBDIP



(c) PTC



(d) ZIBDIP$_{\text{dummy}}$

Figure 1.2.: *Optimality gaps and load ratios for simplified models and* ZIBDIP*. The optimality gap of* ShadowPrice *is inevitably infinite, since the lower bound the LP provides w. r. t. the modified cost (which is the reduced cost) is zero.*

difficult reoptimization problems, however, the original ZIBDIP is superior. On average, however, the results are in favor of $\mathsf{ZIBDIP}_{\text{dummy}}$.

Since the reoptimization cost function of ADAC is quite a complicated mixture of lateness, drive, and overtime costs, we decided to investigate two standard measures on the so-called lateness time vectors (see Figure 1.4 and 1.5 on page 18f). The lateness time of a request is its waiting time portion that exceeds the allowed waiting time, fixed by ADAC. We calculated the L_1 norms and the L_2 norms of the lateness time vectors with one entry for each request. The former norms measure the average waiting time, the latter norms penalize in particular large individual lateness times, which is desirable from a fairness point of view. One should mention that these two criteria are also of vital interest in the evaluation of the long-term behavior of online-algorithms. The ADAC reoptimization objective was chosen to contain more aspects since reoptimization of L_1 and L_2 norms alone, resp., did not lead to satisfactory overall results.

It is evident that w. r. t. these lateness time measures, $\mathsf{ZIBDIP}_{\text{dummy}}$ is never worse than second best; moreover, it performs best in four out of six evaluations. ShadowPrice shows the worst L_1 norms, although the L_2 norms are good. We have no explanation for this.

The good L_1 norms of PTC are due to the fact that, obviously, individual requests are postponed in favor of new requests that can be served faster. This can be seen very clearly in the L_2 norm diagrams, in which PTC performs worst. Uncontrolled deferment of requests is a very undesired property of an online algorithm. Therefore, PTC can not be recommended for tasks in which fairness is an issue. In our application, fairness certainly is an issue, whence the ADAC cost function contains a strictly convex waiting time penalty.

The answer to our main question is that the model error of most of our high-load models leads to worse long-term behavior than the computational error that ZIBDIP produces (Figure 1.3). Therefore, model simplifications have to be treated with great care. In our case, $\mathsf{ZIBDIP}_{\text{dummy}}$ delivers the overall slightly best solution. One needs to be careful, though: a substantially smaller contractor delay of 45 min would lead to a tiny reoptimization gap; it, however, would at the same time produce unacceptable long-term costs because too many requests stay unassigned for too long. (This was, by the way, observed when we were looking for a good dummy contractor delay. Thus, $\mathsf{ZIBDIP}_{\text{dummy}}$ involves some parameter tuning that the original ZIBDIP does not.)

1.4.2. Simplified reoptimization algorithms

The results so far could lead us to the conclusion to keep the original model but to use simplified reoptimization algorithms, since it seems that the optimality gap does not harm too much. After all, the implementation of a dynamic column generation procedure means a substantially larger effort, which is important especially in the industrial context.

Since we hear quite frequently such arguments in order to promote the use of heuristics rather than exact mathematical programming methods, we followed also this line in our simulation experiments and found out the following: Larger computational errors in the reoptimization can increase the long-term costs even more significantly than the model errors above.

This is most incisively shown by the bad performance of BestInsert (Figures 1.6, 1.7, and 1.8 on pages 20ff). Even 2-Opt can not catch up with ZIBDIP and $\mathsf{ZIBDIP}_{\text{dummy}}$ in the heavier instances. In the largest instance (b), 2-Opt ends up at a long-term cost of 20 % above ZIBDIP and $\mathsf{ZIBDIP}_{\text{dummy}}$. Especially striking is the fact that, in the largest instance, the cost of 2-Opt is constantly increasing over time relative to ZIBDIP. That means: the reoptimization errors accumulate.

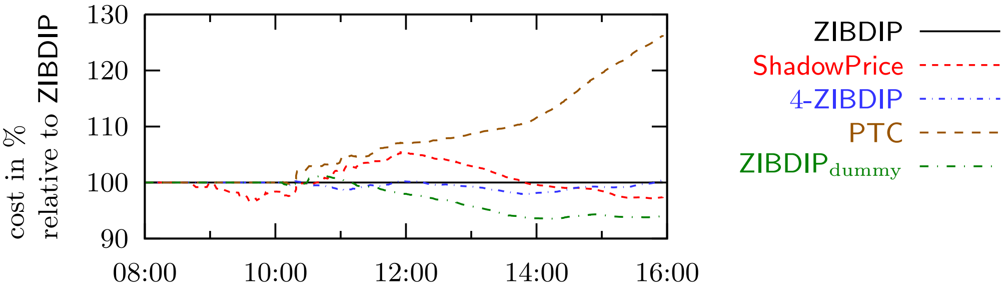
In particular: in our application it is certainly not true, that deliberately sticking to the suboptimal solutions of heuristics like BestInsert in order to leave space for future requests can yield superior long-term behavior (compare [ZO00, p. 5]). We are not saying that reoptimization is the best possible policy, maybe not even in our application. We claim: if anything is wrong with the reoptimization policy then this defect is not cured by using suboptimal solutions to the reoptimization problems.

The good overall performance of $\mathsf{ZIBDIP}_{\text{dummy}}$ may stem not only from closing the optimality gap in the reoptimization process; it seems, moreover, that the special model of $\mathsf{ZIBDIP}_{\text{dummy}}$ makes perfectly sense in the dynamic environment: since requests that are assigned to the dummy contractor would otherwise be served quite far in the future, with a high probability their position in the dispatch will change anyway. These considerations led us to the conclusion to install $\mathsf{ZIBDIP}_{\text{dummy}}$ as the default reoptimization model in the automatic dispatching software for ADAC.

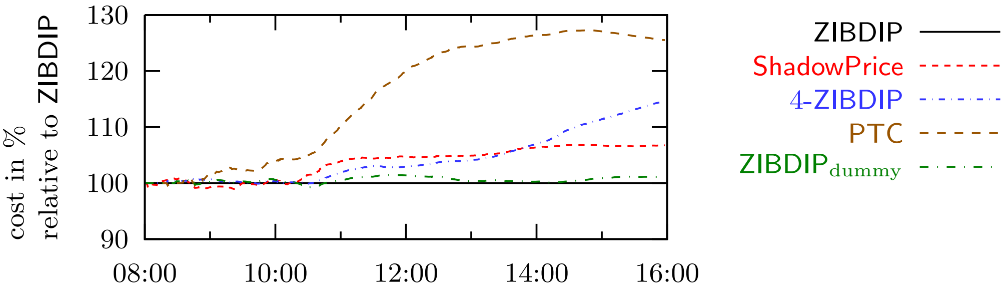
1.5. Significance

The production software for automated dispatching of ADAC service vehicles is delivered by Intergraph Public Saftety (IPS), based on the ZIBDIP algorithm. In the view of the results presented in this work, ADAC has filed a change request for the production software: $\mathsf{ZIBDIP}_{\text{dummy}}$ is now the standard reoptimization model because it has proven to be more robust against sudden load increase.

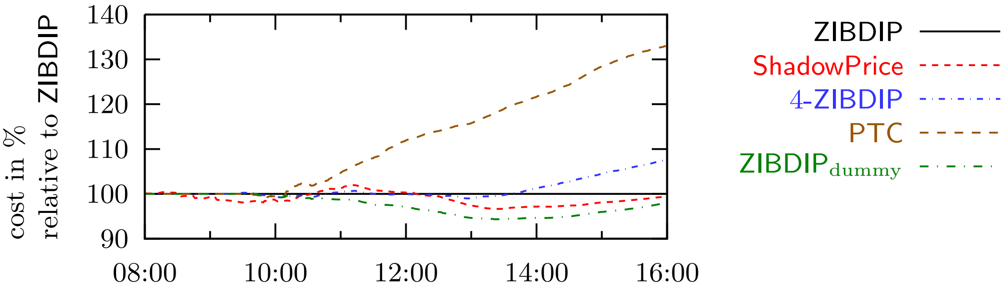
The key learning is that rigorous reoptimization on the basis of mathematical programming – though myopic w. r. t. unknown future requests – yields the best results in this particular application.



(a) 2002/12/07

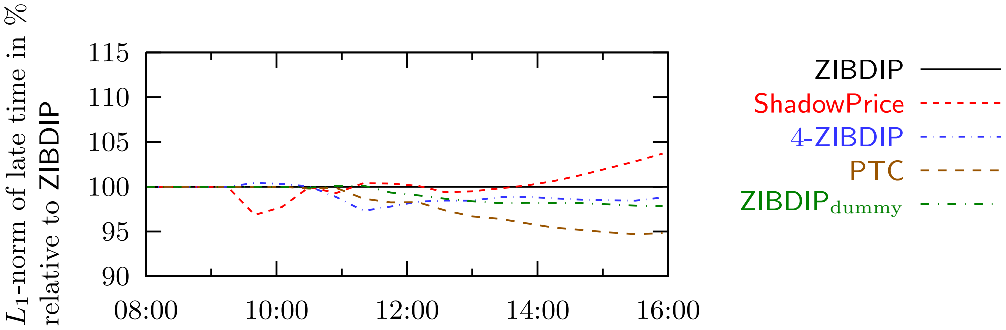


(b) 2002/12/13

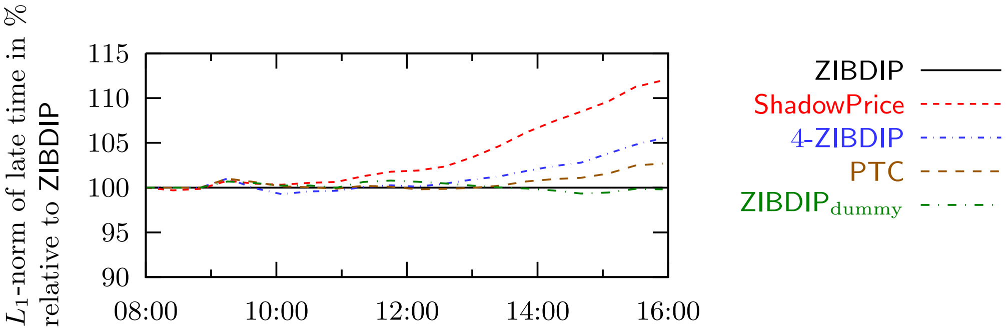


(c) 2002/12/14

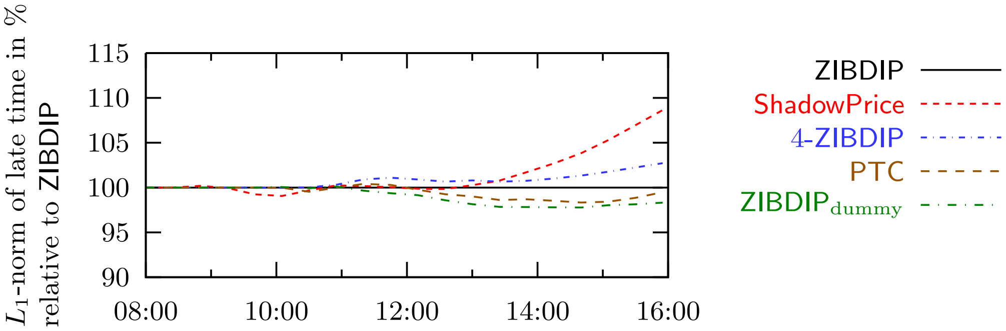
Figure 1.3.: *Comparison of* ZIBDIP *and simplified models w. r. t. the nonlinear cost function used by ADAC.*



(a) 2002/12/07

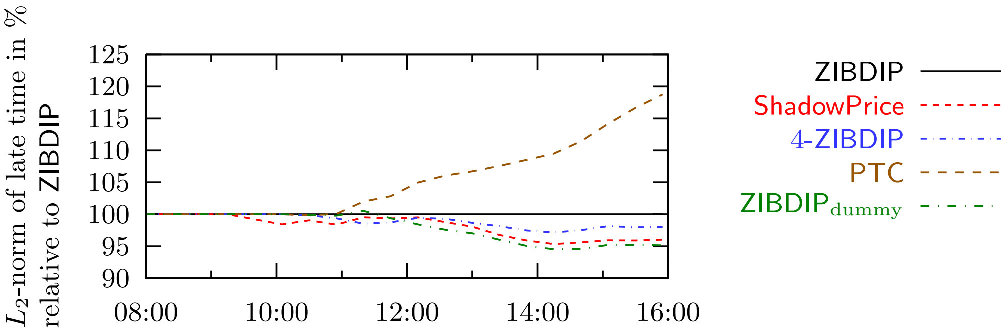


(b) 2002/12/13

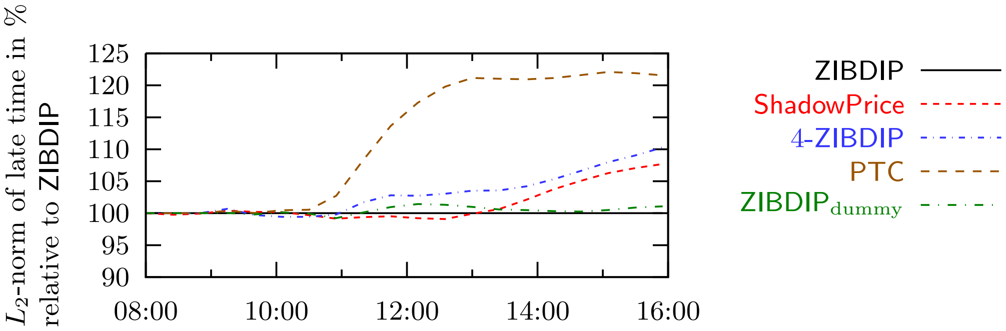


(c) 2002/12/14

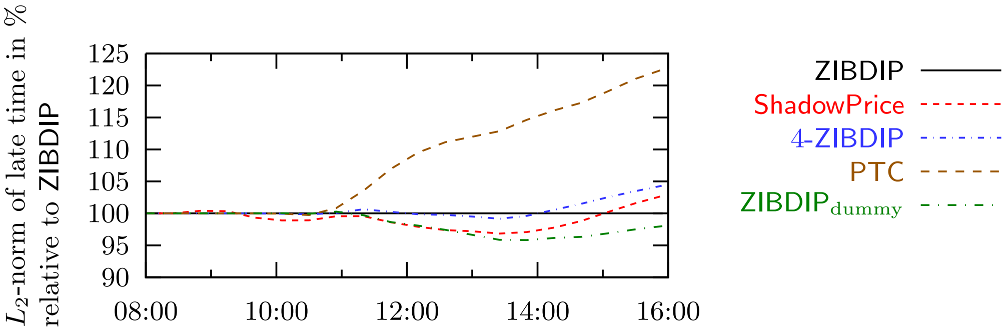
Figure 1.4.: ZIBDIP *vs. simplified models:* L_1*-norm of lateness time.*



(a) 2002/12/07

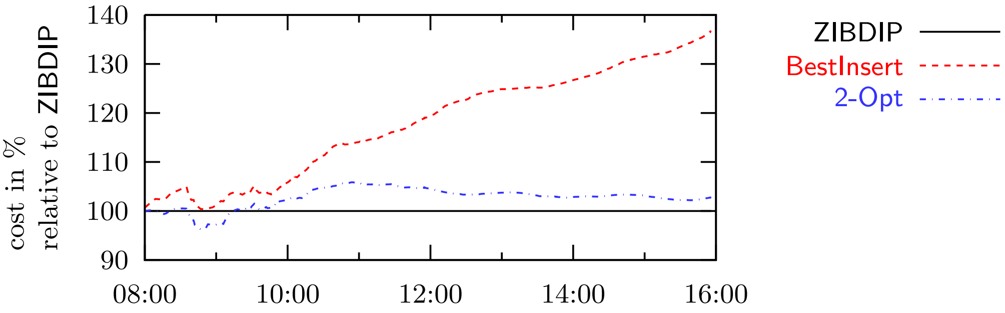


(b) 2002/12/13

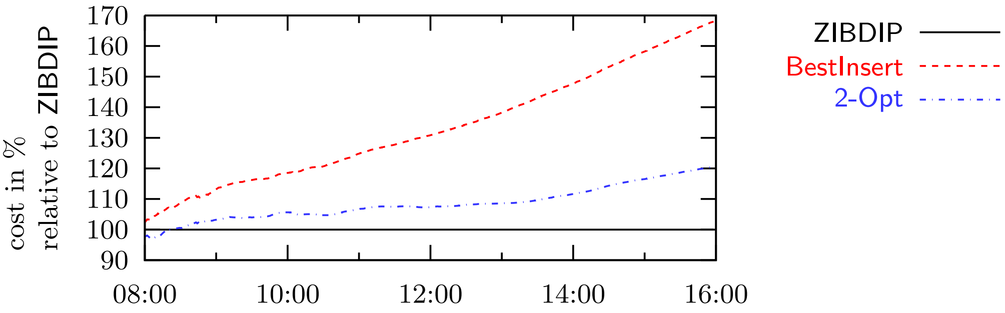


(c) 2002/12/14

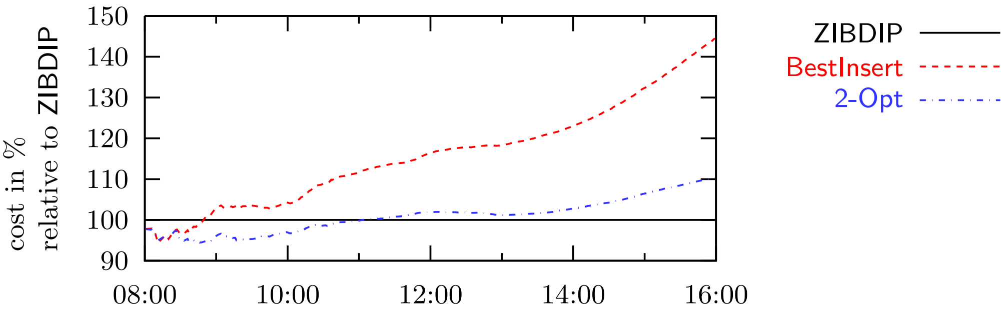
Figure 1.5.: ZIBDIP *vs. simplified models:* L_2*-norm of lateness time.*



(a) 2002/12/07

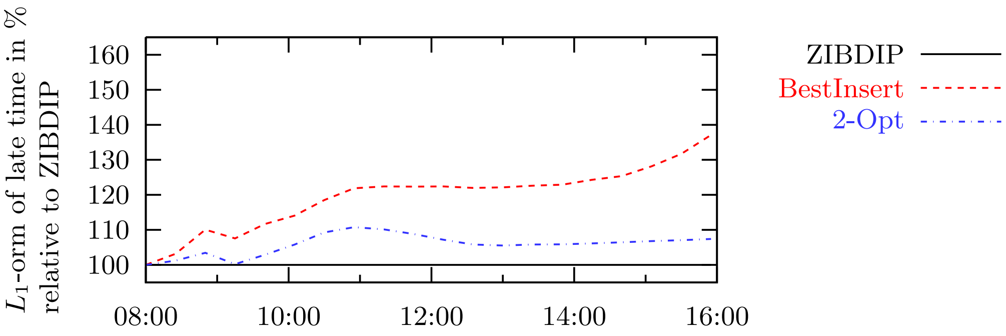


(b) 2002/12/13

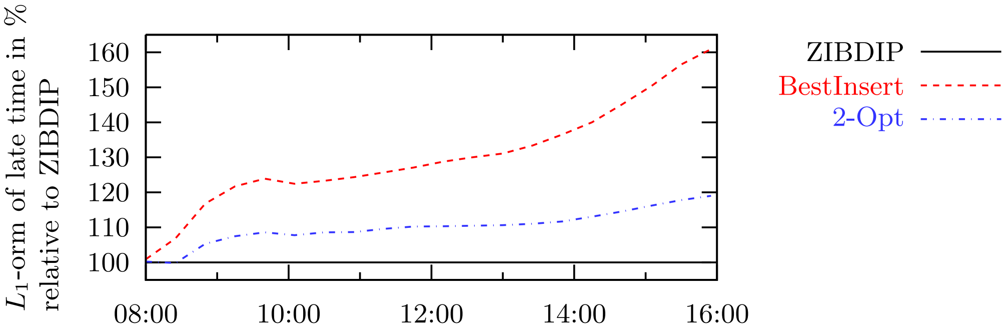


(c) 2002/12/14

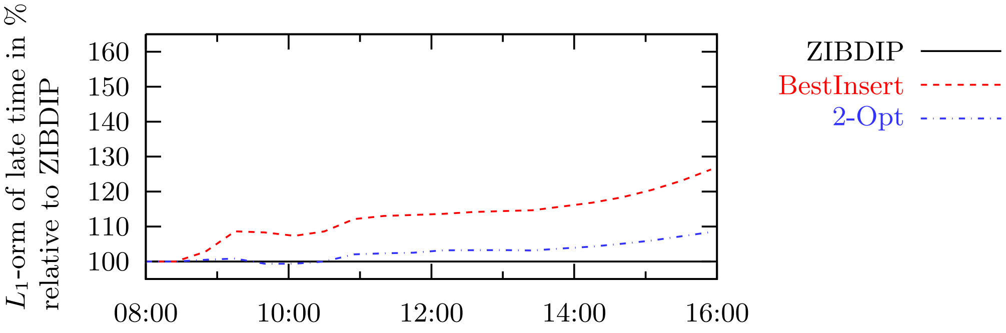
Figure 1.6.: *Comparison of* ZIBDIP *and the heuristics w. r. t. the nonlinear cost function used by ADAC.*



(a) 2002/12/07

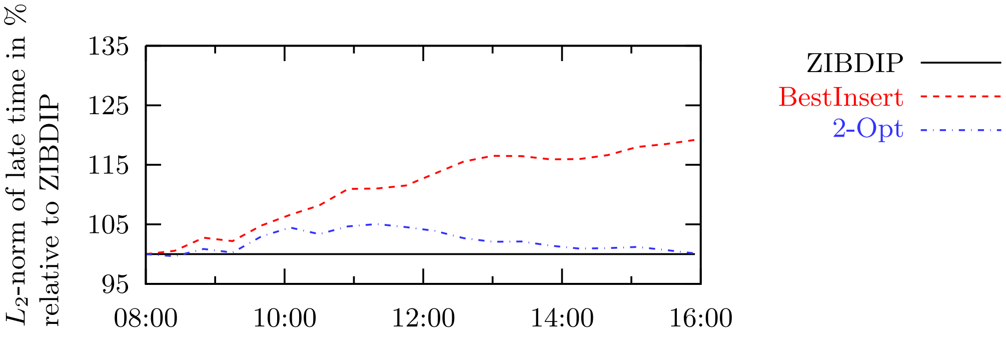


(b) 2002/12/13

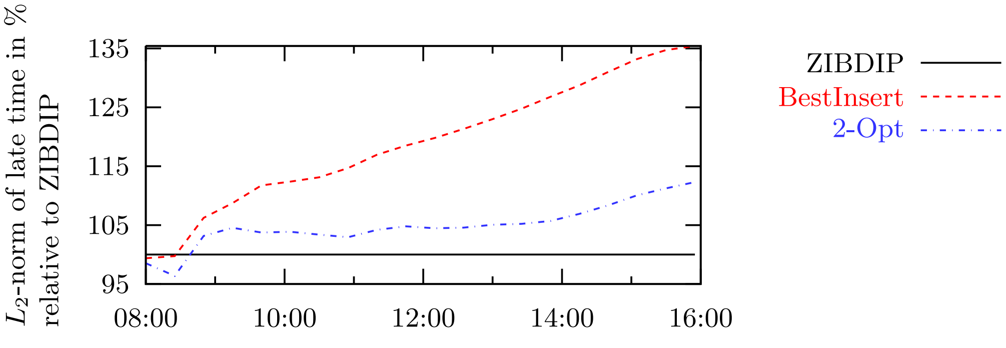


(c) 2002/12/14

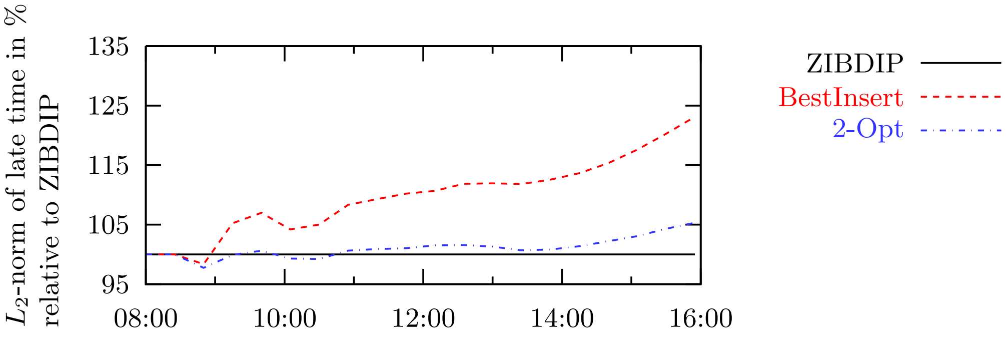
Figure 1.7.: ZIBDIP *vs. heuristics:* L_1*-norm of lateness time.*



(a) 2002/12/07



(b) 2002/12/13



(c) 2002/12/14

Figure 1.8.: ZIBDIP *vs. heuristics: L_2-norm of lateness time.*

Chapter 2.

Group control of elevator systems

The control of elevators is probably the most well-known online optimization problem. The task is to schedule a group of elevators such that they serve the passenger flow efficiently and offer a good service level to the passengers. Obviously, this is an online optimization problem: The transportation requests of passengers become known over time and the control algorithm has to serve known requests without having information about future requests.

A real elevator control is a complex system whose most important task is the mechanical control of the elevators (accelerating, decelerating, opening and closing doors), ensuring high safety requirements. In addition to this mechanical control task the elevator control has to decide which elevator of the group serves which passenger request. Moreover, it has to decide in which order to serve the passenger requests assigned to an elevator. When talking about an elevator control algorithm, we mean an algorithm that generates a dispatch prescribing how the passenger requests are served. Although such an algorithm is just one module in a real elevator control, it is crucial for the performance of the entire elevator group.

Elevator control algorithms for elevator groups were first studied back in the 1950s, when the first automatic elevator controls were installed [Bar02]. These first algorithms were rather simple, since they had to be implemented in hardware using relays. Since then, the performance achieved by elevator control algorithms has become more important as buildings become higher and higher. In addition to algorithmic improvements one possible way to enhance this performance is to use *destination call systems* (sometimes also called destination hall call system). In such a system, a passenger registers his or her destination floor right at his start floor instead of the travel direction only. This way, the elevator control has more information as a basis for its decisions which hopefully leads to better performance. Apart from new high-rise buildings, there is another important application for destination call systems in existing buildings. It may happen that due to changing usage of a building the installed elevator system is not capable to cope with increased passenger demand. In this situation, changing over to a destination call system may be a relatively cheap alternative to upgrading the elevator system itself or installing additional elevators.

Elevators are also used in industrial applications, e. g., in high rack warehouses. Algorithms for these applications have also been studied in the literature, both from a theoretical [AKR00] and applied point of view [FR06]. However, there are significant differences in the constraints for scheduling the elevators. We will therefore restrict the discussion to passenger elevator systems; the algorithms considered here are most suitable for controlling such systems.

In this chapter, we will discuss the basic notions of elevator control and provide a general model for various variants of elevator systems. This model will be the basis for the discussion of elevator control heuristics in Section 2.5 and exact reoptimization algorithms in Chapter 3. Section 2.5 contains a detailed description of the BestInsert heuristic. This heuristic has been developed in a joint research project with our industry partner Kollmorgen Steuerungstechnik and is the basis for the algorithm currently used in their elevator controls. A preliminary version of it was presented at *Operations Research 2007* [HT08].

2.1. Conventional elevator systems vs. destination call systems

In order to realize the degrees of freedom that can be exploited by an elevator control algorithm it is important to understand the flow of information in an elevator system. Most of the terms we use are the ones used by Barney [Bar02], which is a standard reference in the elevator industry.

In a system employing a conventional (2-button) control, a passenger registers his transportation request by issuing a *landing call*, i. e., he pushes the button corresponding to his travel direction. Since most conventional systems indicate that a landing call for a direction has already been registered, the passenger does not need to issue a landing call if another passenger did so before. Even if he pushes the direction button again this will not generate an additional landing call at this floor. Once a landing call has been registered, the elevator control algorithm ensures that some elevator travels to this floor and leaves in the requested direction. Passengers are assumed to enter the first arriving elevator that leaves in the correct direction, so once an elevator arrived the landing call is considered to be served. The passenger needs to issue a *car call* inside the cabin in order to tell the elevator control his destination floor. The elevator control ensures that the elevator stops at all floors corresponding to the car calls. It may happen that the capacity of the arriving elevator is not sufficient to pick up all waiting passengers. In that case, passengers are expected to reissue a landing call.

One major drawback of a conventional system is that the elevator control learns about the desired destination floors only after the passengers boarded the elevator. At this point, it has to stop at all those destination floors. If the elevator control knew the destination floors beforehand, it might have been able to group passengers according to their destination floors, thus reducing the stops required to serve the passengers. This is the main motivation to consider destination call systems.

In a destination call system a passenger issues a *destination call* at his start floor, telling the elevator control his destination floor. In all the systems implemented so far (e. g., by Schindler [Sch90, KO02], Thyssen Krupp [SP02], and KONE [TY05]), the elevator control answers by telling the passenger which elevator is going to serve him. The passenger may then proceed to this elevator and wait for its arrival. Registered destination calls need not be indicated, so the next passenger has to issue his own destination call, whereby the elevator control acquires more information about the passengers waiting at a floor. The elevator has to ensure that the assigned elevator will arrive at this floor, leaving in the required direction. The passengers enter the assigned elevator as it arrives and indicates the matching leaving direction. Inside the elevator there is no further facility to issue calls. Having a possibility to enter a car call inside the elevator would defeat the idea of the destination call, since passengers could enter any destination floor and board the first arriving elevator which can be forced to stop at the desired destination floor. Without this opportunity, a passenger has no incentive to board another than the assigned elevator as he cannot be sure that this elevator will stop at his destination floor. Again, the passengers are expected to reissue a destination call* if they were not able to board the elevator. Note that they have to go back to the registration device to do that and that they may be assigned to a different elevator.

Clearly, the early and irrevocable assignment of a serving elevator is a disadvantage from an optimization point of view, since later reassignments might improve the dispatch substantially. This restriction can be alleviated in the following destination call system, which delays elevator assignments as long as possible. Upon registration of a destination call the elevator control only acknowledges that it registered the call. The passenger waits in the common waiting area of the elevator group for the first arriving elevator indicating his destination floor. A destination call is assigned to an elevator shortly before the elevator opens its door, so the elevator can indicate the corresponding destination floors and the passengers have time to proceed to the door. Again, passengers that could not board have to reissue their calls. Observe that this kind of system resembles the conventional system much more than the one described above; essentially the difference is that destination floors are communicated instead of directions only. From now on, we call the first kind of destination call system *immediate assignment system* and the second kind *delayed assignment system*.

One interesting aspect of passenger elevator control is that it is an online problem with partial observability and partial control only. First, there is the issue of intended or unintended misuse: Passengers might not use the signalling properly or try to trick the system to achieve better service for themselves. It is often reported that in a conventional system people press both directional buttons at the same time. In a destination call system, a passenger might not issue a destination call because he knows that his colleague is going to the same floor. A more selfish passenger might assume that more waiting passengers are considered to be more important by the elevator control and thus issue several destination calls to the same floor. In general, the elevator control cannot avoid nor detect these kinds of misuse. But even if there is no misuse, there is another problem: The elevator control cannot prescribe or detect which passengers board an elevator or not. If there is enough capacity, this is not an issue, since it is safe to assume that all matching passengers board in this situation. If, however, the capacity is insufficient the elevator control does not know anything about which passengers really boarded. In contrast to a

---

[*] The elevator control M10 described in [Sch90] does not require re-registration which is reportedly avoided by book-keeping of the loaded passengers via weighing.

conventional system, the elevator control does not get a feedback at which destination floors it has to stop to drop the passengers. Therefore, the elevator control has to ensure that any passenger that could have boarded can leave at his destination floor, i. e., it has to ensure stops at *all* registered destination floors. This may lead to what we call a *phantom stop*: None of the passengers travelling to a certain floor boards the elevator, but it stops on that floor without loading another passenger, i. e., the stop is completely unnecessary. Note that this cannot happen in a conventional system. All in all, the elevator control cannot reliably keep track of the remaining capacity of an elevator. This is the reason why we require reissuing of destination calls which would be unnecessary if we could track every passenger through the system.

However, one of the claimed advantages of a destination call system is that there is more information on the number of waiting or loaded passengers which can be used to incorporate the elevator capacity into the planning. As discussed above, the lack of boarding control and the remaining uncertainty about the exact number of waiting passengers pose severe difficulties to taking limited elevator capacity into account. We address this issue in the following ways.

1. We assume during planning that the elevator capacity is unlimited, i. e., that all passengers corresponding to destination calls matching the signalling of the elevator board it.

   The advantage of this approach is that we do not need to require a one-to-one relation between passengers and destination calls. Insufficient capacity is simply dealt with by relying on the passengers to reissue their calls. Therefore we do not need any book-keeping for dealing with the capacity.

   As a disadvantage, capacity is always overestimated which means that in high-load situations the planned schedules do not match the real behavior of the system. As a consequence, passengers have to reissue their calls which is annoying and results in long waiting times.

2. We take the limited elevator capacity into account, assuming that each call represents one passenger (i. e., that there is no misuse) and that all passengers board if there is sufficient capacity.

   This approach has the advantage that, if misuses are rare, the predicted behavior resembles the real behavior closely, thus avoiding unnecessary reissuing of calls. If there is in fact no misuse, every passenger planned to board can actually board.

   The drawback is that the remaining capacity of the elevator is underestimated since all potentially loaded passengers are assumed to be loaded. We need some additional book-keeping and the approach works best if there is a close correspondence between the destination calls and the passengers.

Additionally, the elevator control can be designed to be more robust against misuses. For instance, allowing to issue a destination call for several people at once eases the “increase-my-priority” misuse, since a single person can claim to be a group of people. Therefore it should not be used. Increasing the priority of a floor with many waiting

	conventional system	immediate assignment system	delayed assignment system
optimization opportunities			
	– destination information only available after boarding + reassignment of calls between elevators possible + no phantom stops	+ destination information for each passenger available with the call – reassignment of calls is not possible – possibility of phantom stops	+ destination information for each passenger available with the call + reassignment of calls between elevators possible – possibility of phantom stops
handling for passengers			
	– separate landing call and car call needed – passengers need to be attentive to catch their serving elevator [Sch90]	+ only one call needed + passengers can proceed right to their serving elevator	+ only one call needed – passengers need to be attentive to catch their serving elevator [Sch90]
possibilities for misuse			
	– erroneous usage (both buttons pushed) + priority cannot be increased	– erroneous usage (no button pushed) – priority can be increased by issuing multiple destination calls	– erroneous usage (no button pushed) – priority can be increased by issuing multiple destination calls

Table 2.1.: *Comparison of the advantages and disadvantages of the considered elevator control systems for various aspects.*

passengers is, on the other hand, very reasonable. In order not to sacrifice it, the priority might increase only sublinearly in the number of waiting passengers. A more extreme measure is to allow only one destination call per destination floor at each floor, which means that the elevator control cannot use capacity information. The choice between the two approaches discussed above and the use of further measures should be made on the kind of building the elevator group is in.

Table 2.1 summarizes the various advantages and disadvantages of the systems discussed here.

So far we did not discuss the precise goals that an elevator control should achieve. In most situations, the most important goal is a good quality of service. Traditionally, this is measured via the *waiting time* of a passenger, which is the time between the arrival

Grade of service	Percentage of calls answered in		Time to answer calls (s)	
	30s	60s	50%	90%
Excellent	> 75	> 98	20	45
Good	> 70	> 95	22.5	50
Fair	> 65	> 92	25	55
Poor/unacceptable	< 65	> 92	> 25	> 55

Table 2.2.: *Classification of the performance of an elevator system based on characteristics of the response time distribution according to [Bar02, p. 136].*

of a passenger and the time the serving elevator has arrived and opened the door. In a conventional system, this cannot be measured directly, since only the first landing call is registered. The time between the registration of the first landing call and the arrival of the elevator is called *response time* and is often used as a substitute for the real waiting times in conventional elevator control algorithms. Note that in destination call systems, we get a much more precise picture as the elevator control can better discriminate between passengers, possibly even between two passengers with the same destination floor. Moreover, a destination call system can determine the *travel times*, i. e., the time between the arrival of the passenger and the arrival of the serving elevator at the destination floor (again, with door opened), for the passengers it can discriminate. This is not possible in a conventional system, since we cannot match landing calls and car calls, which is the main reason why the travel time is not considered in conventional systems. We measure the quality of service in terms of the waiting time and, when applicable, the travel time distributions. Of course, the average and maximum values should be small, but the literature also gives target values for certain quantiles to classify the performance of a system, see e. g., Table 2.2. Another goal is to avoid having passengers to reissue their calls.

In high load situations short waiting or the travel times for the individual passenger might not be the most important goal. Instead, a high capacity or throughput is required, i. e., the elevators should serve as many passengers in a short period of time as possible in order to reduce the load. Another goal for an optimized elevator control that is becoming more and more important is energy efficiency. Since there are different goals depending on the traffic situation, an interesting question is how to dynamically adjust the planning to the current situation.

2.2. Some more background on elevator control

Control algorithms for conventional systems only decide about the assignment of landing calls to elevators [Bar02]. The set of landing calls assigned to an elevator is then served by a *roundtrip*: The elevator first serves all landing and car calls in its current travel direction, then reverses the direction and serves all calls in this direction, reverses direction again and serves the calls behind it in its original direction (see Figure 2.1). This strategy for

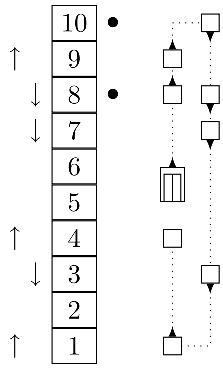


Figure 2.1.: *Construction of an elevator schedule for the assigned calls based on an elevator roundtrip. Arrows on the left indicate registered landing calls, whereas the bullets indicate registered car calls.*

scheduling an elevator to serve a set of calls is known as *Simplex Collective* or Collective for short.

Of course, restricting the kind of schedule considered for each elevator to a single roundtrip simplifies the control algorithms. Using the concept of a roundtrip, it is quite straightforward to compute the response times for each landing call which can be used to evaluate an assignment. However, there may be situations in which it is beneficial to postpone certain calls to the next roundtrip, but such schedules are not considered. For example, the elevator may be moving upwards passing floor 9, where very recently an up landing call has been registered, to pick up a long-waiting down landing call at floor 10. If the schedule is restricted to a single roundtrip, this landing call at floor 10 has to be delayed further. However, an advantage of using roundtrips is that it guarantees a bounded response time for every call. This is due to the fact that a call cannot be delayed indefinitely and the time for a roundtrip and thus the response time is bounded by the sum of times needed to travel from floor 1 to floor 2 to floor 3 and so on and back again.

Probably the main reason for sticking to roundtrips is that too few data is available to consider schedules consisting of multiple roundtrips. Since in destination call systems much more information is available earlier, there is no reason to keep this restriction. It will be interesting to see how much can be gained by considering more complex schedules.

Elevator control algorithms are most often evaluated by simulating various common traffic patterns. For instance, in an office building one can observe the following traffic patterns during the day (cf. Figure 2.2).

Up peak traffic In the morning, there is a high-intensity traffic from the entrance floors to the office floors. The duration of the peak traffic is often relatively short.

Interfloor traffic In the late morning and before noon, people are travelling inside the building from floor to floor. Start and destination floors are more or less uniformly distributed and there is a low traffic intensity.

Lunch traffic Around noon, people leave the building for having lunch, so there is

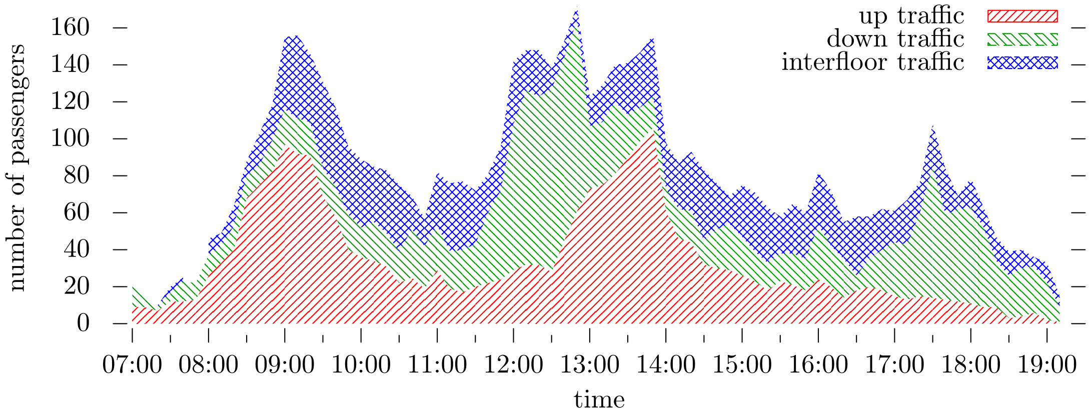


Figure 2.2.: *Typical passenger traffic in an office building. Shown is the number of passengers requiring service in each interval of ten minutes. One can clearly recognize the up peak phase in the morning around 9:00, the interfloor traffic phase before 12:00, lunch traffic between 12:00 and 14:00, afternoon interfloor traffic, and late afternoon down traffic starting at 17:30.*

an intense down traffic. Some time later, people return from lunch, generating intense up traffic which might overlap with the down traffic from other offices.

Down peak traffic In the afternoon, people leave the building. The down traffic is quite intense, but the intensity is lower than for the morning up peak, but spread over a longer time period.

Up peak traffic is considered to be the most demanding traffic situation [Bar02], so it is often used for dimensioning an elevator system. An important figure in this context is the handling capacity HC [Bar02]. The handling capacity is defined as the critical passenger arrival rate: If the passenger arrival rate is higher than the handling capacity, the system cannot cope with the resulting traffic and waiting times increase rapidly.

Consider an elevator system consisting of L lifts with capacity P each, serving a building with N floors. The handling capacity is determined by considering up peak traffic, i. e., all arriving passengers arrive at the main entrance floor and want to travel to the upper floors. Note that during up peak traffic, all a conventional elevator system can do is to send the lifts to the main entrance floor, loading as many passengers as possible, carry them to their floors, and return to the main entrance floor. The system performance therefore strongly depends on the *round trip time* RTT, which is the time needed to serve all loaded passengers and return to the main entrance floor. According to Barney [Bar02], the RTT is used to estimate the handling capacity via

$$HC = \frac{P \cdot L}{RTT}. \tag{2.1}$$

RTT is computed using the formula

$$RTT = 2Ht_v + (S+1)t_s + Pt_l,$$

where H is the highest floor reached, t_v the drive time needed to pass one floor, S is the number of stops needed to unload the passengers, t_s the time per stop, and t_l the time to unload one passenger. Note that only H and S depend on the actual load of the lift. For calculations, H is usually assumed to be N or $N-1$, which is justified if the lift capacity P is high. The number of stops S is calculated by assuming some distribution of the destination floors.

We already mentioned that in a conventional system, during up peak traffic there are basically no control decisions. The only parameter that can be controlled is the time an elevator waits on the main floor before starting its ride. Waiting longer may help to improve the utilization of the cabin and to some extent the waiting time of the passengers, but this will not reduce the number of stops. In contrast, in a destination call system the elevator control can distribute passengers between elevators, which can help a lot to reduce the number of stops during a single roundtrip. Therefore, destination call systems can achieve smaller round trip times and thus higher handling capacities. In Chapter 5 we will give some theoretical evidence for this.

2.3. Assumptions and requirements for elevator dispatches

The dispatching decisions of an elevator control algorithm are limited by certain requirements, that either arise from technical limitations, safety restrictions, or passenger expectations about the behavior of an elevator system. Since the passengers cannot be controlled nor supervised by the elevator control, the elevator dispatch needs to be based on certain assumptions on how a “reasonable” passenger behaves.

In the following, “call” means any of landing call, car call, or destination call, depending on the context. We will say that a landing call or destination call is *picked up* by an elevator that travels to the floor and signals that it is going to serve that call. After a destination call has been picked up, it becomes *loaded*. Similarly, registered car calls are also considered *loaded*. A loaded car call or destination call is *dropped* by an elevator by visiting the destination floor the first time after the call has been picked up (destination call) or registered (car call). A stop at a floor to pick up a call is called *pickup stop*, a stop for dropping a call *drop stop*. Note that this terminology mimics the transportation of a passenger, although in general there is no one-to-one correspondence between calls and passengers. Due to this, the elevator control can only measure the waiting and travel times of calls, not those of passengers. Thus we will only talk of waiting and travel times of calls from now on.

2.3.1. General requirements

The most important restrictions on the structure of a dispatch are consequences of the fact that passengers do not expect to travel against their desired direction [Bar02]. However, individual passengers might nevertheless choose to travel against their direction. Since the elevator control has no way to recognize this, the best we can do is to postulate the following assumptions on the behavior of passengers.

Assumption G.1. *A passenger never boards an elevator going in the wrong direction. In particular, a passenger boards the first elevator that was signaled to serve his call, i. e.,*

- *the first elevator with matching leaving direction in the conventional system,*
- *the assigned elevator when it leaves in the correct direction in the immediate assignment system, and*
- *the first elevator signalling to go to the desired destination floor in the delayed assignment system.*

Assumption G.2. *A passenger leaves an elevator at the first stop at the desired destination floor.*

Based on these assumptions, the elevator control can ensure that no "well-behaving" passenger is going into the wrong direction by fulfilling the following requirement.

Requirement G.3. *An elevator never transports calls in the wrong direction, i. e., the travel direction of the elevator is always equal to the desired direction of the loaded calls. Therefore, an elevator has to complete all requested drop stops before changing the direction.*

Clearly, the elevator control has to ensure that every passenger receives the required service at some point in time.

Requirement G.4. *For every registered landing call and destination call there is a matching pickup stop, i. e., one of the elevators is going to the start floor and leaving in the correct direction. Likewise, there is a drop stop for every loaded call.*

It may well happen that the capacity of the elevator doing a pickup stop is not sufficient to allow boarding of all waiting passengers. We assume that if there are passengers left that still require service, they are registering new calls.

Assumption G.5. *If a passenger could not board its assigned elevator due to lack of capacity, he reregisters his call again.*

The next requirement is more or less a psychological one: Passengers find it irrational if an elevator stops without loading or unloading passengers. As discussed before, such phantom stops cannot be avoided in general, at least not for destination call systems. However, the elevator control should ensure this requirement as far as possible, based on the available information.

Requirement G.6. *An elevator does not stop without loading or unloading potentially loaded passengers. Once an elevator starts decelerating, it stops at the next floor and, if passengers are loaded, opens its door before accelerating again.*

A special case where the elevator control can tell that no passengers board or leave the elevator is when none of the on-board passengers wants to leave at a floor and the elevator control detected via weighing that the elevator is full. In this situation the elevator must not stop at that floor.

Requirement G.7. *When the elevator is fully loaded, the elevator does not stop at floors where no passenger is going to leave.*

Apart from the restrictions implied by passenger service, there are also technical limitations.

Requirement G.8. *An elevator cannot reverse its direction without stopping.*

2.3.2. Additional assumptions

We will use some additional assumptions to simplify the model and the simulation used to compare different elevator control algorithms. Most of these assumptions can be relaxed and the model presented later still remains valid.

Assumption A.1. *Elevators can only stop at floors. In particular, empty elevators need to stop at a floor in order to change direction.*

This assumption implies that the arrival times of the stops can be computed from the time needed to travel from one floor to another. If stops between floors are allowed in order to change direction, the arrival time computation becomes more complex in these cases.

Assumption A.2. *In the simulation, passengers board and leave the elevator only if the door is fully opened.*

In reality, passengers might try to board an elevator whose door is not yet or not anymore fully opened. This may cause an additional delay on that floor, which should trigger an update of the schedule.

Once the door is closing, it is possible to decide whether or not the door should be reopened if another call on the current floor of the elevator is registered. However, loaded passengers might find the additional delay annoying. Of course, we have to reopen the door for security reasons if a passenger steps into the closing door. The only effect of reopening the door is that the execution of the remaining dispatch is delayed. In our simulation environment we allow to reopen the closing door at most a given number of times.

Assumption A.3. *As long as the door is closing it might be reopened, but at most k times on the same floor. Once the door has closed it cannot be reopened again and the elevator will start its ride.*

For evaluating the quality of a dispatch we need to compute or estimate the waiting and the travel time. Note that in the definition of the waiting and the travel time, we did not refer to the exact boarding time nor leaving time, which cannot be measured by the system. Instead, we used the time of arrival of the serving elevator to define them. This definition and the fact that we do not know the order in which passengers board or leave motivate the following assumption.

Assumption A.4. *From the elevator control point of view and in the simulation, passengers board and leave the elevator at the instant the doors are fully opened.*

To determine timing estimates for a dispatch, we need to specify how long an elevator waits on a floor before continuing its schedule. Since the transfer of passengers in and out of the elevator can take a significant amount it is too simplistic to assume a fixed time interval. Although in general we do not know exactly how many passengers board or leave at a floor we might have estimates for these numbers.

Assumption A.5. *The time an elevator waits at a floor with door open is at least a given time interval and increases with the (estimated) number of passengers that are picked up or dropped.*

We assume that an elevator stops at its final floor once it completed its schedule. An alternative to this is to employ some kind of *parking policy* [Bar02] to distribute the elevators over the building in order to achieve short waiting times for the next calls.

Assumption A.6. *If the elevator schedule is empty, it stays at the last visited floor with the doors opened.*

2.3.3. System-specific requirements

Apart from the general requirements and assumption that apply to all elevator control systems, there are further restrictions due to the type of information flow in the different systems.

Conventional system

Obviously, upon arrival of an elevator on a floor the elevator control needs to signal the direction of the landing call it is serving and than move the elevator accordingly.

Requirement CS. *If an elevator stops at a floor where at least one landing call is registered, the elevator control must fix and signal the leaving direction of the elevator before opening the doors. The elevator serves all car calls before changing direction.*

Note that the single-roundtrip based schedules employed in conventional systems naturally satisfy the last part of the requirement.

Destination call systems

We already mentioned that the elevator control in general does not know the exact set of passengers that boarded and therefore has to visit all corresponding destination floors. Of course, it also needs to signal which calls to serve.

Requirement DS. *After stopping at a floor, the elevator stops at all registered (and signalled) destination floors in its leaving direction before changing the direction.*

If an elevator stops at a floor where at least one destination call assigned to this elevator has been registered, the elevator control must fix and signal

- *the leaving direction of the elevator (immediate assignment system)*
- *the destination floors of the destination calls to be served by this elevator (delayed assignment system)*

before opening the doors.

For the delayed assignment system, the fixing of the destination floors to be signalled is typically done some time before the door is opened in order to allow passengers to proceed to the elevator. Note that not all floors the elevator is going to stop at need to be signalled. In fact, the elevator control may choose to signal no destination floor at all.

2.4. A general model for elevator group control

In this section we present a general model for elevator group control that captures the degrees of freedom and restrictions of the three types of systems considered so far: conventional systems, immediate assignment systems, and delayed assignment systems. In addition, it also captures destination call systems with full boarding control, i. e., the elevator control can decide and control which passengers board the elevator on each floor. Although this is an unrealistic setting for passenger elevators, it is interesting from a mathematical point of view for the following reasons:

- It may serve as a yardstick to provide “upper bounds” on the performance that can be achieved by any elevator control algorithm.
- It may be used to measure the “price of freedom”, i. e., the potential performance loss due to not having full boarding control.
- There may be applications where full boarding control can be realized, e. g., cargo elevators.

The main purpose of this model is to provide a unified framework for the algorithms in Section 2.5 and Chapter 3. It may also be used to study “mixed systems” [SP02, TYMR06], where destination calls are used at some of the floors only, whereas the other floors employ conventional landing calls.

We start by collecting the data that is available to the algorithms and thus are part of the snapshot problem.

2.4.1. The structure of the snapshot problem

The key idea to obtain a general model is to consider certain groups of calls as one *request*. A request describes a group of calls with the same start floor that will be picked up by the same elevator, i. e., the elevator control has no way to distribute calls of this request to different elevators. A request is served by picking up the corresponding calls and visiting each destination floor of the destination calls belonging to the request. In addition to serving picked up requests, an elevator may have *drop commitments*, i. e., floors to stop at before changing the direction, due to car calls or destination calls that have been picked up earlier.

The requests can be partitioned into the requests that are already assigned to an elevator e, denoted by $\mathcal{R}(e)$, and the ones that are still unassigned, denoted by $\mathcal{R}_u$. For elevator e, $\mathcal{F}(e)$ is the set of drop commitments. Formally, the task is now to assign the requests $\mathcal{R}_u$ to elevators and to find for each elevator a sequence of stops that fulfills the drop commitments $\mathcal{F}(e)$ and serves the requests $\mathcal{R}(e)$ and the requests from $\mathcal{R}_u$ assigned to that elevator. We call such a solution to the snapshot problem *dispatch*, which consists of one feasible *schedule* for each elevator specifying the stop sequence. The cost of a dispatch is the sum of the cost of its schedules, which are defined in the next subsection.

The exact set of calls belonging to a request and the sets of assigned requests depend as follows on the system considered. Figure 2.3 gives an example illustrating the differences.

conventional system There is one request for each landing call, which consists of this landing call only. A request is assigned to an elevator as soon as the elevator is going to stop at that floor and signalled the corresponding leaving direction.

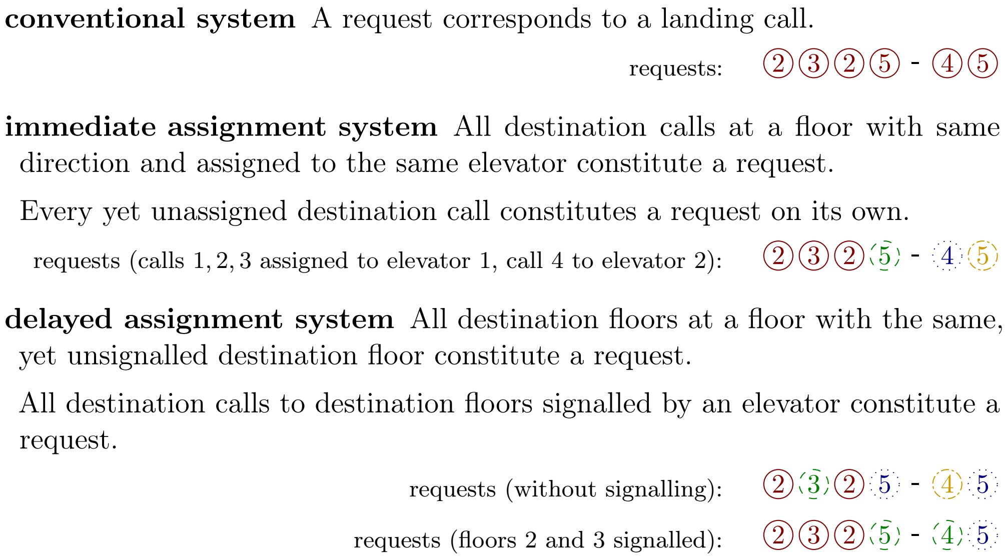
immediate assignment system There is one request for each floor/direction/elevator combination such that a destination call from that floor leaving in the direction has been assigned to the elevator. This request consists of all these destination calls that have been assigned to the elevator. Each such request belongs to $\mathcal{R}(e)$ for the assigned elevator e.

Another type of request arises from each yet unassigned destination call, which makes up a request of its own, reflecting that it still can be assigned to any elevator.

delayed assignment system For each elevator e that signalled destination floors to be served, there is a request in $\mathcal{R}(e)$ consisting of all the destination calls originating from the start floor and going to the destination floors signalled by that elevator.

Unassigned destination calls belong to the unassigned requests in $\mathcal{R}_u$; there is one such request per start floor / destination floor combination.

systems with full boarding control In a system with full boarding control, each destination call constitutes its own request. The assigment to an elevator is handled as in the system without boarding control. However, if the capacity does not suffice to pick up all matching requests, the elevator control algorithm may choose the ones to pick up.



conventional system A request corresponds to a landing call.

requests: ②③②⑤ - ④⑤

immediate assignment system All destination calls at a floor with same direction and assigned to the same elevator constitute a request.

Every yet unassigned destination call constitutes a request on its own.

requests (calls $1, 2, 3$ assigned to elevator 1, call 4 to elevator 2): ②③②⑤ - ④⑤

delayed assignment system All destination floors at a floor with the same, yet unsignalled destination floor constitute a request.

All destination calls to destination floors signalled by an elevator constitute a request.

requests (without signalling): ②③②⑤ - ④⑤

requests (floors 2 and 3 signalled): ②③②⑤ - ④⑤

Figure 2.3.: *Example illustrating the differences in addressing the passengers to serve that are modelled by the request concept. We assume that there are six passengers waiting on floor 1 with destination floors* $2, 3, 2, 5, 4, 5$*, respectively, in the order of arrival of the passengers. Furthermore, assume that passengers 5 and 6 have been registered after the last replanning step, which is depicted by the horizontal bar. The partitioning of the requests into calls is encoded by the dash pattern.*

In addition to the requests, the snapshot problem encompasses a set of distinguishable calls $\mathcal{C}$. Each call is described by its type (landing or destination call), release time, start floor and, if it is a destination call, destination floor.

Finally, $\mathcal{E}$ denotes the set of available elevators. For a single request ϱ or a set of requests S we use the notation $|\varrho|$ and $|S|$ to indicate the number of calls belonging to the requests and thus the required capacity.

2.4.2. A model for elevator schedules

We model the journey of an elevator e as a *schedule* $S = (s_0, \ldots, s_k)$, which is a sequence of stops s_i, $0 \leq i \leq k$. A stop s_i is characterized by the following data:

s_i.*floor* Floor the elevator stops at.

s_i.*direction* Direction in which the elevator continues its travel. Note that in a conventional system, this is not necessarily implied by the floor of the next stop. For instance, an elevator is to serve first an up and then a down landing call from the same floor, but no up car call has been given yet. Then there are successive stops at the same floor and the leaving direction is not implied by the next stop.

s_i.*current_calls* Set of currently loaded destination calls.

s_i.pickups Set of requests picked up at this stop. Used to determine the waiting time of the corresponding calls.

s_i.drops Set of loaded destination calls with this floor as destination. Used to determine the travel time of these destination calls.

s_i.drop_floors Set of pending drop commitments, i. e., floors to visit due to picked up destination calls or car calls.

s_i.arrival (Estimated) Time of arrival at this floor.

s_i.departure (Estimated) Time of departure from this floor.

Note that only destination calls are “trackable” in the sense that we can associate a stop at the destination floor to the call and thus determine the travel time of the corresponding passenger. This is why we only keep track of destination calls in *s_i.current_calls*.

Feasibility of a schedule At any point in time, elevator e is either halting at a floor or travelling towards a floor. Let $f_0(e)$ be this floor. Depending on its current state, elevator e may not stop on every floor. For instance, it may currently bypass floor 5 at maximum speed, but deceleration takes too long to stop on floor 6, so floor 7 is the next floor it can stop at. We denote by $\mathcal{F}_i(e)$ the set of floors that are admissible for the first stop in a schedule. If the elevator is halting at floor $f_0(e)$, we have $\mathcal{F}_i(e) = \{f_0(e)\}$ and it is not possible to have the first stop at a floor different from $f_0(e)$. In case the elevator is full, the only floor allowed as a first stop is the next floor of the drop commitments.

Similarly, the leaving direction of the first stop might already be fixed due to drop commitments or because the leaving direction has already been signalled to the passengers. Denote by $D_0(e)$ the set of feasible leaving directions for the first stop. Finally, $\mathcal{C}(e)$ denotes the set of destination calls that are currently loaded by elevator e.

According to Assumption G.1, passengers board the elevator that was signalled to serve them on its first stop in the matching direction. To model this, we let $P(e, f, d)$ denote the set of requests that are to be picked up by elevator e at its first stop on floor f with leaving direction d. The first such stop of elevator e then has to be followed by stops according to the additional drop commitments implied by all requests in $P(e, f, d)$. The definition of $P(e, f, d)$ depends as follows on the considered system.

conventional system All sets $P(e, f, d)$ are empty. The rationale for this is that in a conventional system there are no implied drop commitments, since the elevator control does not know the destination floors yet.

immediate assignment system $P(e, f, d)$ is the set of requests leaving floor f that have been assigned to elevator e.

delayed assignment system $P(e, f, d)$ is again the set of requests leaving floor f that have been assigned to elevator e. Note that in a delayed assignment system, requests are only assigned to an elevator that is already approaching the floor and signalled the corresponding destination floors.

systems with full boarding control All sets $P(e, f, d)$ are empty, since the elevator can freely choose which passengers/requests are picked up.

We now have all the ingredients to formally define feasibility of a schedule $S = (s_0, \ldots, s_k)$ for elevator e. For convenience, we denote by $P(s_i)$ the set of requests already picked up by the schedule up to, but not including, stop s_i. Note that $P(s_0) = \emptyset$.

The most important feasibility restrictions are due to the requirement that (well-behaving) passengers must not be transported in the wrong direction. This implies restrictions on the sequence of floors and the leaving directions of the stops. For the initial stop s_0 we have the following conditions:

- The floor $s_0.\mathit{floor}$ is one of the floors of $\mathcal{F}_i(e)$.
- The leaving direction $s_0.\mathit{direction}$ is one of $D_0(e)$ if $s_0.\mathit{floor} = f_0(e)$ or the direction towards this stop if $\mathcal{F}(e)$ is nonempty. Otherwise the direction may be arbitrary, but of course matching the requests picked up.
- The drop commitments $s_0.\mathit{drop\_floors}$ are the floors from $\mathcal{F}(e)$ without $s_0.\mathit{floor}$ plus the destination floors of the destination calls picked up at this stop.

The analogues of these conditions for the later stops are a bit more involved.

- There are the following cases depending on the pending drop commitments:

$s_i.\mathit{drop\_floors} \neq \emptyset$: $s_{i+1}.\mathit{floor}$ is between $s_i.\mathit{floor}$ and the next floor from $s_i.\mathit{drop\_floors}$ in the leaving direction of s_i or equal to this floor.

The leaving direction $s_{i+1}.\mathit{direction}$ is the same as on the last stop, if $s_i.\mathit{drop\_floors}$ consists of at least two floors or it is one floor, but $s_{i+1}.\mathit{floor}$ is before this floor. Otherwise the direction may be arbitrary.

$s_i.\mathit{drop\_floors} = \emptyset$: $s_{i+1}.\mathit{floor}$ and the leaving direction are arbitrary.

- The drop commitments $s_{i+1}.\mathit{drop\_floors}$ are the same as $s_i.\mathit{drop\_floors}$ without $s_{i+1}.\mathit{floor}$ plus the destination floors of the destination calls picked up at this stop.

In addition to the conditions due to passenger directions there are conditions on the sets of picked up requests and the sets of current and dropped destination calls.

- $s_i.\mathit{pickups}$ contains all the requests in $P(e, s_i.\mathit{floor}, s_i.\mathit{direction}) \setminus P(s_i)$. Moreover, $s_i.\mathit{pickups}$ does not contain any request that is assigned to a different elevator and, of course, each request starts at floor $s_i.\mathit{floor}$ and travels in direction $s_i.\mathit{direction}$.
- $s_i.\mathit{current\_calls}$ is the set of current calls of the preceding stop ($\mathcal{C}(e)$ for s_0) that have a different destination floor than $s_i.\mathit{floor}$ plus the destination calls corresponding to $s_i.\mathit{pickups}$. Note that these are all calls that could have been picked up if the elevator capacity was not limited.
- $s_i.\mathit{drops}$ is the set of current calls of the preceding stop ($\mathcal{C}(e)$ for s_0) that have destination floor $s_i.\mathit{floor}$.

So far we did not take the limited elevator capacity into account, i. e., we assumed unlimited elevator capacity. We now introduce an optional soft constraint to model limited elevator capacity as sketched in Section 2.1 to avoid having the passengers to reissue their calls. In general, we cannot avoid insufficient elevator capacity so we cannot strictly forbid capacity violations. Instead, we penalize each call of a request that exceeds the elevator capacity *at the pickup stop of the request* with additional cost c_{capacity}. Denoting the capacity of elevator e by $\kappa(e)$, the capacity penalty cost at stop s are

$$c_{\text{capacity}} \cdot \max\bigl\{0, |s.\mathit{current\_calls}| + |s.\mathit{pickups}| - \kappa(e)\bigr\}.$$

Now the overall schedule is feasible if and only if

- every stop is feasible,
- there are no two successive stops on the same floor with the same leaving direction,
- all requests assigned to the elevator are served, i. e., $\mathcal{R}(e) \subseteq P(s_k)$,
- finally, all calls and drop commitments have been served, i. e., $s_k.\mathit{current\_calls} = \emptyset$ and $s_k.\mathit{drop\_floors} = \emptyset$.

Timing of a schedule A schedule allows to evaluate waiting times for all calls of each request and the travel time of each destination call and thus facilitates the computation of an objective value based on these times. Essentially, we are computing these times for each call that can be discriminated by the elevator control, i. e., we assume that each call corresponds to exactly one passenger.

Let e denote the elevator to execute the schedule S and let $f \in \mathcal{F}_i(e)$ be a floor elevator e can stop at next. Denote by $\tau_{\text{arr}}(e, f)$ the time elevator e will stop at floor f and by $\tau_{\text{drv}}(e, f_1, f_2)$ the floor-to-floor travel time from floor f_1 to floor f_2 of elevator e, including the time to close and reopen the door. Moreover, let τ_{stop} be the minimum waiting time on a floor and τ_{load} the transfer time of a passenger, i. e., the time a passenger needs for boarding or leaving. The arrival and departure times of each stop are then given by the equations

$$\begin{aligned}
s_0.\mathit{arrival} &= \tau_{\text{arr}}(e, s_0.\mathit{floor}), && \text{provided } s_0.\mathit{floor} \in \mathcal{F}_i(e)\\
s_0.\mathit{departure} &= s_0.\mathit{arrival} + \max\{\tau_{\text{stop}}, \tau_{\text{load}}\bigl(|s_0.\mathit{pickups}| + |s_0.\mathit{drops}|\bigr)\}\\
s_{i+1}.\mathit{arrival} &= s_i.\mathit{departure} + \tau_{\text{drv}}(e, s_i.\mathit{floor}, s_{i+1}.\mathit{floor}),\\
s_{i+1}.\mathit{departure} &= s_{i+1}.\mathit{arrival} + \max\{\tau_{\text{stop}}, \tau_{\text{load}}\bigl(|s_{i+1}.\mathit{pickups}| + |s_{i+1}.\mathit{drops}|\bigr)\}
\end{aligned}$$

Note that the elevator waiting time is essentially proportional to the number of calls transfered. For conventional systems this is constant since we do not use any estimates for the number of passengers belonging to a request.

Based on the arrival times it is now easy to compute the waiting times for the calls picked up according to $s_i.\mathit{pickups}$ and the travel times for the calls in $s_i.\mathit{drops}$.

Cost of a schedule Different objective or cost functions can be defined based on this model for elevator schedules. In the model, waiting and travel times of each call are computed. From these it is possible to compute the *ride time*, i. e., the time a passenger spends inside the elevator, as the difference between the travel and waiting time. Note that ride and travel times are only defined for destination calls and assumed to be zero for landing calls. This setup allows general objective functions capturing the quality of service for passengers, for instance a weighted sum of waiting times, ride times, and travel times for each call. It is also possible to incorporate the completion time for each elevator in the objective, expressing that the requests should be served as fast as possible.

Formally, let $t_{\text{wait}}(c)$, $t_{\text{ride}}(c)$, and $t_{\text{travel}}(c)$, respectively, denote the waiting, ride, and travel time of call c. The simplest objective function is linear, i. e., the cost for serving call c are $c_{\text{wait}}(c)t_{\text{wait}}(c) + c_{\text{ride}}(c)t_{\text{ride}}(c) + c_{\text{travel}}(c)t_{\text{travel}}(c)$ where $c_{\text{wait}}(c)$, $c_{\text{ride}}(c)$, and $c_{\text{travel}}(c)$ are cost coefficients associated with call c. A disadvantage of linear cost is that for calls with the same coefficient only the total time matters. For instance, suppose that calls c_1 and c_2 experience waiting times of 5 and 55 seconds in one dispatch, and of 35 and 25 seconds in another. With a linear objective, both alternatives incur the same cost, but the second dispatch would be prefered in practice since the maximum waiting time is smaller.

For this reason, we also consider a quadratic objective function. The cost of serving call c is then $c_{\text{wait}}(c)t_{\text{wait}}(c)^2 + c_{\text{ride}}(c)t_{\text{ride}}(c)^2 + c_{\text{travel}}(c)t_{\text{travel}}(c)^2$. Assuming $c_{\text{wait}}(c_1) = c_{\text{wait}}(c_2) = 1$ in the above example, the costs are 3050 vs. 1850, so the second schedule would be prefered. However, a quadratic objective is often less desirable from a computational point of view. A compromise between the linear and the quadratic objective can be obtained by using a linear objective, but choosing the cost coefficients $c_{\text{wait}}(c)$, $c_{\text{ride}}(c)$, and $c_{\text{travel}}(c)$ based on the age of c such that they increase with age. This approach was used by Tanaka et al. [TUA05a]. It avoids the above weakness of the linear objective if calls c_1 and c_2 have different release times, but does not help if the release times are the same.

We emphasize that for systems without boarding control, the service cost are included for *all* signalled calls at each stop, no matter whether the elevator capacity is sufficient or not. Of course, this leads to underestimating the waiting time of a passenger and distorts the evaluation of schedules, but we cannot avoid this. However, depending on the kind of system used the elevator control may be able to reduce capacity violations by assigning calls to elevators with sufficient capacity.

In addition to these service quality cost capacity penalty cost are incurred as described earlier. However, it turned out that the service quality cost and the capacity penalty alone do not lead to a good grouping of the destinations calls according to their destination floor. Figure 2.4 gives an example of the elevator movements in a simulation of high load up peak traffic. As we can see, the elevator performs a lot of stops before returning to the main entrance floor 1 again, giving rise to long round trip times, resulting in bad service. These long round trip times could be avoided by an improved grouping according to destination floors. To achieve this, we introduce a round trip time penalty function $c_{\text{RTT}} \colon \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$, that assigns a round trip time t a penalty of $c_{\text{RTT}}(t)$. This RTT penalty is incurred for every roundtrip starting and ending at the main entrance floor. If

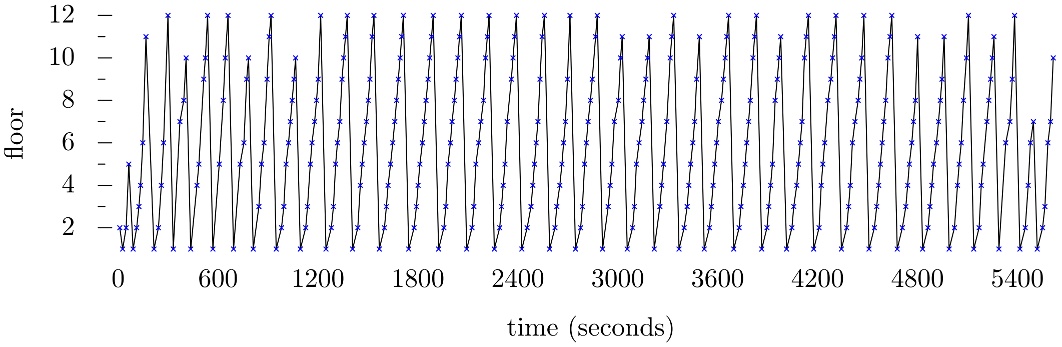


Figure 2.4.: *Movement diagram of an elevator during high load up peak traffic. The simulation was done with the capacity penalty enabled.*

the schedule of an elevator does not end at the main entrance floor, a corresponding stop is added for the purposes of computing the round trip time.

Finally, Table 2.3 summarizes all the data constituting a snapshot problem.

2.5. Heuristic algorithms for group elevator control

Almost any algorithmic paradigm has been applied to the control of elevators. For instance, there are algorithms based on reinforcement learning [CB98], genetic or evolutionary algorithms [SSE03, TY06], artificial intelligence methods [KO02] as well as exact ones based on dynamic programming [Glo70] and Branch & Bound [TUA05b]. Fuzzy logic has been used to switch the scheduling policy according to the observed traffic situation [KSLKK98]. However, very few of the scientific publications on elevator control seem to have had practical impact. In this section we will give a brief overview on algorithmic concepts and ideas that are known to be used or have been used in practice. We will only consider non-exact or heuristic approaches; algorithms based on exact solution methods will be described in-depth in Chapter 3.

2.5.1. Classical elevator control

Barney [Bar02, Chapter 10] describes some “classical” elevator control algorithms, i. e., algorithms that were implemented using electromechanical hardware before microprocessors were used. As these are algorithms for conventional systems, each single elevator is scheduled according to the Collective strategy mentioned in Section 2.2, i. e., it serves assigned calls on a single roundtrip. The main task of the elevator control algorithm is to decide which landing call will be served by which elevators.

The landing call allocation is done according to fixed rules. The simplest rule is to allocate each landing call to the next elevator, which defines the NearestCar algorithm. Barney reports that this algorithm gives reasonable performance only for small buildings with few elevators.

symbol	meaning
$\mathcal{R}$	set of all requests
$\mathcal{C}$	set of registered and unserved landing and destination calls
$\mathcal{E}$	set of available elevators
$\mathcal{R}(e)$	set of requests assigned to elevator e
$\mathcal{R}_u$	set of requests that are not assigned to any elevator
$\mathcal{F}(e)$	floors elevator e needs to stop at due to car or destination calls before changing direction
$f_0(e)$	floor elevator e is travelling to or halting at
$\mathcal{F}_i(e)$	set of floors elevator e still can stop at before reaching floor $f_0(e)$
$D_0(e)$	set of possible leaving directions for elevator e
$\kappa(e)$	(rated) capacity of elevator e
$\mathcal{C}(e)$	set of loaded destination calls of elevator e
$P(e, f, d)$	set of requests picked up by elevator e on its first stop on floor f with leaving direction d
$\tau_{\mathrm{arr}}(e, f)$	time of arrival of elevator e at floor $f \in \mathcal{F}_i(e)$
$\tau_{\mathrm{drv}}(e, f_1, f_2)$	floor-to-floor travel time from floor f_1 to floor f_2 for elevator e
τ_{stop}	minimum stopping time for all elevators
τ_{load}	transfer time of a passenger
$c_{\mathrm{wait}}(c)$	waiting time cost coefficient of call c
$c_{\mathrm{ride}}(c)$	ride time cost coefficient of call c
$c_{\mathrm{travel}}(c)$	travel time cost coefficient of call c
c_{capacity}	penalty for each picked up call that exceeds elevator capacity
c_{RTT}	round trip time penalty function

Table 2.3.: *Overview of input data for the elevator control snapshot problem.*

More advanced rules employ *sectoring*: The floors of the building are divided into sectors, which are to be served by one elevator each. There are several ways on how to assign elevators to sectors.

In the FSO sectoring scheme there are as many sectors as elevators. A sector is either assigned to an elevator or it is vacant. Once an elevator is present in a vacant sector it is assigned to that sector. An elevator serves the landing call in its sector and the ones in vacant sectors above it; the lowermost elevator is responsible for its sector and all sectors below it. If an elevator has no more calls to serve, it is assigned to one of the vacant sectors. Thus the algorithm tries to distribute the elevators evenly in the building. According to Barney, it achieves good performance for up peak and interfloor traffic.

Another way to employ the sectoring concept is the dynamic sectoring system DS. Here, the sectors are dynamically defined based on the positions of the elevators. The sector of an elevator extends from the current floor of the elevator to the floor of the next elevator travelling in the same direction. If there is no other elevator in that direction, the sector "wraps around" and continues in the opposite direction. Each elevator serves landing calls with matching travel directions in its sector. An additional strategy is used to maintain *free* elevators which do not take part in this sectoring scheme, but provide additional capacity to be send to heavy traffic sectors. Barney states that the performance for up peak and interfloor is very good, the one for down peak rather poor.

2.5.2. Computer Group Control

In this section we describe a variant of the *Computer Group Control (*CGC*)* algorithm sketched by Barney [Bar02, Case Study 16] in greater detail, as it will be used in our computational comparisons. Unlike the classical control algorithms, this algorithm is not rule-based but uses a cost function to decide the assignment of landing calls to elevators.

The CGC algorithms inserts requests (i. e., landing calls) successively in a set of elevator schedules serving all car calls of each elevator plus the requests assigned so far. The cost of request ϱ incurred by assigning it to elevator e is the waiting time of ϱ in the resulting single-roundtrip schedule for e. In order to compute reasonable estimates for this, CGC assumes that the (unknown) destination floor of each request is halfway between the start floor and the last floor in the request direction. Note that the schedule model of Section 2.4 may be used to evaluate the resulting waiting times for each request.

In general, request ϱ is assigned to the elevator with minimal cost, with two exceptions based on a parameter HTT called "high threshold time". In order to reduce the number of stops, request ϱ is allocated to an elevator with a car call at the start floor of ϱ, provided it does not get waiting time greater than HTT due to this allocation. Moreover, request ϱ is not allocated to the minimum cost elevator if the waiting time of another request increases beyond HTT. If this cannot be avoided, ϱ is allocated to the minimum cost elevator anyway. The details are summarized in Algorithm 2.1.

Following the recommendations of Barney, HTT is thrice the average system response time, which is measured over an interval of 90 seconds. Thus HTT reflects the current load situation of the system.

2.5.3. Genetic algorithms

Genetic algorithms are a well-known metaheuristic for solving optimization problems that can relatively easy be applied to elevator control. In fact, KONE has several publications [SSE03, TY06] as well as patents [TY99b, TY99a, TY05, TYMR06] on using genetic algorithms for this purpose. The genetic algorithm decides the assignment of landing calls to elevators and evaluates an assignment by evaluating the resulting single round trip schedule, thus following the scheme described in Section 2.2. However, the publications do not give an indication on the details of the genetic algorithm used, e. g., the population size, the mutation rate, the crossover operation and so on.

2.5.4. A cost-based best-insertion heuristic

In this section we describe an elevator control algorithm for destination call systems designed for Kollmorgen Steuerungstechnik, our partner from industry [HT08]. A variant

```
procedure CGC
    1. Sort the requests by decreasing waiting time; assume that the order is $\varrho_1, \ldots, \varrho_n$.
    2. Initialize the schedules of the elevators according to their car calls.
    3. For each request $\varrho \leftarrow \varrho_1, \ldots, \varrho_n$:
        a) If there is an elevator $e$ with a car call at the start floor of $\varrho$ and the resulting waiting time for $\varrho$ is at most HTT, then assign $\varrho$ to elevator $e$ and update the schedule of elevator $e$. Continue with the next request.
        b) Allocate $\varrho$ to the elevator $e$ that realizes the lowest waiting time for $\varrho$, such that no other request served by $e$ experience waiting time more than HTT. If there is no such elevator, allocate $\varrho$ to the elevator with the lowest waiting time for $\varrho$, updating $e$'s schedule accordingly.
end procedure
```

Algorithm 2.1: *Overview of the implemented* CGC *algorithm.*

of it has been implemented by Kollmorgen Steuerungstechnik and is now running in their elevator controls. However, the original algorithm was designed for immediate assignment systems whereas this version is based on the generic request model from Section 2.4 and thus also suitable for delayed assignment systems. Hence the semantics for the immediate assignment are slightly different to the original algorithm described in [HT08].

The high-level algorithm

Kollmorgen Steuerungstechnik posed severe restrictions on the computing environment for the elevator control algorithm: The algorithm is supposed to run on embedded microcontrollers with computation times of at most 200 ms using not more than 200 kB of memory. Thus computational resources are very scarce, rendering exact optimization methods infeasible. We therefore propose an insertion heuristic. Insertion heuristics are well-known for e. g., the Travelling Salesman Problem. However, the structure of a tour for an elevator is much more complex, making the insertion operation non-trivial.

Our algorithm BestInsert relies on the schedule model described in the last section. In contrast to other heuristic destination call algorithms it exploits the possibility of using multiple roundtrips. It assumes a set of schedules that already serve a subset R of the requests and then successively assigns the requests $\mathcal{R} \setminus R$ to elevators. Usually, R is the set of requests that were already known and scheduled in the last replanning step. In order to assign requests to elevators, it considers suitable insertion positions and chooses, for each elevator, the one with minimum additional cost. The elevator with overall minimum additional cost then receives the request. The outline of BestInsert is given in Algorithm 2.2.

```
procedure BestInsert (subset $R$ of the requests $\mathcal{R}$)
    1. Assume that the schedules of the elevators are initialized such
       that all requests $R$ are already served.
    2. Sort the requests in $\mathcal{R} \setminus R$ by decreasing waiting time; assume
       that the order is $\varrho_1, \dots, \varrho_n$.
    3. For each request $\varrho \leftarrow \varrho_1, \dots, \varrho_n$:
        a) For each elevator $e \in \mathcal{E}$: determine the minimum insertion
           cost $c(e, \varrho)$ as follows:
            i. Let $S = (s_0, \dots, s_k)$ be the current tentative schedule of
               elevator $e$.
            ii. For each feasible insertion pair $(s_i, s_j)$, $i \leq j$, compute
                the new tentative schedule $S'$ via

                    $S' \leftarrow \mathsf{AddRequest}(S, s_i, s_j, \varrho)$.

                The insertion cost for this insertion pair are $c(S') - c(S)$.
                Update $c(e, \varrho)$ if this insertion is cheaper.
        b) Assign request $\varrho$ to the elevator $e$ minimizing $c(e, \varrho)$.
        c) Replace the tentative schedule of $e$ with the one corresponding
           to the cheapest insertion.
end procedure
```

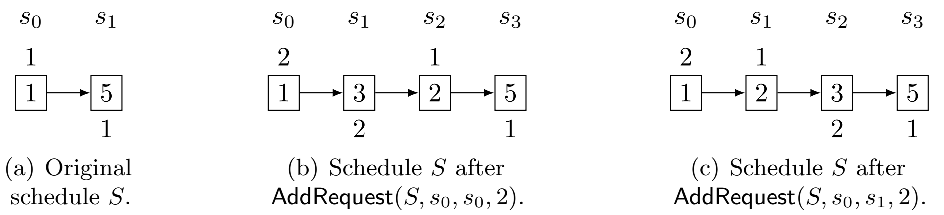
Algorithm 2.2: *Overview of the* BestInsert *algorithm.*

The main ingredient of BestInsert is the procedure AddRequest(S, s_i, s_j, ϱ), which inserts request ϱ at positions s_i and s_j in the schedule S. The stops s_i and s_j indicate the insertion position of the stop for picking up and dropping, respectively. The details of AddRequest are rather involved and will be explained in the next section. We note that BestInsert in this version is computationally too expensive; the variant implemented by Kollmorgen Steuerungstechnik therefore considers only a restricted set of insertion positions.

The insertion procedure

The general semantics of AddRequest(S, s_i, s_j, ϱ) are that the pickup stop for ϱ is either s_i itself or inserted before it. Likewise, the stop for dropping the calls of ϱ with the destination floor that is furthest away is either s_j itself or inserted before s_j. Figure 2.5 gives an example, showing that in order to support schedules with multiple roundtrips two stops for specifying the insertion are needed.

AddRequest is non-trivial due to the cases that may arise. For instance, it may be necessary to split an existing stop into two new stops to avoid direction changes for passengers, see Figure 2.6 for an example. Worse, it may happen that a newly inserted

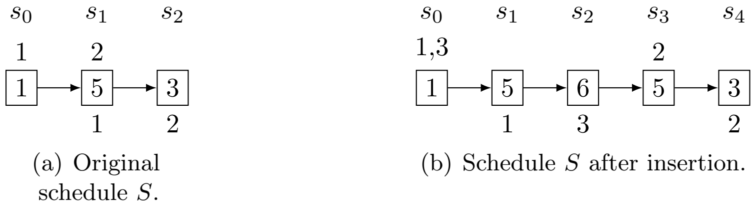


(a) Original schedule S.

(b) Schedule S after AddRequest$(S, s_0, s_0, 2)$.

(c) Schedule S after AddRequest$(S, s_0, s_1, 2)$.

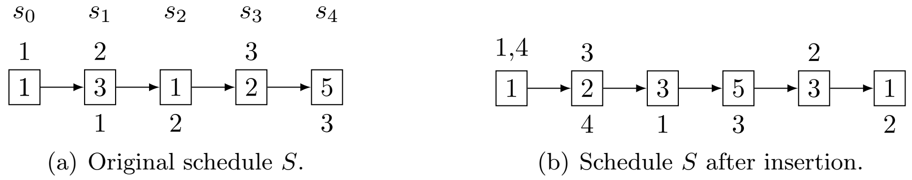
Figure 2.5.: *Example illustrating the semantics of* AddRequest. *Our graphical notation represents a stop by a boxed floor number. Numbers above or below a stop indicate requests picked up or dropped, respectively. The original schedule* S *serves only the request* $1\colon 2 \rightarrow 5$. *The schedule is extended by inserting request* $2\colon 1 \rightarrow 3$.



(a) Original schedule S.

(b) Schedule S after insertion.

Figure 2.6.: *Inserting request* $3\colon 1 \rightarrow 6$ *via* AddRequest$(S, s_0, s_1, 3)$ *requires to split stop* s_1.



(a) Original schedule S.

(b) Schedule S after insertion.

Figure 2.7.: *Inserting request* $4\colon 1 \rightarrow 2$ *via* AddRequest$(S, s_0, s_1, 4)$ *requires a non-trivial adjustment.*

stop becomes the first stop on a floor with this leaving direction. According to our schedule model, all requests in the respective set $P(e, f, d)$ have to be picked up at this new stop, leading to corresponding stops for dropping them. This means that it is necessary to adjust the tour, maybe changing its structure quite substantially. As an example, consider the tour in Figure 2.7(a) and suppose we want to insert the request $4\colon 1 \rightarrow 2$ via the stops s_0 and s_1 (which is the only feasible insertion position). We need to create a new stop at floor 2, leaving upwards due to request 1. But then request 3 will enter and we need to go to floor 5 before we can leave floor 3 downwards. Therefore we need to adjust the tour, keeping it close to the original one; the resulting tour is shown in Figure 2.7(b). There are more complex cases to take into account.

Of course, the schedule AddRequest generates has to be feasible. The most important requirements that have to be ensured are

- The requests in $P(e, f, d)$ are picked up at the first matching stop.

- The drop floors of the loaded calls are visited before changing the direction.
- The new schedule serves exactly the requests of the old schedule plus the new request.

The tour is adjusted such that these requirements are established and the resulting tour is as close as possible to the original one. In fact, if all $P(e, f, d)$ are empty, AddRequest inserts just the stops needed to pickup and drop ϱ.

Algorithm 2.3 shows a rough outline of the AddRequest procedure. For convenience, the algorithm description assumes an additional (sentinel) stop s_∞ after s_k. To ensure pickup of the requests in $P(e, f, d)$ we first make a copy $Q(e, f, d)$ which keeps track of the requests still to pick up on the first matching stop. We call a stop *unnecessary*, if no calls are picked up or dropped there. Due to the insertion of stops and the subsequent service of the corresponding drop floors it may happen that requests receive service earlier and later stops become unnecessary.

After having created pickup and drop stops, we adjust the schedule to maintain feasibility (Steps 5 to 5). We have the following invariant: The partial schedule $(s_0, \ldots, s)$ is feasible and the leaving direction of s has already been determined, whereas the remaining schedule $(s', \ldots s_k)$ still needs to be adjusted. Moreover, the set of requests served is exactly that of the old schedule plus the new request. This invariant is established by Step 5. It is understood that while scanning the remainder of the tour for the first stop s' that is still necessary unnecessary stops are removed from the schedule. Thus s is always the predecessor of s'.

In order to push the invariant to the next stop we need to determine the next stopping floor f and its leaving direction d. The next stop floor is either prescribed by the drop floors of the current stop or by the next necessary stop, if any. Note that if there is no necessary stop any more, there are also no drop floors to visit and the procedure terminates. Similar to the next stopping floor, the leaving direction is also either determined by the drop floors of the current stop s or by the next necessary stop s'. Finally, a stop corresponding to f and d is created (or an existing one reused), ensuring that the requests $Q(e, f, d)$ are picked up. Observe that no necessary stop is deleted, so stops with requests not picked up earlier will be reused eventually. Thus all requests served by the original schedule are included in the new schedule.

```
procedure AddRequest ($S, s_i, s_j, \varrho$)
    1. Let $Q(e, f, d)$ be a copy of $P(e, f, d)$ for all $e$, $f$, and $d$.
    2. For all stops from $s_0$ to the predecessor of $s_i$: Reset the corresponding $Q(e, f, d)$ to the empty set.
    3. Create pickup stop: Either create a new stop or reuse $s_i$, splitting the stop if necessary. The requests to pickup are $Q(e, f, d)$ plus $\varrho$, where $f$ and $d$ are the start floor and the direction of $\varrho$, respectively.
    4. Create drop stop: Either create a new stop or reuse $s_i$, splitting the stop if necessary. The requests to pickup are not yet determined.
    5. Let $s$ be the predecessor of $s_i$ (or the first copy of the split stop in case of splits) or $s_0$ if $s_i$ is the first stop. Let $s'$ be the first still necessary stop after $s$.
    6. Avoid successive stops on same floor: If $s$ and $s'$ are on the same floor, but $s$ has no drop floors, then transfer the pickups of $s'$ to $s$ and replace $s'$ by the next stop that is still necessary.
    7. While $s' \neq s_\infty$ or stop $s$ has drop floors:
        a) Determine the next stopping floor $f$:
            i. If stop $s$ has drop floors: Let $f$ be the nearest floor of these. Let $s'$ be the next stop after $s$ that is either necessary or whose floor is before floor $f$. If $s'.\mathit{floor}$ is before $f$, set $f \leftarrow s'.\mathit{floor}$.
            ii. Otherwise: Let $s'$ be the next necessary stop after $s$. If $s' = s_\infty$, stop, else set $f \leftarrow s'.\mathit{floor}$.
        b) Determine leaving direction $d$ for next floor:
            i. Let $F$ be the drop floors of stop $s$ without $f$.
            ii. If $F$ is nonempty: $d$ is the current direction.
            iii. Otherwise: Let $s'$ be the next necessary stop after $s$. The direction $d$ is STOP, if $s' = s_\infty$. It is the leaving direction of $s'$ if $s'.\mathit{floor} = f$ and the direction from $f$ to $s'.\mathit{floor}$ otherwise.
        c) Create update next stop:
            i. If $s'$ matches $f$ and $d$: Remove the requests that have been picked up earlier from $s'.\mathit{pickups}$. Let $s \leftarrow s'$ and let $s'$ be the first necessary stop after $s$.
            ii. Otherwise: Let $s$ be a new stop at floor $f$ with leaving direction $d$ and picked up requests $Q(e, f, d)$.
        d) Reset $Q(e, f, d)$ to the empty set.
end procedure
```

Algorithm 2.3: *Overview of the* AddRequest *procedure.*

Chapter 3.

Exact elevator group control algorithms

In this chapter we develop and study *exact reoptimization algorithms*, which are algorithms that solve the snapshot problem exactly, i. e., they compute a provably optimal solution, maybe up to a certain optimality gap. These algorithms are interesting since they allow to investigate how much the performance of an elevator system may be improved by using control algorithms that find solutions with a higher quality. Moreover, they may be the basis for developing high-quality heuristics to be used in practice.

Let us emphasize that computing an optimal dispatch for each snapshot does not necessarily lead to a good or “optimal” online control algorithm. This is due to the online nature of the control problem: The decisions of an optimal schedule for a certain snapshot problem may turn out to be bad in view of the realized future. In fact, it is hard to define a practically useful notion of “optimal online algorithm”. In theory, one usually calls an online algorithm optimal, if it attains the optimal *competitive ratio* – see Chapter 4 for a discussion of this concept. This notion is, however, often not useful for designing practically relevant online algorithms, since for more complex problems it often fails to discriminate between algorithms that perform very different in practice. Another approach to define optimality is to assume some stochastic model; an algorithm is then said to be optimal if it minimizes the *expectation* of some performance measure. This approach has indeed been applied to the elevator control problem for conventional systems [CB98, PC97, PC98, NB03]. For special traffic situations like up peak it is possible to derive optimal control algorithms, though this can only be achieved since the number of control actions is low and there are relatively few input parameters. The situation is much more complex for destination call systems. If optimality of an algorithm cannot be proved, it is still possible to compute optimal control decisions w. r. t. to the stochastic model. Such stochastic optimization algorithms tend to require much more computational effort for their solution than their deterministic counterparts. Another drawback of the stochastic approach is that often no reliable statistical data is available, so it is hard to construct an appropriate stochastic model. We will not consider stochastic optimization algorithms in this thesis.

It is known that several simple versions and special cases of the elevator scheduling problem are already NP-hard. For instance, Seckinger and Koehler [SK99] showed that deciding whether or not there is a feasible schedule for a single elevator which serves all requests with at most k stops is NP-hard. The elevator scheduling problem may be viewed as an instance of the Dial-a-Ride problem, where one or more servers with capacity k have to serve transportation requests in a graph. Each transportation request specifies a source and a target node; usually the objective is to minimize the total travel distance which is equivalent to the makespan in the single server case. Krumke et al. [HKRW01] proved that the single server Dial-a-Ride problem for a server with capacity 1 is NP-hard already on a caterpillar graph, which is a path with an additional node attached to each node on the path. They also argue that a caterpillar graph can conveniently be used to model elevator operation, since the elevator moves along a line (i. e., the backbone path of the caterpillar) and the "legs" may be used to model acceleration and deceleration as well as floor stopping times.

Given the hardness of the problem and the tight real-time requirements of elevator control it may seem daring to aim for exact reoptimization algorithms. Still, there have been promising results indicating that (almost) optimal solutions can be obtained in very short time for similar problems [KRT02, FR06]. We will start the chapter with a review of related work on exact elevator control algorithms. We will then introduce our new algorithm, that follows the same set partitioning-based approach used in [KRT02, FR06]. The main emphasis is on how to solve the pricing problem and in particular, how to compute lower bounds to be used in our Branch & Bound pricing algorithm. Finally, we conclude by providing computational results on the performance of our reoptimization algorithms and its impact on the performance of the elevator system.

In the entire chapter we restrict ourselves to systems without boarding control which are more realistic anyway. However, the ideas presented can be applied to such systems as well. Since assuming boarding control usually significantly increases the number of requests, we believe that our algorithms will not achieve satisfying running times when applied to such systems.

3.1. Previous work

3.1.1. The algorithms of Closs

Closs was the first who proposed a destination call system in his PhD thesis in 1970 [Glo70]. His main motivation was to develop optimal or almost optimal elevator control algorithms. In order to reasonably define optimality, he assumed the destination floor information to be known and proposed an elevator control system with destination floor input.

As a first step, Closs studies the problem to schedule an elevator to serve a fixed set of assigned destination calls. He observes that an approach based on Dynamic Programming is not feasible, since the state space is rather complex and huge, and the transitions between states are such that few states can be eliminated. He concludes that a Dynamic Programming algorithm yields little advantage over an enumeration algorithm and therefore develops such an exhaustive enumeration algorithm. The basic idea for this algorithm is to look for an optimal dispatch by traversing a tree whose nodes correspond to elevator states and a successor state is generated by moving the elevator to the next (feasible) floor.

It turned out that the computation time of this enumeration algorithm was prohibitively large on the computers available at that time. For instance, it took almost ten minutes [*sic*] to solve an instance with four calls. Therefore, it could not be used as a basis for real elevator control algorithms. Instead, the optimal dispatches computed by the exhaustive enumeration algorithm for some small instances were compared to the corresponding dispatches according to the Collective strategy (see Section 2.2). Since in most cases the dispatches were identical and in the remaining cases Collective was at most 5 % worse than the optimal solution, Closs concluded that Collective is almost optimal. However, he only considered offline instances with at most five destination calls that were generated randomly. It is therefore questionable whether Collective is really optimal for more realistic traffic situations with higher load. We also saw in Chapter 1 that relatively small differences in snapshot solution quality can give much bigger differences in online solution quality.

In order to control a group of two elevators, Closs gives an algorithm that systematically evaluates all assignments of up to eight calls to the two elevator. Based on his observation that Collective gives almost optimal dispatches for a single elevator, each elevator is controlled according to this dispatch. The resulting algorithm is thus an exact algorithm where the feasible schedules are exactly the schedules consisting of a single roundtrip. It is interesting to note that Closs considers a delayed assignment system to be controlled by this algorithm. Again, the computation time is too long for a practical algorithm. Therefore, Closs introduces a more heuristic algorithm that he calls *Single Car Allocation* (SCA). SCA assigns the destination calls successively in the order of arrival by tentatively assigning each call to each elevator to compute the cost of this assignment. Finally, each call is assigned to the elevator with minimum cost increase, i. e., in a best-insertion way, before the next call is considered. Due to this way of inserting calls, a call is always assigned to the same elevator, implying that only new calls need to be assigned, which is important to obtain short running times. Moreover, the SCA algorithm essentially realizes an immediate assignment system, which Closs describes in more detail. Closs compares SCA to the algorithm that determines the optimal assignment using simulation. He reports that the average travel time is at most 10 % and the average waiting time at most 14 % worse than that achieved by the optimal assignment algorithm. Closs concludes that SCA is an almost optimal algorithm for two elevators, that can easily be generalized to larger groups of elevators. In further simulations, he establishes that SCA significantly outperforms a conventional system for up peak traffic. For instance, SCA achieves an average travel time comparable to that of the conventional system with six instead of eight elevators.

However, destination call systems were not established until around 1990. The first implementation seems to be Schindler's MICONIC 10 elevator control [Sch90] for an immediate assignment system. The control algorithm is a variant of the SCA algorithm developed by Closs.

3.1.2. The algorithm of Seckinger and Koehler

Seckinger and Koehler [Sec99, SK99] investigated the dispatching problem for an immediate assignment system. Their work was motivated by Schindler's MICONIC 10 system, which employs a heuristic algorithm similar to the SCA algorithm described above. They develop exact algorithms for scheduling a single elevator to serve a fixed set of calls based on Constraint Programming techniques. These algorithms are then used to assign a new call to an elevator.

The elevator dispatch is modelled as a sequence of floors, such that for each destination call to be picked up there is a stop at the start floor and a later stop at the destination floor. This simple model may be extended by additional requirements, e. g., capacity restrictions and that passengers do no travel in the opposite direction. Objectives considered are the number of stops, the sum of travel times, and the sum of squared travel times. They show that deciding whether or not there is a feasible sequence with at most k stops is NP-hard.

In their first attempt to solve this problem with one of the travel time objectives, they create a Constraint Programming model to be solved by a general solver. However, using the Oz solver they find that running times are far too long to be useful. In order to obtain better running times, they develop an alternative forward search algorithm. This algorithm searches a tree of partial floor sequences, where at each level of the tree the next stopping floor is decided. This floor maybe a destination floor of one of the currently loaded calls or the start floor of one of the calls still waiting. Tree nodes corresponding to infeasible floor sequences are rejected as they are encountered, which allows to take several extensions into account. In addition, they use a bounding scheme for the travel times of each call to prune the search tree, i. e., the forward search algorithm is in fact a Branch & Bound algorithm. The travel time of a currently loaded call is bounded by the time to reach the destination floor directly from the current floor. Similarly, the travel time of a waiting call is bounded by the time needed to travel directly to the start floor and then immediately to the destination floor. Note that these lower bounds do not take into account the correct travel directions of the passengers nor the capacity of the elevator. In fact, both features are considered as optional extensions by the authors.

Both the Constraint Programming model and the forward search algorithm are evaluated by solving several instances. As expected, the forward search algorithm significantly outperforms the Constraint Programming model. When taking capacity and correct travel directions into account, the authors report that the algorithm terminates within less than a second for buildings with up to 16 floors. Even for buildings with up to 25 floors the algorithm determines solutions of a good quality within one second computation time. The performance can be improved further by using a suitable node selection strategy when searching the tree.

Finally, the authors propose a group control algorithm based on the forward search algorithm. A new call is tentatively assigned to each of the elevators and the forward search algorithm is used to compute a corresponding dispatch. The call is then assigned to the elevator which minimizes a certain criterion. Simulation of a five elevator group for a 15 floor building shows that the algorithm provides good performance for moderate traffic intensity, even if the computation time per elevator is 0.2 seconds.

3.1.3. The algorithm of Tanaka et al.

Tanaka et al. [TUA05a, TUA05b] introduce a Branch & Bound algorithm for optimally dispatching a single elevator. They observe that existing control algorithms for destination call systems use the destination information only for deciding about the assignment of the call to an elevator, whereas each elevator is scheduled according to the Collective strategy, i. e., serves all requests on a single roundtrip. The goal of their study is to assess the improvement possible by incorporating the destination information in the scheduling of each elevator and to schedule the elevator optimally.

In their model, they assume that the elevator control has full boarding control. The only restriction is that the group of boarding passengers with the same destination floor is as large as possible, i. e., only limited by the elevator capacity. For instance, if the elevator has remaining capacity three and there are two passengers for floor 2, two for floor 4, and one for floor 10 it is not allowed to board only one from each group. Moreover, they also study a variant that does not forbid calls travelling in the opposite direction. The algorithm computes a schedule that is optimal w. r. t. to the weighted sum of *completion times*. The completion time of a call is the *additional* time needed to transport it to its destination floor. If the weight of a call is just the cost coefficient of the travel time, this objective is equivalent to the weighted sum of travel times. However, Tanaka et al. also consider weights depending on the time the call is already in the system to avoid long deferals of single calls.

We only give a brief review of their computational results here. The algorithm itself is described in Section 3.2.4 in detail. Tanaka et al. consider a 10-floor building and assume that passengers are selected by displaying the destination floors as in a delayed assignment system, but only for a single elevator. Since full boarding control is assumed, it is not necessary for passengers to reenter their calls if capacity is not sufficient because the system can keep track of them. They simulate the algorithm on both up peak and down peak traffic, which is mixed with a small fraction of interfloor traffic. The simulation results indicate that

- The destination call system is stable for higher traffic intensities than the conventional system controlled by the Collective strategy. However, for low load the Collective strategy achieves lower travel times, in particular lower maximum travel times.
- The performance with travelling in the opposite direction of loaded calls allowed is significantly better than if it is forbidden.
- The computation time needed is usually below 0.1 seconds for up to 30 calls, but sometimes more than 10 seconds.

The authors conclude that this algorithm is only suitable for simulation, but not for controlling real systems.

3.1.4. Comparison of algorithms for groups of unit capacity cargo elevators

Friese and Rambau [FR06] developed an exact algorithm for controlling a group of cargo elevators in a distribution center. The algorithm is based on a set partitioning model and is, in fact, a variant of the ZIBDIP algorithm described in Chapter 1. Adapting the ZIBDIP algorithm for elevator control is possible since the considered cargo elevators have unit capacity, which is quite different from elevators with higher capacity. The required extension is that requests are now waiting in queues and may only be served in the order given by the queues. They report that the exact algorithm gives very good solutions after only one second of computation time.

In a very comprehensive study, they compare the performance of a broad range of algorithms, ranging from simple rule-based policies like FIFO or NearestNeighbor over reoptimizing heuristics similar to BestInsert and 2-Opt from Chapter 1 to their exact reoptimization algorithm. They find that the optimizing algorithms are much superior to the simple policies and that the exact algorithm is still able to improve the average waiting time by 10 % over that of BestInsert. Friese and Rambau also study the impact of the control algorithm to system capacity by changing the number of elevators. It turns out that FIFO requires 8 elevators to achieve an average waiting time of at most 40 seconds, while NearestNeighbor needs only 6 and the exact reoptimization algorithm can do with 5 elevators. Thus exact reoptimization algorithms offer the possibility to efficiently utilize the capacity offered by a system.

Although there has been some work on exact elevator control algorithms for a single elevator, there are no exact algorithms for controlling a group of passenger elevators yet. In order to assess the performance of existing algorithms it is desirable to have such exact algorithms as a yardstick for performance evaluation.

3.2. New exact reoptimization algorithms for elevator group control

In this section we propose new exact reoptimization algorithms for the snapshot problem introduced in Section 2.4. They are based on the request model and can therefore be used to control the different kinds of systems discussed in Section 2.1. Thus they are suitable for comparing the performance of these systems.

The existing exact algorithms for elevator control determine optimal schedules for a single elevator only. To extend them to exact group control algorithms, two approaches are possible.

1. Extend the Branch & Bound scheme by a “top-level” part which determines the assignment of requests to elevators. Once assignments are fixed, the existing single elevator algorithms may be used to obtain optimal schedules for each elevator.

2. Determine an optimal dispatch using a set partitioning approach as for the ADAC problem in Chapter 1. Existing algorithms may then be extended to pricing algorithms for determining improving columns.

We decided to follow the second approach since it worked so well for the ADAC problem and the scheduling of cargo elevators [FR06]. An advantage of this approach is that dual information guides the search for better schedules, so usually only few schedules need to be considered. We also expect the set partitioning approach to work well for the case of many unassigned requests which is important for delayed assignment systems. However, it is not easily possible to handle conventional systems via set partitioning, since there are interdependencies between the schedules of the elevators: A request is actually served by the first elevators reaching the floor, so request assignment cannot be decided for each elevator independently.

3.2.1. A set partitioning model

Recall that a dispatch distributes the unassigned requests $\mathcal{R}_u$ among the elevators and gives, for each elevator, a feasible schedule that serves its assigned requests $\mathcal{R}(e)$ and the unassigned requests it received.

Let $\mathcal{S}(e)$ be the set of feasible schedules for elevator e and define $\mathcal{S} := \bigcup_{e \in \mathcal{E}} \mathcal{S}(e)$ (we assume $\mathcal{S}(e_1)$ and $\mathcal{S}(e_2)$ are disjoint whenever $e_1 \neq e_2$). A dispatch is then a collection of schedules, one for each elevator, such that each unassigned request is served by exactly one schedule. For each schedule $S \in \mathcal{S}$ we introduce a decision variable $x_S \in \{0,1\}$ for including it in the dispatch or not. We denote the cost of schedule S by $c(S)$. To compute an optimal dispatch, we can solve the following set partitioning problem:

$$\min \sum_{S \in \mathcal{S}} c(S) x_S \tag{3.1}$$

$$\sum_{S \in \mathcal{S}\colon \varrho \in S} x_S = 1 \qquad \varrho \in \mathcal{R}_u, \tag{3.2}$$

$$\sum_{S \in \mathcal{S}(e)} x_S = 1 \qquad e \in \mathcal{E}, \tag{3.3}$$

$$x_S \in \{0,1\} \quad S \in \mathcal{S}. \tag{3.4}$$

This Integer Programming (IP) problem cannot be solved by an off-the-shelf IP solver directly since the number of feasible schedules and thus columns is very large. However, it is possible to solve the Linear Programming (LP) relaxation using dynamic column generation (see, e. g., [DDS05]), which can be integrated in a general IP solver framework.

Let π_ϱ, $\varrho \in \mathcal{R}_u$, and π_e, $e \in \mathcal{E}$, denote the dual prices associated with constraints (3.2) and (3.3), respectively. The pricing problem for the LP relaxation of the above IP is then to find, for each elevator e, a set of feasible schedules for elevator e having negative reduced cost

$$\tilde{c}(S) := c(S) - \sum_{\varrho \in \mathcal{R}_u \cap S} \pi_\varrho - \pi_e \tag{3.5}$$

or to decide that no such schedule exists. Usually the pricing problem is solved by considering the optimization version asking for a minimum reduced cost schedule.

During the solution of the LP relaxation of (3.1)–(3.4) using column generation we are solving a sequence of subproblems which encompass only subsets of the columns (i. e., schedules). The optimal LP solution of a subproblem is guaranteed to have the same objective value as the LP relaxation of the full IP if there are no more columns with negative reduced cost. However, it may be desirable to stop the column generation process earlier if a specified solution quality has been achieved. To do this, we use the

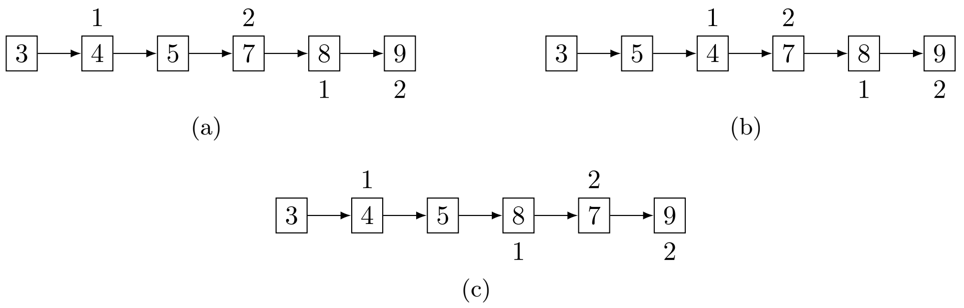


Figure 3.1.: *Three feasible schedules corresponding to the request sequence* $1, 2$*, showing that this sequence does not uniquely determine the schedule. We assume that the elevator is currently located at floor 3 and has to stop at floor 5 before changing direction and that there are two requests* $1\colon 4 \to 8$ *and* $2\colon 7 \to 9$.

following lower bound on the optimal solution value of the LP relaxation of (3.1)–(3.4) due to Lasdon.

Theorem 3.2.1 ([Las70]) *Let* $\tilde{\mathcal{S}}$ *be the set of schedules defining the current subproblem,* $(x_S^*)_{S\in\tilde{\mathcal{S}}}$ *be an optimal solution of the current subproblem and* $(\pi_\varrho^*)_{\varrho\in\mathcal{R}_u}$ *and* $(\pi_e^*)_{e\in\mathcal{E}}$ *be the corresponding dual solution. Moreover, let* $\tilde{\underline{c}}(e)$ *be a lower bound on the minimum reduced cost of any feasible schedule for elevator e, i. e.,*

$$\tilde{\underline{c}}(e) \le \min_{S\in\mathcal{S}(e)} \left\{ c(S) - \pi_e^* - \sum_{\varrho\in\mathcal{R}_u\cap S} \pi_\varrho^* \right\}. \tag{3.6}$$

Then the cost c^* *of an optimal solution of the LP relaxation of* (3.1)–(3.4) *satisfies*

$$c^* \ge \sum_{S\in\tilde{\mathcal{S}}} c(S) x_S^* + \sum_{e\in\mathcal{E}} \tilde{\underline{c}}(e). \tag{3.7}$$

Note that suitable bounds $\tilde{\underline{c}}(e)$ can be easily computed during pricing.

3.2.2. Pricing via Branch & Bound

In order to solve the pricing problem, we need a way to systematically consider all feasible schedules. For the Travelling Salesman problem a sequence of cities uniquely determines a tour for the salesman. However, for the elevator scheduling problem it is not sufficient to give a sequence of requests to determine a schedule. To see this, consider the following example. Assume that the elevator is currently located at floor 3, with a drop commitment at floor 5. Moreover, assume that there are two reqests, with request 1 going from floor 4 to floor 8, and request 2 from floor 7 to floor 9. Figure 3.1 shows that in this situation there are three different schedules corresponding to the request sequence $1, 2$.

The reason why a request pickup sequence does not determine a schedule is that it does not specify whether or not pending drop commitments are to be executed before or after a request is picked up. It turns out that including this drop information is sufficient to characterize a schedule. More precisely, each schedule can be described by giving a sequence of pickup actions and drop actions, specifying which request to pick up and at which floor to stop for dropping, respectively. Of course, not all of these sequences represent feasible schedules.

We can use this description to make up a search tree that can be traversed in order to find schedules with negative reduced costs. Each node in the tree corresponds to a stop of the elevator at a floor, where a subset of the possible requests has already been picked up. The nodes are joined by edges corresponding to pickup or drop actions. An action is feasible at a node if it is compatible with the drop commitments of the node, i. e., executing the action will not transport a passenger in the wrong direction. A full search tree for the small example above is shown in Figure 3.2.

Formally, a node v of the search tree is labelled with the following data:

A_v Set of not yet picked up (assigned) requests from $\mathcal{R}(e)$.

O_v Set of not yet picked up (optional) requests from $\mathcal{R}_u$.

S_v A feasible schedule that serves all requests in $\mathcal{R}(e) \setminus A_v$. This schedule is determined by the path from the root node to node v. In addition to stops corresponding to the actions on this path S_v has stops at the pending drop floors.

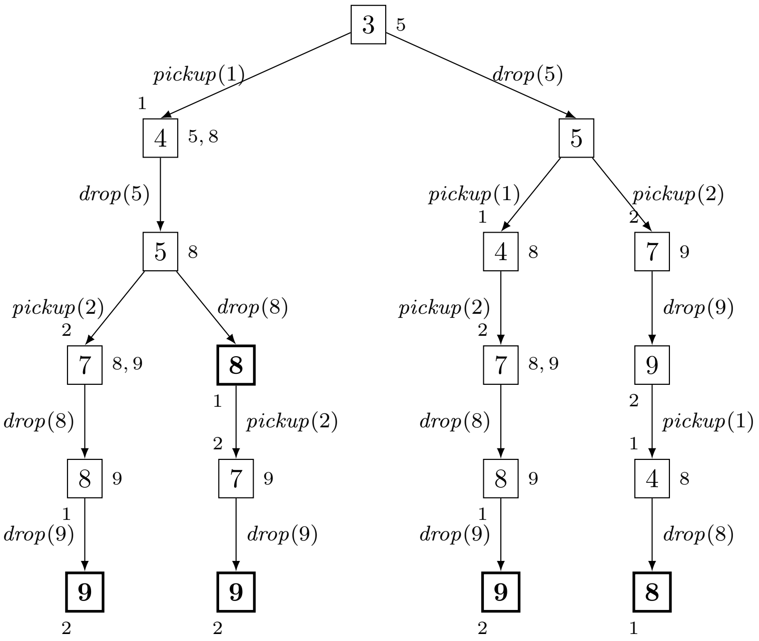
s_v A stop from S_v, namely the stop resulting from the sequence of actions along the path from the root node.

Node v represents all schedules that may be obtained by picking up unserved requests at stop s_v or later. The schedule S_v has the property that there are no pickups after stop s_v, i. e., these stops are there only for dropping loaded destination calls. Node v is called *feasible* if A_v is empty and s_v is the last stop of S_v, i. e., has no pending drop floors. A feasible node corresponds to a feasible schedule that serves all requests assigned to the elevator.

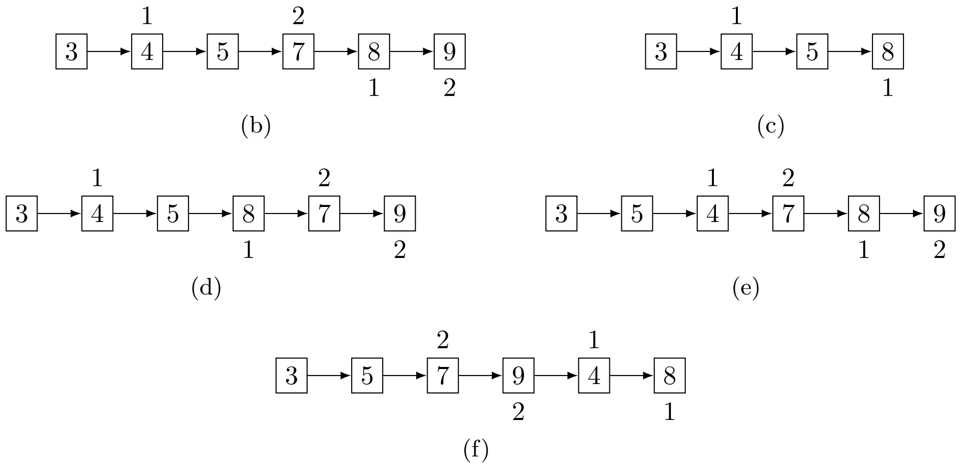
There is a separate search tree for every floor where the elevator can still stop at next, provided that a request is waiting there and the floor is not beyond the next pending drop floor. More precisely, let F be the subset of $\mathcal{F}_i(e)$ where at least one request from $\mathcal{R}(e) \cup \mathcal{R}_u$ is waiting and which are not beyond floor $f_0(e)$, plus floor $f_0(e)$. For each $f \in F$, there is a root node v initialized by

$$\begin{aligned} A_v &:= \mathcal{R}(e), \\ O_v &:= \mathcal{R}_u, \\ S_v &:= \text{schedule for dropping all loaded calls of} \\ &\ \text{elevator } e \text{ with first stop at floor } f, \\ s_v &:= \text{first stop of } S_v. \end{aligned}$$

As described before, a child node v' of node v arises by two actions: Either a request is picked up or the elevator moves to the next floor for dropping a loaded destination call.



(a) Search tree.


(b)

(c)

(d)

(e)

(f)

Figure 3.2.: *Full search tree for the example situation of Figure 3.1 and the feasible schedules in the tree. Figure (a) shows the search tree of all schedules, arising by enumerating all feasible pickup- and drop-action sequences. Note that each pickup-action refers to a request, whereas each drop-action refers to a floor. Each node denotes a stop, the numbers at the top and bottom giving the requests being picked up and the calls being dropped, respectively (for simplicity, we assume a one-to-one correspondence between requests and calls). Additionally, the pending drop floors are shown at the right of each node. Assuming that request* $1\colon 4 \rightarrow 8$ *has been assigned to the elevator and request* $2\colon 7 \rightarrow 9$ *is still unassigned, the nodes corresponding to feasible schedules are marked in bold. A stop corresponds to a feasible schedule if all assigned requests have been served and there are no further drop floors. The feasible schedules are shown in Figures (b)–(f) in the order they are encountered in the search tree.*

Assuming that s_v is not the last stop of S_v, the data of v' according to the drop action is

$$A_{v'} := A_v, \quad O_{v'} := O_v, \quad S_{v'} := S_v, \quad s_{v'} := \text{successor stop of } s_v.$$

For the pickup action, there are two cases. If the start floor of request $\varrho \in A_v \cup O_v$ is the same as $s_v.\mathit{floor}$, then

$$\begin{aligned} A_{v'} &:= A_v \setminus \{\varrho\}, \\ O_{v'} &:= O_v \setminus \{\varrho\}, \\ S_{v'} &:= \mathsf{AddRequest}(S_v, s_v, s(s_v, \varrho), \varrho), \\ s_{v'} &:= s_v, \end{aligned}$$

where we use the AddRequest-operation defined in Section 2.5.4 to compute the schedule resulting from picking up request ϱ. Here, $s(s_v, \varrho)$ denotes the first stop after s_v that is at or beyond the furthest destination floor of request ϱ. In case request ϱ starts at a different floor, the child node labels are

$$\begin{aligned} A_{v'} &:= A_v \setminus s_{v'}.\mathit{pickups}, \\ O_{v'} &:= O_v \setminus \{\varrho\}, \\ S_{v'} &:= \mathsf{AddRequest}(S_v, s'_v, s(s_v, \varrho), \varrho), \\ s_{v'} &:= \text{successor of } s_v \text{ in } S_{v'}, \end{aligned}$$

where s'_v is the successor stop of s_v in S_v. We remove $s_{v'}.\mathit{pickups}$ instead of $\{\varrho\}$ only from A_v since other requests might have been picked up as well since the new stop may be the first stop with this leaving direction. In the case of an immediate assignment system without boarding control $O_{v'}$ has to be restricted further: All optional requests leaving from $s_v.\mathit{floor}$ in direction $s_v.\mathit{direction}$ need to be removed as they would be picked up on stop s_v even if selected later.

Note that, in general, not all requests $\varrho \in A_v \cup O_v$ are valid for the pickup action. This is only true if $s_v.\mathit{drop\_floors}$ is empty. If $s_v.\mathit{drop\_floors}$ is not empty, only requests in direction $s_v.\mathit{direction}$ with start floor between $s_v.\mathit{floor}$ and (including) the first floor from $s_v.\mathit{drop\_floors}$ may be picked up next.

Branching is now done by creating all valid child nodes for a node v. So far it may happen that the same schedule is represented multiple times in the search tree due to symmetries. For instance, assume that requests ϱ_1 and ϱ_2 have the same start floor and direction. Picking them up in the order ϱ_1, ϱ_2 or ϱ_2, ϱ_1 results in the same schedule. To avoid this, we use an order $\prec_{\mathcal{R}}$ on the requests and require that requests can only be picked up in this order at the same stop.

Algorithm 3.1 gives an overview on the overall procedure used to search for schedules with negative reduced costs. It returns a set M of schedules with negative reduced costs; if M is empty then there is no schedule with negative reduced cost. In addition, the algorithm computes a lower bound for the value of the LP relaxation according to Theorem 3.2.1 to be used to measure the progress of the optimization. The procedure uses a queue Q to organize the search through the decision tree. Depending on the kind

Input: snapshot problem data, elevator e, dual prices π_e and $(\pi_\varrho)_{\varrho \in \mathcal{R}(e) \cup \mathcal{R}_u}$, parameters $k \in \mathbb{N}$ and $\theta < 0$

Output: set M of columns with negative reduced cost, lower bound $\tilde{\underline{c}}(e)$ on the reduced cost for any schedule for elevator e

```
procedure EnumSchedules
    $M \leftarrow \emptyset$,
    $Q \leftarrow \emptyset$, $M \leftarrow \emptyset$
    for all feasible first stop floors $f$ do
        Create root node $v$ at floor $f$.
        if LowerBound($v$) $< \theta$ then
            Add $v$ to $Q$.
            if $v$ is feasible then
                Add $S_v$ to $M$.
                $\theta \leftarrow$ minimum reduced cost of the feasible schedules in $M$
            end if
        end if
    end for
    while $Q \neq \emptyset$ and $|M| < k$ do
        Pick first node $v$ from $Q$.
        Let $N$ be the child nodes of $v$ obtained via branching.
        for all $u \in N$ do
            if LowerBound($u$) $< \theta$ then
                Add $u$ to $Q$.
                if $u$ is feasible then
                    Add $S_u$ to $M$.
                    $\theta \leftarrow$ minimum reduced cost of the feasible schedules in $M$
                end if
            end if
        end for
    end while
    Let $\tilde{\underline{c}}(e)$ be the minimum of the cost of the cheapest schedule in $M$ and the
    minimum reduced cost of any node in $Q$.
    return $M$, $\tilde{\underline{c}}(e)$
end procedure
```

Algorithm 3.1: *Outline of Branch & Bound-search for schedules with negative reduced cost.*

of queue used, different search strategies can be implemented. For each new node v a lower bound $\mathsf{LowerBound}(v)$ on its reduced cost is computed and the node is pruned if is not negative enough. We stop the search once we found k feasible schedules with negative reduced costs, since it may not be worthwhile to add all improving schedules if there are many. The parameter θ serves a similar purpose: Initially, it is set to a small negative value, say -1.0×10^6. Everytime a new feasible schedule with reduced cost below θ is encountered, θ is set to the minimum reduced cost of a feasible schedule found so far. This is done to avoid finding many schedules whose reduced cost are only slightly negative.

The lower bound computed by $\mathsf{LowerBound}(v)$ consists of two parts: a lower bound on the reduced cost of the requests already picked up and dropped or currently loaded and a lower bound on the *additional* reduced cost for serving still unserved requests. The reduced cost for the picked up requests are at least $\tilde{c}(S_v)$. To see this, note that the service-related costs (i. e., waiting time, ride time, and travel time costs) increase with time. Due to the construction of S_v, the waiting times of all calls are already fixed and the ride time (and thus the travel time) of currently carried calls can only increase by picking up further requests, since the calls cannot be dropped earlier than at the already existing drop stops. Since capacity penalty cost are only incurred when picking up a request and all pickups of S_v are irrevocable, they cannot decrease either. Finally, the main floor cannot be visited earlier than in S_v after stop s_v, since S_v contains only unavoidable drop stops. Thus the RTT penalty cost of S_v are a lower bound for the RTT penalty cost of any extension of S_v.

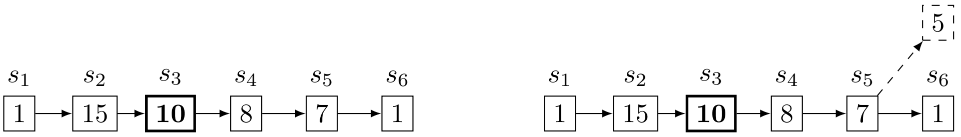
The lower bound for the *additional* reduced costs may be computed using one of the methods discussed in the next subsections. These bounds try to account for the additional service and capacity penalty costs, but not for additional RTT penalty costs.

3.2.3. Greedy lower bounds

A straightforward way to obtain lower bounds for the additional reduced cost of all schedules represented by a node is to determine lower bounds on the pickup and drop times for each request. This is essentially what Seckinger and Koehler [SK99] did, but we also take into account the current drop commitments and the requirement that passengers must not travel in the opposite direction.

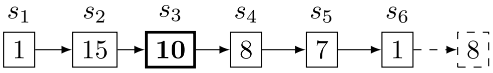
The idea of this “greedy” bound is to determine for each unserved request earliest pickup and drop times. Serving the request incurs at least the service cost according to these times, so the sum of all of these costs gives a lower bound for the additional cost. Note that this corresponds to assuming that *all* requests can be served at their earliest pickup times, i. e., the interactions between requests due to serving a set of them are neglected. Figure 3.3 illustrates this idea.

Consider a node v in the decision tree and a request $\varrho \in A_v \cup O_v$ and let $f^+(\varrho)$ be the start floor of ϱ. In case the direction of ϱ is opposite to the leaving direction from s_v, the elevator has to visit all drop floors before it can pickup ϱ. If the direction matches, the elevator has to pass the drop floors before (and including) $f^+(\varrho)$, too. For both cases, let s' be the first stop of S_v where all preceding drop floors have been passed. A lower



(a) Schedule S_v with $s_v = s_3$. Clearly we have $\underline{t}^+(8 \to 2) \geq s_4.\mathit{arrival}$.

(b) We have $\underline{t}^+(5 \to 3) \geq s_5.\mathit{arrival}$, since the elevator needs to pass floors 8 and 7 before it can stop at floor 5.



(c) We have $\underline{t}^+(8 \to 9) \geq s_6.\mathit{arrival}$, since the elevator needs to serve all drop floors before changing direction.

Figure 3.3.: *Example situations for estimating the pickup time. Figure (a) shows a schedule S_v corresponding to a node v in the search tree, the current stop s_v being marked bold. Figures (b) and (c) show how the pickup time $\underline{t}^+(\varrho)$ of request ϱ can be bounded.*

bound $\underline{t}^+(\varrho)$ for the pickup time of ϱ is then given by

$$\underline{t}^+(\varrho) = \begin{cases} s'.\mathit{arrival} & f^+(\varrho) = s'.\mathit{floor}, \\ s'.\mathit{arrival} + \tau_{\text{stop}} + \tau_{\text{drv}}(e, s'.\mathit{floor}, f^+(\varrho)) & f^+(\varrho) \neq s'.\mathit{floor}. \end{cases}$$

Now consider a call $c \in \varrho$ and let $f_1, \ldots, f_l$ the sequence of floors to be visited before dropping c, where $f_1 = s'.\mathit{floor}$ and f_l is the destination floor of call c. This sequence includes both drop commitments after stop s' and the destination floors of other calls of ϱ. A lower bound $\underline{t}^-(c)$ for the drop time of c is

$$\underline{t}^-(c) = s'.\mathit{arrival} + \max\{\tau_{\text{stop}}, |\varrho|\tau_{\text{load}}\} + \sum_{i=1}^{l-1} \tau_{\text{drv}}(e, f_i, f_{i+1}) + (l-2)\tau_{\text{stop}}.$$

Note that the difference $\underline{t}^-(c) - \underline{t}^+(\varrho)$ does *not* provide a lower bound for the ride time of call c, since drop commitments from stop s' might delay arrival at the destination floor of c. It is however always possible to serve a request ϱ without stopping at other than the request's destination floors. We therefore bound the ride time of c by

$$\underline{t}^r(c) = \max\{\tau_{\text{stop}}, |\varrho|\tau_{\text{load}}\} + \sum_{i=1}^{l-1} \tau_{\text{drv}}(e, f_i, f_{i+1}) + (l-2)\tau_{\text{stop}}, \tag{3.8}$$

where this time $f_1, \ldots, f_l$ is the sequence of floors to be visited before dropping c according to request ϱ only; f_1 and f_l are start and destination floors of c, respectively.

To estimate the additional capacity penalty cost we have to treat the immediate assignment system and the delayed assignment differently, due to the differences in signalling. Recall that in the immediate assignment system, a request has to be picked up on the first stop with matching start floor and leaving direction. Denote by $n(f,d)$ the number of calls that are unavoidably picked up when leaving floor f in direction d. The value of $n(f,d)$ is given by the number of all *assigned* requests (i. e., requests in A_v) starting at floor f with travel direction d, plus the number of calls that are loaded when the elevator leaves floor f in direction d when doing the drop stops according to S_v. Based on $n(f,d)$, we can determine two types of contributions to the additional capacity penalty cost: costs $\underline{c}_{\text{cap}}(\varrho)$ that can be attributed to picking up a request ϱ, and costs $\underline{c}_{\text{cap}}(S_v, A_v)$ that arise from A_v and the structure of S_v so far. Intuitively, the latter part is either due to assigned requests at a floor / direction pair exceeding the capacity, or due to earlier picking up requests that now take up too much capacity to accomodate the remaining assigned requests. Let $\nu(S_v, s_v)$ be the total number of calls exceeding the elevator capacity on stops later than s_v; these are already accounted for in $c(S_v)$. We then have

$$\underline{c}_{\text{cap}}(S_v, A_v) = c_{\text{capacity}} \cdot \left(\sum_{f,d} \max\{0, n(f,d) - \kappa(e)\} - \nu(S_v, s_v) \right),$$

where the sum just counts all calls to be picked up beyond the elevator capacity, which needs to be corrected by those already considered in $c(S_v)$. Additional cost capacity penalty $\underline{c}_{\text{cap}}(\varrho)$ can also arise due to picking up request ϱ, so that the current load (further) exceeds the capacity. Writing $n(\varrho) := n(f,d)$ for the start floor f and direction d of request ϱ, we get the lower bound

$$\underline{c}_{\text{cap}}(\varrho) = \begin{cases} c_{\text{capacity}} \cdot \Big(|\varrho| - \max\{0, \kappa(e) - n(\varrho)\} \Big) & n(\varrho) + |\varrho| > \kappa(e), \\ 0 & \text{otherwise,} \end{cases}$$

for each $\varrho \in A_v \cup O_v$.

For the delayed assignment system, we use $\underline{c}_{\text{cap}}(S_v, A_v) = 0$. The rationale is that exceeding the capacity due to too many loaded calls can be avoided by first emptying the elevator, i. e., signalling no destination floor at the remaining drop floors of S_v. Since every request can effectively be served on its own the only way to incur a capacity penalty is that a single request already exceeds the elevator capacity, i. e., $|\varrho| > \kappa(e)$. This leads to the bound $\underline{c}_{\text{cap}}(\varrho) = c_{\text{capacity}} \cdot \max\{0, |\varrho| - c_{\text{capacity}}(e)\}$ for each $\varrho \in A_v \cup O_v$.

Using these estimates, it is easy to obtain a lower bound $\underline{c}(\varrho)$ on the additional cost for serving request ϱ in any schedule by summing the cost according to the time bounds and the capacity penalty cost, i. e.,

$$\underline{c}(\varrho) = \sum_{c \in \varrho} \big(c_{\text{wait}}(c)\underline{t}^{+}(\varrho) + c_{\text{ride}}(c)\underline{t}^{r}(c) + c_{\text{travel}}(c)\underline{t}^{-}(c) \big) + \underline{c}_{\text{cap}}(\varrho)$$

for linear service costs.

An important observation is that we can do some kind of *dual fixing* based on $\underline{c}(\varrho)$ for requests $\varrho \in O_v$: If $\pi_\varrho \leq \underline{c}(\varrho)$ it will never be favorable to serve this request, since it cannot decrease the reduced cost and it does not have to be served by this elevator. All such requests can thus be removed from O_v, which helps to prune the search tree.

Assuming that there are no requests with $\pi_\varrho \leq \underline{c}(\varrho)$ in O_v, we thus get the following bound $\tilde{\underline{c}}(v)$ for the reduced cost of any schedule in the subtree of v:

$$\tilde{\underline{c}}(v) = c(S_v) + \sum_{\varrho \in A_v} \underline{c}(\varrho) + \sum_{\varrho \in O_v} \left(\underline{c}(\varrho) - \pi_\varrho\right) + \underline{c}_{\mathrm{cap}}(S_v, A_v) - \pi_e.$$

The time complexity of this bound is $\mathcal{O}\left(|A_v \cup O_v| L\right)$, where L is the sum of the number of stops after s_v and the maximum number of destination floors in any request in $A_v \cup O_v$.

3.2.4. Lower bounds via the algorithm of Tanaka et al.

The single elevator Branch & Bound algorithm proposed by Tanaka et al. [TUA05b] is similar to our Branch & Bound pricing algorithm. In particular, its lower bounding scheme can be incorporated into our algorithm. As our setting is more general and differs in some aspects, we describe a modified version of their algorithm that is generalized to our model introduced in Section 2.4 in greater detail. Moreover, we remove the assumption of full boarding control and we will not assume that the passengers are picked up or dropped in a particular order as Tanaka et al. did. However, we stick to linear cost functions and do not take capacity or round trip time penalty cost into account.

In the following, we discuss the algorithm of Tanaka et al. for solving the elevator scheduling problem for a single elevator. At the end of this section we explain how the resulting lower bounds can be applied in each node of our pricing decision tree. Observe that since the algorithm was designed for a single elevator it can only handle the requests already assigned to an elevator.

Basic solution approach

The basic idea is to think of the elevator operation as a sequence of actions: Each currently boarded passenger requires a *drop action* and each waiting passenger a *pickup action* followed by a matching drop action. The task is to schedule these actions subject to several restrictions such that some objective is minimized. In contrast to the original work, the objective function considered here is a linear function of both the waiting and travel times of the calls instead of travel time only. The cost coefficients for the waiting and travel time of call c are denoted by $c_{\mathrm{wait}}(c)$ and $c_{\mathrm{travel}}(c)$, respectively.

As before, let $\mathcal{C}(e)$ denote the set of loaded calls of elevator e and $\mathcal{R}(e)$ the set of requests that need to be served by this elevator. The set J consists of n actions, each described by a weight w_j and a duration d_j, according to $\mathcal{C}(e)$ and $\mathcal{R}(e)$ as follows. For each call $c \in \mathcal{C}(e)$ there is one drop action j with weight $w_j := c_{\mathrm{travel}}(c)$ and duration $d_j := \tau_{\mathrm{load}}$. For each request $\varrho \in \mathcal{R}(e)$ there is one pickup action with weight $w_j := \sum_{c \in \varrho} c_{\mathrm{wait}}(c)$ and duration $d_j := |\varrho| \tau_{\mathrm{load}}$. Moreover, there is one drop action for each destination floor required by ϱ. Let C be the set of destination calls of ϱ with common destination floor f. The weight of the corresponding drop action is $w_j := \sum_{c \in C} c_{\mathrm{travel}}(c)$ and its duration is $d_j := |C| \tau_{\mathrm{load}}$. The floor associated with action j is denoted by f_j, P is the set of pickup actions and the set of drop actions D is the disjoint union of D_B and D_W, corresponding to dropping loaded and waiting calls, respectively. The map *pickup*: $D_W \to P$ gives the pickup action corresponding to a drop action.

The task is now to find a minimum cost schedule of the actions of J, where each drop action is scheduled after its corresponding pickup action and that fulfills the following additional constraints.

1. Drop actions happen as soon as possible after the corresponding pickup action, i. e., on the first stop on the destination floor.
2. There is no change of direction if a drop action in the current direction is still pending. (This constraint was considered optional by Tanaka et al.)
3. The schedule respects the first-stop pickups $P(e, f, d)$.

Formally, we look for an execution order $\pi\colon [n] \to J$ of the actions J, which determines a start time T_j for each action $j \in J$, as in the following model.

$$\min \sum_{j \in J} w_j T_j \tag{3.9}$$

$$\text{s. t.}\quad \pi\colon [n] \to J \text{ is bijective,} \tag{3.10}$$

$$T_{\pi(i)} = \begin{cases} T_{\pi(i-1)} & f_{\pi(i)} = f_{\pi(i-1)}, \\ T_{\pi(i-1)} + \ell(\pi, i-1) + \tau_{\text{drv}}(e, f_{\pi(i-1)}, f_{\pi(i)}) & f_{\pi(i)} \neq f_{\pi(i-1)}, \end{cases} \quad \forall i \in [n], \tag{3.11}$$

$$T_j \geq T_{\mathit{pickup}(j)} + \tau_{\text{drv}}(e, f_{\mathit{pickup}(j)}, f_j) + \tau_{\text{load}} \qquad \forall j \in D_W, \tag{3.12}$$

$$\pi \text{ fulfills the additional constraints above.} \tag{3.13}$$

Here $\ell(\pi, i) := \max\{\tau_{\text{stop}}, P(\pi, i)\}$ denotes the loading time at floor $f_{\pi(i)}$, where $P(\pi, i)$ is the total duration of consecutive actions that take place on $f_{\pi(i)}$ not later than the ith action $\pi(i)$. Moreover, note that we implicitly assume $T_{\pi(0)} = 0$ and $f_{\pi(0)} = f_0$ for notational convenience. Constraints (3.11) prescribe the delay between successive actions, depending on whether they happen on the same floor or not. The precedence constraints of a pickup action and its corresponding drop actions are modelled by (3.12). They could have been formulated using the execution order π, but this formulation is more suitable for the solution approach.

The model given by (3.9)–(3.13) is solved using a Branch & Bound algorithm, that successively fixes $\pi(1)$ up to $\pi(n)$. An initial feasible solution is obtained by inserting the new call in the schedule from the last replanning step.

Computation of lower bounds

The additional constraints (3.13) are not taken into account by the lower bound computation, i. e., they are relaxed. The key idea for computing lower bounds for the remaining problem is to decompose it into three subproblems, dealing with the time needed for loading / unloading, stopping due to visiting different floors, and travelling between floors separately. The decomposition relies on a linear approximation of the floor-to-floor travel times. Suppose we know (large) constants τ_S and τ_F such that

$$\tau_{\text{drv}}(e, f_1, f_2) \geq \tau_S + \tau_F|f_1 - f_2| \quad \forall f_1 \neq f_2. \tag{3.14}$$

We can then relax the timing constraints (3.11) to

$$T_{\pi(i)} \geq \begin{cases} T_{\pi(i-1)} & f_{\pi(i)} = f_{\pi(i-1)}, \\ T_{\pi(i-1)} + \ell(\pi, i-1) + \tau_S + \tau_F|f_{\pi(i-1)} - f_{\pi(i)}| & f_{\pi(i)} \neq f_{\pi(i-1)}, \end{cases} \quad \forall i \in [n]. \tag{3.15}$$

Note that in the case $f_{\pi(i)} \neq f_{\pi(i-1)}$, the delay of action $\pi(i)$ depends on three components due to loading / unloading, stopping, and travelling between floors.

To decompose the problem in subproblems corresponding to these three components, we introduce three separate execution orders $\pi^L, \pi^S, \pi^F \colon [n] \to J$ and corresponding starting times L_j, S_j, F_j. The overall model then reads

$$\min \sum_{j \in J} w_j(L_j + S_j + F_j) \tag{3.16}$$

$$\text{s.t. } \pi^L, \pi^S, \pi^F \colon [n] \to J \text{ is bijective,} \tag{3.17}$$

$$L_{\pi^L(i)} \geq \begin{cases} L_{\pi^L(i-1)} & f_{\pi^L(i)} = f_{\pi^L(i-1)}, \\ L_{\pi^L(i-1)} + \ell(\pi^L, i-1) & f_{\pi^L(i)} \neq f_{\pi^L(i-1)}, \end{cases} \quad \forall i \in [n], \tag{3.18}$$

$$S_{\pi^S(i)} \geq \begin{cases} S_{\pi^S(i-1)} & f_{\pi^S(i)} = f_{\pi^S(i-1)}, \\ S_{\pi^S(i-1)} + \tau_S & f_{\pi^S(i)} \neq f_{\pi^S(i-1)}, \end{cases} \quad \forall i \in [n], \tag{3.19}$$

$$F_{\pi^F(i)} \geq \begin{cases} F_{\pi^F(i-1)} & f_{\pi^F(i)} = f_{\pi^F(i-1)}, \\ F_{\pi^F(i-1)} + \tau_F|f_{\pi^F(i-1)} - f_{\pi^F(i)}| & f_{\pi^F(i)} \neq f_{\pi^F(i-1)}, \end{cases} \quad \forall i \in [n], \tag{3.20}$$

$$L_j \geq L_{\mathit{pickup}(j)} + \tau_{\text{load}} \quad \forall j \in D_W, \tag{3.21}$$

$$S_j \geq S_{\mathit{pickup}(j)} + \tau_S \quad \forall j \in D_W, \tag{3.22}$$

$$F_j \geq F_{\mathit{pickup}(j)} + \tau_F|f_{\mathit{pickup}(j)} - f_j| \quad \forall j \in D_W. \tag{3.23}$$

Obviously, this problem decomposes into the three subproblems. Moreover, its optimal solution value is a lower bound for a solution of (3.9)–(3.13). The subproblem associated with the variables L_j, S_j, and F_j is denoted by (L), (S), and (F), respectively.

The basic approach for solving all three subproblems[*] is to apply Lagrange relaxation to the precedence constraints. Since the optimal value of the Lagrange relaxation for every choice of Lagrange multipliers is a lower bound for the original problem, a lower bound for the Lagrange relaxation is a lower bound for the original problem, too. For fixed Lagrange multipliers, an optimal solution of each relaxed subproblem executes all

---

[*] In the original paper, subproblem (L) is solved exactly, which is not possible for our generalization to the request model. The main reason for this is that in our model, all pickups at the same floor happen at the same time, whereas in the original paper the times differ by a constant loading time. Thus we chose to apply the same solution approach for (L) that Tanaka et al. used for the other two subproblems.

actions on one floor at the same time. We can thus aggregate all actions at one floor to a single action identified by the floor f, where the weight ω_f and the duration d_f of this floor depend on the weights and the durations of the aggregated actions and the Lagrange multipliers. We denote this set of m aggregated actions / floors by $F(J)$. The remaining task is to solve the aggregated problem and to derive values for the Lagrange multipliers such that the solution is large to yield a good lower bound.

Let us consider the Lagrange relaxation of (L) for fixed multipliers $\lambda_j \geq 0$, $j \in D_W$, as an example, which is given by

$$\min \sum_{j \in J} w_j L_j + \sum_{j \in D_W} \lambda_j (L_{pickup(j)} + \tau_{\text{load}} - L_j) \tag{3.24}$$

$$\text{s. t. } \pi^L \colon [n] \to J \text{ is bijective,} \tag{3.25}$$

$$L_{\pi^L(i)} \geq \begin{cases} L_{\pi^L(i-1)} & f_{\pi^L(i)} = f_{\pi^L(i-1)}, \\ L_{\pi^L(i-1)} + \ell(\pi^L, i-1) & f_{\pi^L(i)} \neq f_{\pi^L(i-1)}, \end{cases} \quad \forall i \in [n]. \tag{3.26}$$

Clearly, the objective (3.24) can be rewritten as

$$\min \sum_{j \in J} \bar{w}_j L_j + \tau_{\text{load}} \sum_{j \in D_W} \lambda_j \tag{3.27}$$

with

$$\bar{w}_j := \begin{cases} w_j + \sum_{j' \in D_W \colon pickup(j') = j} \lambda_{j'} & j \in P, \\ w_j - \lambda_j & j \in D_W, \\ w_j & j \in D_B. \end{cases} \tag{3.28}$$

It is obvious that in an optimal solution of (3.24)–(3.26) actions at the same floor happen at the same time. Furthermore it is always optimal to perform actions at the current floor f_0 immediately. Hence we assume that there are no actions at f_0, i. e., $f_0 \notin F(J)$. The weight ω_f of a floor $f \in F(J)$ representing all actions there is given by

$$\omega_f := \sum_{j \in J \colon f_j = f} \bar{w}_j. \tag{3.29}$$

The relaxation and aggregation steps work as well for the other two subproblems. We will now describe how the resulting subproblems are solved and how the Lagrange multipliers are chosen to yield good lower bounds.

Computing lower bounds for (L) The aggregated subproblem for (L) is

$$\min \sum_{f \in F(J)} \omega_f L_f + \tau_{\text{load}} \sum_{j \in D_W} \lambda_j \tag{3.30}$$

$$\text{s. t. } \pi \colon [m] \to F(J) \text{ is bijective,} \tag{3.31}$$

$$L_{\pi(i)} \geq L_{\pi(i-1)} + d_{\pi(i-1)} \qquad \forall i \in [m], \tag{3.32}$$

where the loading time d_f at floor f is given by $d_f := \max\{\tau_{\text{stop}}, \sum_{j \in J\colon f_j = f} d_j\}$. It is easy to see that, for constant Lagrange multipliers λ_j, this problem is essentially a single machine scheduling problem with the weighted sum of completion times as objective which is commonly abbreviated as $1||\sum w_j C_j$. This problem can be optimally solved by the well-known WSPT rule (weighted shortest processing time first) [Smi56]. An optimal solution $\pi\colon [m] \to F(J)$ according to the WSPT rule is obtained by ordering the floors nondecreasingly by the ratio d_f/ω_f. For simplicity, we choose all Lagrange multipliers to be 1.

Computing lower bounds for (S) Applying the same steps to subproblem (S), we obtain the aggregated subproblem

$$\begin{aligned} \min \quad & \sum_{f \in F(J)} \omega_f S_f + \tau_S \sum_{j \in D_W} \lambda_j && (3.33) \\ \text{s.t.} \quad & \pi\colon [m] \to F(J) \text{ is bijective}, && (3.34) \\ & S_{\pi(i)} \geq S_{\pi(i-1)} + \tau_S \qquad \forall i \in [m]. && (3.35) \end{aligned}$$

This subproblem is again a $1||\sum w_j C_j$ scheduling problem with optimal solution value given by

$$\tau_S \sum_{i \in [m]} \omega_{\pi(i)} i + \tau_S \sum_{j \in D_W} \lambda_j,$$

where $\pi\colon [m] \to F(J)$ orders the floors nonincreasingly according to their weight. For fixed Lagrange multipliers λ_j, the second term is constant and dominated by the first. We therefore aim to maximize the first term by choosing appropriate λ_j; this can be achieved by values for ω_f which are both large and balanced.

Tanaka et al. propose a heuristic to do that; a generalization to our request model is shown in Algorithm 3.2. The basic idea is to determine a value κ such that, after updating with $\lambda_j = \kappa$, ω_{f_j} is equal to $\omega_{f_{pickup(j)}}$ (step 7). If however λ_j would be negative or the change to ω_{f_j} would be negative, λ_j is set to a truncated κ (step 8). Finally, values for all λ_j, $j \in D_W$, have been chosen and the values of ω_f, $f \in F(J)$, are set in accordance with (3.27) and (3.28).

Computing lower bounds for (F) The relaxed and aggregated problem of subproblem (F) is

$$\begin{aligned} \min \quad & \sum_{f \in F(J)} \omega_f F_f + \tau_F \sum_{j \in D_W} \lambda_j |f_{pickup(j)} - f_j| && (3.36) \\ \text{s.t.} \quad & \pi\colon [m] \to F(J) \text{ is bijective}, && (3.37) \\ & F_{\pi(i)} \geq F_{\pi(i-1)} + \tau_F |f_{\pi(i-1)} - f_{\pi(i)}| \qquad \forall i \in [m]. && (3.38) \end{aligned}$$

Again, we consider this problem for fixed λ_j. The task is to find a visiting order for the floors in $F(J)$, such that a weighted sum of arrival times is minimized.

```
procedure FindWeights(set of actions $J$ with weights $w_j$)
    Initialize all $\omega_f$ for $f \in F(J)$ to zero.
    for all $j \in P \cup D_B$ do
        $\omega_{f_j} \leftarrow \omega_{f_j} + w_j$
    end for
    for all $j \in D_W$ do
        $\kappa \leftarrow (\omega_{f_j} + w_j - w_{f_{pickup(j)}})/2.$
        $\lambda_j \leftarrow \begin{cases} 0 & \kappa < 0, \\ \kappa & 0 \leq \kappa \leq w_j, \\ w_j & \kappa > w_j. \end{cases}$
        $\omega_{f_j} \leftarrow \omega_{f_j} + w_j - \lambda_j$
        $\omega_{f_{pickup(j)}} \leftarrow \omega_{f_{pickup(j)}} + \lambda_j$
    end for
    return $(\omega_f)_{f \in F(J)}$
end procedure
```

Algorithm 3.2: *Heuristic for determining Lagrange multipliers to obtain balanced weights in lower bound computation for subproblem (S).*

In the special case that all floors in $F(J)$ are in the same direction from f_0, it is optimal to visit the floors in order of increasing distances. Tanaka et al. mention that the general problem can be solved by Dynamic Programming, but is computationally expensive. Instead, they propose the following way to compute lower bounds.

Suppose that are floors in both directions from f_0. In case that $m = |F(J)| = 2$ there are only two possible visiting orders which are both tried. If $m \geq 3$ then there are at least two floors in one direction. Assume w. l. o. g. that this is the up direction and consider the two floors $f_{m-1} < f_m$ that are furthest up. In an optimal solution, f_{m-1} is visited (not necessarily immediately) before f_m, since an interchange would yield a cost increase. Therefore there are two feasible orders for $m = 3$, too. For $m = 4$, we can apply the same argument to $f_1 < f_2$, which are furthest down. In this case there are six orders possible, namely

$$\begin{gathered} f_2, f_1, f_{m-1}, f_m \\ f_2, f_{m-1}, f_1, f_m \\ f_2, f_{m-1}, f_m, f_1 \\ f_{m-1}, f_2, f_1, f_m \\ f_{m-1}, f_2, f_m, f_1 \\ f_{m-1}, f_m, f_2, f_1. \end{gathered}$$

If there are more than four different floors, we use the following decomposition. Let F_1 denote the set of (at most four) extreme floors as above and $F_2 := F(J) \setminus F_1$. The objective can then be written as $\sum_{f \in F(J)} \omega_f F_f = \sum_{f \in F_1} \omega_f F_f + \sum_{f \in F_2} \omega_f F_f$. A lower bound for the first term is then obtained as in the case for $m = 3$ or $m = 4$. The second term is bounded by assuming that each floor in F_2 is served first, i. e., $\sum_{f\in F_2} \omega_f F_f \geq \sum_{f \in F_2} \omega_f |f - f_0|$. The time to compute the overall bound is $\mathcal{O}(m)$.

Unlike the previous subproblem, the constant term in the objective (3.36) is relatively large, so Tanaka et al. choose $\lambda_j = 1$ for all $j \in D_W$.

Putting the lower bounds together A lower bound for the original problem is obtained by summing the lower bounds for subproblems (L), (S), and (F). The time needed to compute this lower bound is $\mathcal{O}(n \log n)$.

This lower bound can be readily incorporated into our Branch & Bound algorithm by considering A_v instead of $\mathcal{R}(e)$ as the requests to be scheduled. Similarly, the set of loaded calls is given by $s_v.\mathit{current\_calls}$ instead of $\mathcal{C}(e)$. As mentioned before, unassigned requests are not handled by Tanaka et al.'s approach. To obtain a meaningful bound, we therefore add the greedy bound for the unassigned requests. Denoting the Tanaka et al. lower bound for the *additional* cost of serving the assigned requests by $\underline{c}_{\text{Tanaka}}(S_v, A_v)$, we thus have the following bound $\tilde{\underline{c}}(v)$ for the reduced cost of any schedule in the subtree of v:

$$\tilde{\underline{c}}(v) = c(S_v) + \underline{c}_{\text{Tanaka}}(S_v, A_v) + \sum_{\varrho \in O_v \colon \underline{c}(\varrho) < \pi_\varrho} \big(\underline{c}(\varrho) - \pi_\varrho\big) - \pi_e.$$

Note that in order to obtain the *additional* cost for serving assigned requests, one has to modify the lower bound obtained by summing the lower bounds for the subproblems. In particular, one has to subtract the travel cost for the currently loaded calls and to add the waiting cost of the requests in A_v up to now.

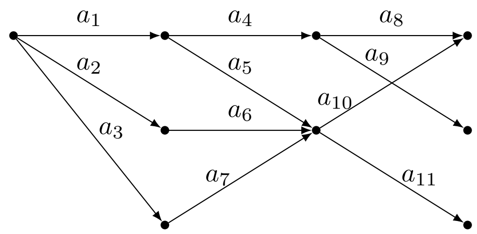
3.2.5. Lower bounds based on state-space relaxations

A common technique to solve routing problems is Dynamic Programming. The basic idea is to formulate the routing problem as a shortest path problem in a suitable graph. In principle, every combinatorial optimization problem may be solved using this way. The resulting *state-space graph* $(\mathcal{S}, \mathcal{A}(\mathcal{S}), c)$ uses the nodes $\mathcal{S}$ to capture certain decision stages called *states* and the arcs $\mathcal{A}(\mathcal{S})$ describing further decisions, i. e., transitions between states. Finally, the arc cost function $c \colon \mathcal{A}(\mathcal{S}) \to \mathbb{R}$ models the cost of the original problem. For many (realistic) problems, however, the resulting graph turns out to be very huge so the approach is not promising. This is also the case for our elevator scheduling pricing problem, due to the complexity of the side constraints and the objective function.

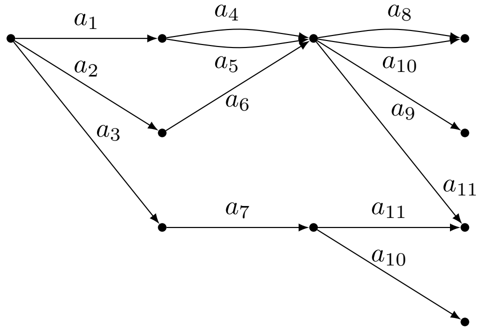
Christofides et al. [CMT81] proposed a method called *state-space relaxation* to allow shortest path algorithms to be used for computing lower bounds. A state-space relaxation of a state-space graph $(\mathcal{S}, \mathcal{A}(\mathcal{S}), c)$ is another (much smaller) state-space graph $(\hat{\mathcal{S}}, \mathcal{A}(\hat{\mathcal{S}}), \hat{c})$ together with a mapping $g \colon \mathcal{S} \to \hat{\mathcal{S}}$ satisfying

$$(s,t) \in \mathcal{A}(\mathcal{S}) \implies (g(s), g(t)) \in \mathcal{A}(\hat{\mathcal{S}}), \text{ and,} \tag{3.39}$$

$$\hat{c}(\hat{s}, \hat{t}) \leq \min\{c(s,t) \mid g(s) = \hat{s}, g(t) = \hat{t}\} \text{ for all } (\hat{s}, \hat{t}) \in \mathcal{A}(\hat{\mathcal{S}}). \tag{3.40}$$



(a) Original state-space graph.



(b) Relaxed state-space graph that is not a homomorphic image of the original state-space graph.

Figure 3.4.: *Difference between the notion of state-space relaxation used in [CMT81] and the notion used here. The relaxed state-space graph is not the homomorphic image of the original state-space graph which can be seen by considering the arcs* a_2*,* a_6*,* a_3*, and* a_7*. Nevertheless, each path in the original state-space graph is also represented in the relaxed state-space graph.*

Requirement (3.39) states that g is a graph homomorphism from the big to the small state-space graph, whereas (3.40) ensures that an arc in the small graph is at most at expensive as the cheapest of its preimages in the original graph. From now on we will refer to the *original state space* and the *relaxed state space*.

Let $C(s,t)$ denote the length of a shortest s,t-path in $(\mathcal{S}, \mathcal{A}(\mathcal{S}), c)$ and $\hat{C}(\hat{s},\hat{t})$ the length of a shortest $\hat{s},\hat{t}$-path in $(\hat{\mathcal{S}}, \mathcal{A}(\hat{\mathcal{S}}, \hat{c})$. Due to the homomorphism property, it is clear that

$$\hat{C}(g(s), g(t)) \leq C(s,t) \quad \text{for all } s, t \in \mathcal{S}, \tag{3.41}$$

so we can compute lower bounds by shortest path algorithms in a much smaller graph.

We use a more general version of the state-space relaxation idea (see Figure 3.4 for an illustration). The key property guaranteed by (3.39) and (3.40) is that for each path P in $(\mathcal{S}, \mathcal{A}(\mathcal{S}), c)$ there is an image path $\hat{P}$ in $(\hat{\mathcal{S}}, \mathcal{A}(\hat{\mathcal{S}}), \hat{c})$ which has at most the cost of P, i. e., $\hat{c}(\hat{P}) \leq c(P)$. This ensures that a shortest path in the relaxed state-space graph provides a lower bound for one in the original state-space graph. In our variant, we construct the state-space relaxation graph with this property inductively.

We assume that the original state-space graph $(\mathcal{S}, \mathcal{A}(\mathcal{S}), c)$ is acyclic, has an initial state s_0, and each of its arcs arises due to some decision or action (think of the pickup and drop actions in the schedule search tree). For a state $s \in \mathcal{S}$, we denote by $\mathcal{A}(s)$ the set of actions allowed in state s, i. e., $\mathcal{A}(s)$ is the set of arcs leaving s. Let state $\hat{s}_0$ be the image of the initial state s_0 in the relaxed state-space graph. Our relaxed state-space satisfies the following properties:

(SSR1) For each state $s \in \mathcal{S}$, there is an injective map $\phi_s \colon \mathcal{A}(s) \to \mathcal{A}(\hat{s})$. Intuitively, this means that for each action possible in the original state space there is a corresponding action in the relaxed state space.

(SSR2) For each path $P = (s_0, a_0, \ldots, s_{n-1}, a_{n-1}, s_n)$ in $(\mathcal{S}, \mathcal{A}(\mathcal{S}), c)$ the path $\hat{P} = (\hat{s}_0, \phi_{s_0}(a_0), \ldots, \hat{s}_{n-1}, \phi_{s_{n-1}}(a_{n-1}), \hat{s}_n)$ is a path in $(\hat{\mathcal{S}}, \mathcal{A}(\hat{\mathcal{S}}), \hat{c})$.

(SSR3) For each state $s \in \mathcal{S}$ and each $a \in \mathcal{A}(s)$ holds $\hat{c}(\phi_s(a)) \leq c(a)$.

The first two statements guarantee the existence of an image path by ensuring that in the relaxed state-space graph the set of allowed actions reflects the actions in the original state-space graph. The third statement guarantees the lower bound property.

To model our pricing problem, we use a state space where each state $s \in \mathcal{S}$ is a tuple $s = (A, O, f, t_p, L)$ with the following meaning of each component. Note that the meaning of A and O is opposite to the data in the Branch & Bound nodes, i. e., they denote the set of *picked up* instead of *not-yet picked up* requests.

A Set of picked up (assigned) requests from $\mathcal{R}(e)$.

O Set of picked up (optional) requests from $\mathcal{R}_u$.

f Current stopping floor.

t_p Time for picking up the next call, i. e., the arrival time at this floor.

L Set of currently loaded destination calls.

We use the notation $A(s)$ and so on to refer to the components of a state. Observe that $f(s)$, $t_p(s)$, and $L(s)$ are just the information needed to compute the service cost (the capacity and RTT penalty costs are ignored here for simplicity) in a destination call system: Using $t_p(s)$ we can determine the waiting time for a request being picked up on the same floor, from $L(s)$ we can determine the drop floors $\mathcal{F}(L(s))$, and using $f(s)$ and $L(s)$ it is possible to calculate the time for reaching any other floor. The components $A(s)$ and $O(s)$ are used to keep track of the requests already picked up.

The transitions between states arise from applying a pickup or drop action just as in the search tree. In fact, the whole state-space model can be viewed as an alternative description of the search space used in the Branch & Bound pricing algorithm. For an unserved request ϱ, we denote the successor state resulting from a pickup action by $\mathit{pickup}(s, \varrho)$. Likewise, we use the notation $\mathit{drop}(s, f)$ for the successor state resulting from a drop action at floor $f \in \mathcal{F}(L(s))$. Moreover, $\Delta_{\mathit{pickup}(s,\varrho)}$ denotes the *additional* cost due to the pickup action, i. e., the service cost for request ϱ, but also the additional service cost for the destination calls in $L(s)$. The additional cost $\Delta_{drop(s,f)}$ associated to a drop action are 0, since the schedule does not change due to this action. The cost for the arcs in the state-space graph are thus

- $\Delta_{pickup(s,\varrho)}$ for a pickup action with request $\varrho \in \mathcal{R}(e) \setminus A(s)$,
- $\Delta_{pickup(s,\varrho)} - \pi_\varrho$ for a pickup action with request $\varrho \in \mathcal{R}_u \setminus O(s)$, and
- $\Delta_{drop(s,f)} = 0$ for every drop action at a floor $f \in \mathcal{F}(L(s))$.

The initial state s_0 is given by $A(s_0)$, $O(s_0)$, $f(s_0)$, $t_p(s_0)$, and $L(s_0)$ corresponding to the node in the search tree for which a lower bound is computed. Observe that the state-space graph $(\mathcal{S}, \mathcal{A}(\mathcal{S}), c)$ defined this way is acyclic.

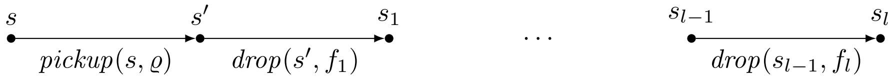
Obviously, the size of this state space is exponential in the number of unserved requests, not to mention the complexity due to including time in the model. Therefore, dynamic programming is not efficient even for small instances. This is why we employ the idea of using a state-space relaxation as described above to compute lower bounds. To arrive at a reasonably small relaxed state space, we use a number of measures to reduce the size:

- We only keep the *number* of assigned and unassigned requests already served. This idea has been used e. g., by Christofides et al. [CMT81] for the Travelling Salesman Problem.
- We discretize the time using some discretization δ.
- We only keep a limited number k of drop floors from $\mathcal{F}(L(s))$.

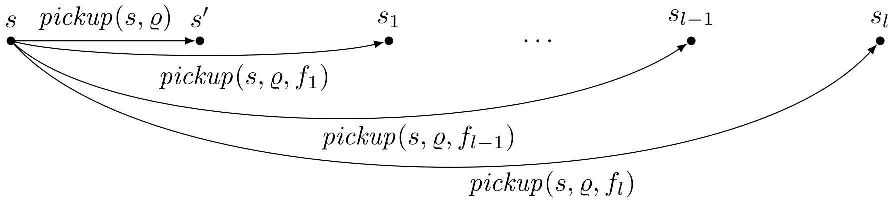
The time discretization will be achieved by rounding down the pickup times of each stop to the nearest multiple of δ. In order to avoid unnecessary many round-downs that degrade the quality of the lower bound, we use a modified but equivalent definition of the actions in the original state space, which will serve as the basis of the state-space relaxation. Consider a pickup of request ϱ in state s, i. e., assume that $f(s) = f^+(\varrho)$, the start floor of ϱ. The drop floors to be visited from the resulting state are the floors $\mathcal{F}(L(s) \cup \varrho) \setminus \{f^+(\varrho)\} = \{f_1, \ldots, f_l\}$. We now replace every path starting with the action *pickup*(s, ϱ) and followed by subsequent drop actions at floors $f_1, \ldots, f_i$ by a single arc corresponding to a *composite action* called *pickup*(s, ϱ, f_i), meaning that ϱ is picked up from state s and then all drop floors up to f_i are visited without any pickup in between (see Figure 3.5). Observe that since this is just the meaning of the original path, this modification is equivalent to the original definition and every sequence of actions is still represented in the state-space graph. This construction will later allow us to round down just once for every composite action, instead of everytime a state is reached. The cost of the composite action *pickup*(s, ϱ, f_i) is that of the original path, which is that of the pickup action.

We are now ready ro describe the relaxed state-space graph. Formally, a state $\hat{s} \in \hat{\mathcal{S}}$ is a tuple $\hat{s} = (n_A, n_O, f, \hat{t}_p, F_k, \varrho)$ with the following meaning.

n_A Size of A.



(a) Path in the original definition of state transitions.


s
$pickup(s, \varrho)$
s'
s_1
$\cdots$
s_{l-1}
s_l
$pickup(s, \varrho, f_1)$
$pickup(s, \varrho, f_{l-1})$
$pickup(s, \varrho, f_l)$

(b) Modified definition of state transitions using composite actions.

Figure 3.5.: *Idea of replacing pickup action / drop actions paths by composite actions combining one pickup and a series of drop actions.*

n_O Size of O.

f Current stopping floor.

$\hat{t}_p$ Time for picking up the next call.

F_k A subset of $\mathcal{F}$ of size at most k, giving the drop floors to be considered.

ϱ The request that was the last to be picked up so far. If there was no pickup so far, the dummy request $\perp$ is used. This component is used to eliminate short cycles in the state-space graph.

As sketched earlier, we inductively construct the relaxed state-space by applying the same actions as in the original state-space, starting from the initial state s_0. The state $\hat{s}$ corresponding to original state s (i. e., reached by the same path of actions) will fulfill the following natural invariant w. r. t. s:

1. $n_A(\hat{s}) = |A(s)|$ and $n_O(\hat{s}) = |O(s)|$,
2. $f(\hat{s}) = f(s)$,
3. $\hat{t}_p(\hat{s}) \leq t_p(s)$,
4. $F_k(\hat{s}) \subseteq \mathcal{F}(L(s))$, and
5. $\varrho(\hat{s})$ is the request of the last pickup action occuring on the path to s.

For a state $\hat{s}$ the set $R(\hat{s})$ of *admissible requests* is defined to be

- any request apart from $\varrho(\hat{s})$ if $F_k(\hat{s})$ is empty, or
- if $F_k(\hat{s})$ is nonempty, any request $\varrho \neq \varrho(\hat{s})$ that 1. starts on floor $f(\hat{s})$, leaves in the direction implied by $F_k(\hat{s})$, and satisfies $\varrho(\hat{s}) \prec_{\mathcal{R}} \varrho$ for some fixed total order $\prec_{\mathcal{R}}$ on the requests, or 2. has start floor between $f(\hat{s})$ and the nearest floor in $F_k(\hat{s})$, including this floor.

A pickup action *pickup*$(\hat{s}, \varrho)$ is only allowed if $\varrho \in R(\hat{s})$. Note that $\varrho(\hat{s})$ is used to eliminate paths corresponding to picking up the same request twice without picking up another request in between. Similarly, the order $\prec_{\mathcal{R}}$ is used to break the symmetry due to different pickup orders on the same floor.

The components describing state $\hat{s}' = \textit{pickup}(\hat{s}, \varrho)$ are determined as follows. Of course, $n_A(\hat{s}') = n_A(\hat{s}) + 1$ or $n_O(\hat{s}') = n_O(\hat{s}) + 1$, depending on whether $\varrho \in \mathcal{R}(e)$ or $\varrho \in \mathcal{R}_u$, $f(\hat{s}')$ is the start floor $f^+(\varrho)$ of ϱ, and $\varrho(\hat{s}') = \varrho$. The lower bound θ_0 for the pickup time of ϱ is either $\hat{t}_p(\hat{s})$ if ϱ starts on floor $f(\hat{s})$, or $\hat{t}_p(\hat{s}) + \tau_{\text{stop}} + \tau_{\text{drv}}(e, f(\hat{s}), f^+(\varrho))$ otherwise. The value of $\hat{t}_p(\hat{s}')$ is defined as θ_0 rounded down to the next multiple of δ. Observe that τ_{stop} is a lower bound on the stopping time on floor $f(\hat{s})$. Since $\hat{t}_p(\hat{s})$ is a lower bound for $t_p(s)$ for state s reached on path P, θ_0 and thus $\hat{t}_p(\hat{s}')$ is a lower bound for $t_p(s')$ for state s' reached on path $(P, \textit{pickup}(s, \varrho))$. Finally, $F_k(\hat{s}')$ is defined to be the k floors of $\big(F_k(\hat{s}) \setminus \{f^+(\varrho)\}\big) \cup F^-(\varrho)$ that are furthest away from $f(\hat{s}')$, where $F^-(\varrho)$ are the destination floors of ϱ.

Moreover, there is a composite action *pickup*$(\hat{s}, \varrho, f)$ for every $f \in F := \big(F_k(\hat{s}) \setminus \{f^+(\varrho)\}\big) \cup F^-(\varrho)$. Note that F may at this point contain more than k floors. For each floor $f \in F$ we can compute a lower bound θ_f for the time of reaching this floor, starting from $\hat{t}_p(\hat{s})$ and adding the appropriate floor stop times and floor-to-floor travel times for all intermediate floors from F. Denoting by $\hat{s}''$ the state resulting from picking up ϱ from stop $\hat{s}$ and perfoming all the drop actions up to and including floor f, we can then set $\hat{t}_p(\hat{s}'')$ to θ_f rounded down to the nearest multiple of δ. This way, we rounded just once for the entire composite action. Observe that this construction also ensures that θ_f and $\hat{t}_p(\hat{s}'')$ are lower bounds for $t_p(s'')$, with notation and arguments analogously to the pickup action above. As expected, $n_A(\hat{s}'')$, $n_O(\hat{s}'')$, $f(\hat{s}'')$, and $\varrho(\hat{s}'')$ are the same as for the pickup action without a subsequent drop action. The set of drop floors $F_k(\hat{s}'')$ is the set of drop floors from $F_k(\hat{s}')$ remaining after visiting floor f.

In summary, the definition of the arcs between states satisfy the invariant given above. It remains to define the cost corresponding to each arc and to establish that they are indeed suitable for computing lower bounds of the original problem. Since the drop action does not incur additional costs, the arcs corresponding to composite actions get the same cost as that corresponding to the pickup action only. It is thus sufficient to consider the cost for pickup actions. These costs are computed based on the time estimates θ_0 and θ_f obtained as described above (note that these are the values before rounding down). Since these times are lower bounds for the corresponding times in the original state space and the drop floors $F_k(\hat{s})$ are a subset of the floors in the original state space, we can use them to obtain lower bounds for the waiting time and the travel time of each call of ϱ. The ride time can be bounded via Equation (3.8) as in the greedy lower bound. We thus arrive at a lower bound on the cost for serving request ϱ from state s in the original state space, which is the cost of the pickup action arc in the relaxed state space. This lower bound is a lower bound for the *additional* cost incurred by serving request ϱ, which is the cost of the pickup action in the original graph. Note that we neglect the additional service cost for the loaded calls due to serving request ϱ in the state-space relaxation. Finally, we observe that the constructed graph has the properties (SSR1), (SSR2), and (SSR3) and thus is a valid state-space relaxation.

We remark that due to keeping track of the last picked up request in the relaxed state space, the relaxed state space is in general not a homomorphic image of the original state space: If there are two requests starting at the same floor both pickup orders will lead to the same state in the original state space, whereas they lead to different states in the relaxed state space. Moreover, the fact that there is in general no homomorphism is also due to the time discretization employed.

Before turning to the description of the actual algorithm used to compute the state-space relaxation lower bound, observe that the size of the relaxed state space is polynomial in the input size of the snapshot problem for fixed k and δ. To see this, it is sufficient to argue that there are only polynomially many values of $\hat{t}_p$. This follows from the fact that in an optimal schedule, there are no stops without picking up or dropping a call. Thus, the total time to execute any schedule is bounded from above by the time it takes to serve all requests by serving each request on its own. Clearly, the time needed to serve each request is constant w. r. t. input size.

Although the state space size is polynomial, it is still rather huge. Yet there is some hope that the bounds are computationally tractable, since usually there are not many requests (in general, at most 30). Additionally, there are some ways to keep the size of the state space reasonable, at the expense of a weaker bound. The easiest way to do so is to choose small k and large δ. It is not sensible to drop F_k completely, as a large share of the cost –most importantly, waiting cost for requests picked up later– is due to stopping to drop the loaded passengers.

We use a label-correcting algorithm to compute, for each state $\hat{s}$ reachable from the initial state $\hat{s}_0$, the cost $\hat{C}(\hat{s})$ of a shortest path from $\hat{s}_0$ to $\hat{s}$, see Algorithm 3.3. The algorithm maintains the states in sets $S(n_A, n_O)$ according to their first components n_A and n_O and uses a priority queue Q to determine the order in which the sets $S(n_A, n_O)$ are processed. It is crucial that all states in $S(n_A, n_O)$ are processed before any states in $S(n_A+1, n_O)$ and $S(n_A, n_O+1)$, since only then the values of $\hat{C}(\hat{s})$ for $\hat{s} \in S(n_A+1, n_O) \cup S(n_A, n_O+1)$ are guaranteed to be correct. Therefore, the partial order $\prec$ used by Q has to satisfy

$$\begin{aligned} &(n_A, n_O) \prec (n_A+1, n_O), \text{ and} \\ &(n_A, n_O) \prec (n_A, n_O+1). \end{aligned} \tag{3.42}$$

There is another measure we call *time reduction* to reduce the size of the state space that may be used by the algorithm. The idea is to identify all states differing only in the $\hat{t}_p$-component, i. e., replacing all of these states by the one which has the minimal $\hat{t}_p$-value. The corresponding state-space relaxation still fulfills properties (SSR1), (SSR2), and (SSR3) and thus remains a valid state-space relaxation. It is easily possible to incorporate time reduction in Algorithm 3.3.

3.2.6. The overall algorithm ExactReplan

By now we described all the ingredients of our new exact reoptimization algorithm ExactReplan. The algorithm computes a dispatch for a given snapshot problem as follows. If there is at most one unassigned request, we solve the elevator scheduling problem using enumeration. In case there is no unassigned request, we compute an optimal schedule

```
procedure StateRelax
    1. Let $Q$ be a priority queue for pairs $(n_A, n_O)$, using a partial order $\prec$
       satisfying (3.42).
    2. Given an initial state $s_0$, let $\hat{s}_0$ be the state corresponding to $s_0$ and set
       $\hat{C}(\hat{s}_0) := c(s_0)$.
    3. Let $\mathcal{F}(L(s_0)) = \{f_1, \dots, f_l\}$ and $\hat{s}_i$ be the states corresponding to states
       $drop(s_0, f_i)$ for $1 \leq i \leq l$. Set $\hat{C}(\hat{s}_i) := c(s_0)$ for $1 \leq i \leq l$.
    4. Put all pairs $(n_A, n_B)$, $0 \leq n_A < |A_v|$ and $0 \leq n_O < |O_v|$, into $Q$.
    5. While $Q \neq \emptyset$:
        a) Remove $(n_A, n_O)$ from $Q$.
        b) For each $\hat{s} \in S(n_A, n_O)$:
              For each $\varrho \in R(\hat{s})$:
              - Using $f(\hat{s})$, $\hat{t}_p(\hat{s})$, and $F_k(\hat{s})$ compute the time $\theta_0$ for arriving at
                the start floor of $\varrho$ and the times $\theta_{f_i}$ for arriving at the drop
                floors $\{f_1, \dots, f_l\}$ given by $F_k(\hat{s})$ and the destination floors of $\varrho$.
              - Let $\hat{s}' = pickup(\hat{s}, \varrho)$ be the state resulting from picking up $\varrho$
                and $\hat{\Delta}$ the cost incurred by this, based on times $\theta_0$ and $\theta_{f_i}$.
                Update the cost of $\hat{s}'$ via

                    $\hat{C}(\hat{s}') \leftarrow \min\{\hat{C}(\hat{s}'), \hat{C}(\hat{s}) + \hat{\Delta} - \pi_{\varrho}.\}$

              - For each $1 \leq i \leq l$, let $\hat{s}'' = pickup(\hat{s}, \varrho, f_i)$ be the state
                resulting from the composite action. Update the cost of $\hat{s}''$ via

                    $\hat{C}(\hat{s}'') \leftarrow \min\{\hat{C}(\hat{s}''), \hat{C}(\hat{s}) + \hat{\Delta} - \pi_{\varrho}.\}$

    6. Return $\min\left\{\hat{C}(\hat{s}) \,\middle|\, \hat{s} \in \bigcup_{0 \leq n_O \leq |O_v|} S(|A_v|, n_O)\right\}$.
end procedure
```

Algorithm 3.3: *Overview of the label-correcting algorithm to compute lower bounds based on state-space relaxation.*

for each elevator via EnumSchedules (see Algorithm 3.1), which together constitute an optimal dispatch. We exploit here that EnumSchedules finds a schedule that is optimal w. r. t. to the primal cost if all dual prices are set to zero. For a single unassigned request, we additionally compute for each elevator an optimal augmented schedule serving the fixed requests plus this unassigned request. The request is then assigned to the elevator serving it with least additional cost, so the optimal dispatch is given by the augmented optimal schedule for this elevator and the unaugmented optimal schedules for the remaining elevators. Upper bounds for the enumeration are obtained by constructing feasible schedules via BestInsert (Algorithm 2.2).

The general case of at least two unassigned requests is treated by solving the LP relaxation of the set partitioning IP (3.1)–(3.4) using column generation. The IP resulting from column generation is then solved to optimality using a standard IP solver. Note that in general this approach does not guarantee optimal solutions since we solve the LP relaxation to optimality only in the root node of the IP search tree. The set of schedules in the IP model is initialized by the schedules obtained from inserting the unassigned requests in the schedule from the last reoptimization run using BestInsert. Thus the IP is always feasible. The pricing problem is solved by EnumSchedules, invoking it with the dual prices from the current LP relaxation.

In all invocations of EnumSchedules, the lower bound procedure LowerBound computes either the greedy lower bound, the Tanaka lower bound, or the state-space relaxation lower bound as described in the preceding sections. Moreover, it is also possible to use the Tanaka or the state-space relaxation lower bound as a secondary lower bound for the greedy bound, invoking it only if the greedy lower bound did not prune the tree node. More precisely, a secondary lower bound is only computed if the cost of the current node are at least the threshold θ or if there are unserved fixed requests at the node. The rationale behind this is that only then the node can potentially be pruned due to the lower bound. When using secondary lower bounds, the greedy bound is always used as the primary bound since it is the only bound that allows dual fixing as explained on page 65. It turns out that the pruning achieved by this dual fixing is crucial to obtain reasonable running times. Therefore it is applied everytime the greedy lower bound is computed.

There are two other optional features of the algorithm. The first is *early stopping*, meaning that the pricing process is stopped as soon as the Lasdon lower bound according to Theorem 3.2.1 certifies a certain optimality gap for the LP relaxation. The second is the *reuse of old schedules*: Schedules that were generated during the solution process of the last reoptimization run are considered during pricing before invoking EnumSchedules, i. e., we check whether any of these old schedules have negative reduced cost. If so, we include these schedule in the LP, resolve it, and start the next pricing round. Note that this necessitates keeping the old schedules and updating them to the changes happening between successive reoptimization runs. This reuse enables to take advantage of the considerable effort for finding good schedules we invested in the last reoptimization run.

3.3. Computational results

We already mentioned in Section 2.2 that a key figure for the performance of an elevator system is its handling capacity. Conceptually, the handling capacity measures which passenger arrival rate the system can handle. Equation (2.1) from Section 2.2 is a way to provide an estimate for the handling capacity under pure up peak traffic based on probabilistic assumptions. Elevator practitioners use another more practical and also pragmatic definition [Bar02] based on the observation that the up peak traffic is the most demanding traffic. According to this definition, the handling capacity is the percentage of the entire building population that may arrive in a 5 minute interval such that the elevator system provides a satisfactory service. Its value is determined by simulating the arrival of a corresponding number of passengers in 5 minutes and verifying whether the service quality is sufficient. Barney [Bar02, p. 88] states that this provides a reasonable approximation to the usual morning up peak and is thus a justified simplification. A 5 minute handling capacity of 14% of the building population is considered to be very good.

In this section we will use this pragmatic approach to evaluate the performance increase that is possible by using destination call systems and advanced control algorithms. As our main measures of service quality we use the median $\alpha_{0.5}$ and the 0.9-quantile $\alpha_{0.9}$ of the waiting time and the number of call-reissues that were necessary, measured as a percentage of the total number of calls. Table 2.2 on page 28, taken from the book of Barney [Bar02], shows that the elevator performance is considered "fair" if the waiting time median is at most 25 seconds and the 0.9-quantile at most 55 seconds. We will use these values and a limit of 10% reissues as our thresholds, i. e., the handling capacity is the largest percentage of building population for which the system achieves a waiting time median of at most 25 seconds, a 0.9-quantile of at most 55 seconds and not more than 10% of the passengers have to reissue their calls.

As indicated before, we will not consider systems with full boarding control, since the high-intensity traffic situations lead to extremely many requests in a snapshot problem, namely one per destination call which may be up to 100. Such a large number of requests cannot be handled efficiently by our algorithms.

We implemented our algorithms and the simulation environment in C++, using the SCIP framework [Ach07, ABK$^+$] to implement the IP-based ExactReplan algorithm, using CPLEX [CPL] as LP solver. All computations ran under Linux on a system with an Intel Core 2 X9650 CPU with 3.0 GHz and 8 GB of RAM. We did not use the 64 bit facilities of this machine, but compiled 32 bit code using GCC 4.3 [GCC].

In all the tests performed we used the following settings for the cost structure of the snapshot problem, which were selected ad-hoc without making an attempt to optimize the resulting behavior. The service cost coefficients were set to $c_{\text{wait}}(c) = 2$, $c_{\text{ride}}(c) = 0$, and $c_{\text{travel}}(c) = 1$ for all calls c, reflecting that waiting time is more important than travel time. The results reported later compare the difference due to penalizing the service times either linearly or quadratically. In any case, the capacity penalty c_{capacity} corresponds to the waiting time cost of 120 seconds. Finally, the round trip time penalty function $c_{\text{RTT}} \colon \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ (henceforth called RTT penalty, for short) is defined as follows. Let N

be the number of floors of the building and m the number of elevators in the group. We compute a *critical travel time* T, which is the average time needed to serve $\lceil (N-1)/m \rceil$ floors from the main floor, i. e., we assume that the destination floors travelled to from the main floor are distributed evenly between all elevators. Note that this time includes the times for unloading a full cabin as well as the time to return to the main floor. The RTT penalty c_{RTT} is of the form

$$c_{\mathrm{RTT}}(t) = \begin{cases} a_1 t & t \leq T, \\ \mathrm{e}^{a_2 t} & t \geq T. \end{cases}$$

The parameters a_1 and a_2 are set such that $c_{\mathrm{RTT}}(1.5T)$ corresponds to the waiting cost of one hour. We remark that this setup is somewhat arbitrary and its main point is to discourage to serve more than the average number of floors from the main floor.

We start our evaluation by investigating how our exact algorithm ExactReplan performs on snapshot problems for the different traffic situations.

3.3.1. Solving the snapshot problem

To generate a selection of snapshot problems as a test set for ExactReplan, we performed a simulation run of five minutes traffic in a 23-floor building ("building B" in Table 3.5 on page 87) for six typical traffic patterns. These include the traffic patterns interfloor, up peak, down peak and lunch peak mentioned earlier. In addition, we also consider the patterns *real up peak* and *real down peak*, which mix the up peak and down peak traffic which 5% of interfloor and 5% of down peak and up peak traffic, respectively. These two patterns are supposed to model the real traffic conditions more closely than the pure ones. As traffic intensity, we chose 16% of the building population in five minutes, which represents a high load situation and thus a stress test for our algorithm. We used these 16% for all traffic patterns except for interfloor traffic, where we assumed 10% since interfloor traffic usually has a lower intensity.

From the recorded snapshot problems of the simulation runs, we selected 10 snapshot problems for each of the six traffic patterns. The selection was done such that we obtained the instances with the longest running time, featuring many requests and calls, but such that the reoptimization times of the selected instances are at least 10 seconds apart to ensure a certain diversity. These snapshot problems thus do not reflect typical performance, but rather point to performance limitations of the algorithm and are thus suitable for investigating which techniques may improve the performance. Table 3.1 summarizes some statistics of our test set. Note that in each of the intermediate assignment snapshot problems there is only one unassigned request, due to the fact that reoptimization is invoked after each new destination call.

In a first test, we investigated how much the optional features *reuse of old schedules* and *early stopping* improve the performance (see Table 3.2). In this experiment, we used only the greedy lower bound. When early stopping was active, we stopped the pricing process as soon as a gap of 1% was reached. As Table 3.2 reveals the algorithm in its default variant without reuse of old schedules and early stopping is already very fast for

traffic pattern	calls			requests			unassigned requests		
	min	avg	max	min	avg	max	min	avg	max
IA, Interfloor	35	42.1	48	15	17.0	19	1	1	1
IA, Real Down Peak	52	59.0	70	14	15.6	19	1	1	1
IA, Down Peak	51	64.4	80	13	16.2	21	1	1	1
IA, Lunch Peak	63	69.1	78	15	17.4	19	1	1	1
IA, Real Up Peak	58	76.6	91	6	10.5	14	1	1	1
IA, Up Peak	55	65.3	79	2	3.6	5	1	1	1
DA, Interfloor	29	36.3	42	14	16.4	19	14	16.4	19
DA, Real Down Peak	24	46.6	57	13	15.1	19	13	15.1	19
DA, Down Peak	36	51.4	70	13	14.5	18	13	14.5	18
DA, Lunch Peak	50	63.9	77	16	19.7	28	16	19.7	28
DA, Real Up Peak	63	71.8	79	17	22.4	28	17	22.4	28
DA, Up Peak	30	45.3	59	11	13.7	18	11	13.7	18

Table 3.1.: *Overview of the size of the snapshot problems in our test set. Shown are the minimum, average and the maximum number of calls, requests and unassigned requests, each for the immediate assignment system (IA) and the delayed assignment system (DA). In each of the IA snapshot problems there is only one unassigned request, due to the fact that reoptimization is invoked after each new destination call.*

most situations. It solves all immediate assignment (IA) snapshots in significantly less than a second; the same holds for delayed assignment (DA) snapshots from interfloor and down peak traffic. Reuse of old schedules reduces the maximum running times by roughly 20–50%, which is significant. Combined with early stopping, the maximal running times of the critical traffic situations up peak and real up peak are reduced to 8% and 44% of the default variant's running time. These are then the only traffic situations which require (significantly) more than one second to solve to provable optimality with at most 1% gap.

Our second experiment assesses the quality of the lower bound methods. In contrast to the first experiment, we ignore the capacity and RTT penalty costs here, since they are only accounted for by the greedy lower bound. Table 3.3 shows the results for using the three bounding methods as primary lower bounds on the immediate assignment snapshots. For the state-space relaxation bound we use time discretization $\delta = 4$ s and we limit the number of drop floors to $k = 3$. We test these parameters once with *time reduction* (see page 78) turned off and once turned on. All methods perform very well which is not too surprising, since the search trees are not large as there is only one request to incorprate in the existing schedules. However, the greedy lower bound performs best, followed by the Tanaka bound. Using or not using the time reduction in the state-space lower bound has not a big impact on the quality of the bounds, which can be seen from the number of nodes, but effects the running time strongly.

The results for the delayed assignment (DA) snapshots are shown in Table 3.4. It turns out that in almost all of the DA snapshots there are no fixed requests, rendering the Tanaka bound useless, so we omit it here. We tested the state-space relaxation bound with the same settings as for the IA snapshots, setting a time limit of one hour for the pricing

scenario	nodes		time [s]	
	avg	max	avg	max
immediate assignment, default settings				
IA, Interfloor	59	76	0.00	0.01
IA, Down Peak	86	135	0.00	0.01
IA, Real Down Peak	88	167	0.00	0.01
IA, Lunch Peak	87	187	0.01	0.01
IA, Up Peak	11	14	0.00	0.00
IA, Real Up Peak	25	38	0.00	0.01
delayed assignment, default settings				
DA, Interfloor	7,703	17,820	0.18	0.44
DA, Down Peak	9,141	21,234	0.19	0.46
DA, Real Down Peak	14,030	73,903	0.29	1.56
DA, Lunch Peak	45,052	152,366	1.09	3.97
DA, Up Peak	2,002,852	9,915,185	96.08	527.84
DA, Real Up Peak	15,736,032	45,818,844	765.53	2,421.16
delayed assignment, reuse of old schedules				
DA, Interfloor	4,655	11,431	0.11	0.27
DA, Down Peak	5,152	16,316	0.11	0.35
DA, Real Down Peak	5,927	35,429	0.13	0.76
DA, Lunch Peak	25,639	64,625	0.61	1.62
DA, Up Peak	1,137,871	4,864,135	52.98	257.73
DA, Real Up Peak	13,266,581	37,115,554	648.52	1,909.64
delayed assignment, reuse of old schedules, early stopping				
DA, Interfloor	2,952	6,098	0.07	0.16
DA, Down Peak	3,132	12,943	0.07	0.29
DA, Real Down Peak	3,403	20,845	0.08	0.44
DA, Lunch Peak	11,408	29,686	0.29	0.77
DA, Up Peak	258,925	1,011,181	11.47	45.75
DA, Real Up Peak	4,430,791	20,385,333	218.23	1,076.69

Table 3.2.: *Performance of our exact reoptimization algorithm* ExactReplan *on the snapshot problem test set. Shown are the total number of nodes in all search trees, i. e., both for the initial solution and the pricing phase. With the default settings, the running times for the immediate assignment system are already more than sufficient. Reusing schedules from the last reoptimization run and checking the LP gap via the Lasdon bound (see Theorem 3.2.1) to stop the pricing process early significantly improves the running time.*

scenario	nodes		time [s]	
	avg	max	avg	max
greedy lower bound				
IA, Interfloor	59	76	0.00	0.01
IA, Down Peak	86	135	0.00	0.01
IA, Real Down Peak	88	167	0.00	0.01
IA, Lunch Peak	87	187	0.00	0.01
IA, Up Peak	11	14	0.00	0.01
IA, Real Up Peak	25	38	0.00	0.01
Tanaka lower bound				
IA, Interfloor	71	101	0.00	0.01
IA, Down Peak	119	187	0.01	0.01
IA, Real Down Peak	118	227	0.00	0.01
IA, Lunch Peak	109	261	0.01	0.01
IA, Up Peak	11	14	0.00	0.00
IA, Real Up Peak	25	37	0.00	0.01
state-space relaxation lower bound, with time reduction				
IA, Interfloor	80	189	0.01	0.02
IA, Down Peak	114	185	0.01	0.01
IA, Real Down Peak	124	245	0.01	0.01
IA, Lunch Peak	92	227	0.00	0.01
IA, Up Peak	11	14	0.00	0.01
IA, Real Up Peak	23	30	0.00	0.01
state-space relaxation lower bound, without time reduction				
IA, Interfloor	79	187	0.02	0.04
IA, Down Peak	113	181	0.01	0.02
IA, Real Down Peak	121	242	0.01	0.02
IA, Lunch Peak	87	199	0.02	0.07
IA, Up Peak	11	14	0.00	0.01
IA, Real Up Peak	23	30	0.00	0.01

Table 3.3.: *Performance of the* ExactReplan *algorithm employing different lower bound methods for solving the immediate assignment system snapshot problems. The state-space relaxation lower bound was computed with time discretization* $\delta = 4$ s *and the number of drop floors limited to* $k = 3$*.*

time of ExactReplan. In addition to nodes and running time data, Table 3.4 provides information on how often the state-space relaxation bound exceeded the greedy bound (it was "better") and how often this led to pruning the node ("prunes"). We can see that the state-space relaxation bound is roughly in 40% of its invocations better than the greedy bound for all traffic types except up peak. It can prune the node in 7% to 18% of its invocation for these traffic types. Unfortunately, it is not helpful for the critical up peak traffic types, where it is better much less frequently and pruning is even rarer. On second thought this is not too surprising, as the information used by the greedy bound already captures the up peak behavior rather accurately. Note that the lower number of nodes of the state-space relaxation bound for the up peak traffic types is caused by stopping due to a timeout frequently. The quality of the state-space relaxation bound does not change much if time reduction is used, whereas the running time decreases drastically. Still, the running times of the state-space relaxation bound are much inferior to that of the greedy bound. We remark that the state-space relaxation idea might still be useful to improve the algorithm in more sophisticated ways than just to compute a lower bound at each node. For instance, one might extend it to a *decremental state-space relaxation* pricing algorithm as suggested by Righini and Salani [RS08].

Our results indicate that ExactReplan using the greedy lower bound only, together with reusing old schedules and early stopping, gives the best running time results. We will therefore use this setting in our simulation experiments.

3.3.2. Simulation results

In our simulations we consider two example buildings / elevator systems whose data is given in Table 3.5. Building A has 12 floors served by a group of eight elevators. Its population is 3,300 persons, which is relatively high. In contrast, the 23-floor building B has only a population of 1,400 persons. The elevator group in building B consists of six elevators with a significantly higher maximum speed than that of the elevators in building A. This high maximum speed is necessary to provide a reasonable service in such a tall building. We are interested in how much the handling capacity of both elevator systems can be improved by applying the advanced destination control algorithms developed in the preceding sections.

We investigate this by studying the service quality achieved by our algorithms on up peak and real up peak traffic. For each building, traffic type and population percentage from 6% to 20% we generated ten realizations of five minutes of traffic. In the following simulation experiments, the elevators immediately returned to the main entrance floor once they served their schedule. This strategy is very important to obtain reasonable up peak traffic performance.

In a first extensive set of simulations, we considered the effect of the various cost structure variants on the service quality and thus on the handling capacity. We simulated the BestInsert algorithm on each of the traffic realizations for the following cost structure settings.

service Only service costs, i. e., waiting and travel costs, are incurred.

scenario	better	prunes	nodes		time [s]	
			avg	max	avg	max
greedy lower bound						
DA, Interfloor			5,881	12,536	0.12	0.28
DA, Down Peak			5,721	13,572	0.11	0.25
DA, Real Down Peak			9,632	52,364	0.18	0.98
DA, Lunch Peak			20,812	78,117	0.45	1.78
DA, Up Peak			786,979	4,605,010	35.17	223.83
DA, Real Up Peak			3,189,122	9,083,347	138.68	384.00
state-space relaxation lower bound, with time reduction						
DA, Interfloor	44.25%	7.59%	4,029	8,259	2.28	9.51
DA, Down Peak	39.32%	7.62%	4,243	8,464	0.71	1.81
DA, Real Down Peak	47.70%	12.45%	5,847	27,308	1.13	6.95
DA, Lunch Peak	44.17%	10.04%	16,361	49,574	17.12	89.66
DA, Up Peak	2.56%	0.02%	163,288	427,530	1,350.34	3,601.00^{3}
DA, Real Up Peak	5.54%	0.55%	324,254	758,678	2,789.19	3,600.68^{6}
state-space relaxation lower bound, without time reduction						
DA, Interfloor	47.16%	7.11%	3,709	6,024	76.10	389.25
DA, Down Peak	42.28%	9.33%	4,275	8,701	27.27	122.57
DA, Real Down Peak	57.65%	18.64%	5,340	19,521	39.52	257.99
DA, Lunch Peak	45.62%	9.61%	11,878	22,192	714.20	3,608.67^{1}
DA, Up Peak	7.80%	0.05%	25,755	43,611	2,799.16	3,851.45^{7}
DA, Real Up Peak	12.40%	0.88%	20,102	53,103	3,626.29	3,723.87^{10}

Table 3.4.: *Performance of the* ExactReplan *algorithm employing different lower bound methods for solving the delayed assignment system snapshot problems. "better" counts how often the state-space relaxation lower bound was better than the greedy one, "prunes" counts how often it led to pruning the node when the greedy lower bound did not prune the node. The state-space relaxation lower bound was computed with time discretization $\delta = 4$ s and the number of drop floors limited to $k = 3$. The small number at the top of the maximum running time indicates the number of terminations due to a timeout.*

	building A	building B
population	3300	1400
floors	12	23
elevators	8	6
cabin capacity	19	13
acceleration [m/s^2]	1.0	0.9
maximum speed [m/s]	2.0	5.0
deceleration [m/s^2]	1.0	0.9

Table 3.5.: *Details of the elevator systems in the buildings considered.*

service + cap In addition to service cost also capacity penalty cost are incurred.

service + cap + RTT Service costs, capacity costs, and RTT penalty cost are in effect.

service + cap + mmf The cost are as in the "service + cap" variant. However, there is a hard limit for each elevator on how many destination floors it accepts at the main floor. As the RTT penalty, this limit shall reduce the round trip time and thus increase the service level at the main floor. The limit is only implemented in BestInsert so far.

For delayed assignment systems, we used a slightly modified version of BestInsert that actually constructs two alternative dispatches and chooses the better one in each reoptimization step. The alternative dispatch to inserting only the new requests in the existing schedules is constructed by inserting *all* requests in the order of their release times into the empty schedules. Thus BestInsert has the chance to reconsider some of its decisions and thus can use the freedom offered by the delayed assignment.

The service characteristics resulting from these simulation runs are reported in the Appendix in Tables A.1 to A.8. We briefly summarize the most important observations here. In general, one can say that

- The delayed assignment (DA) system gives better waiting times than the immediate assignment (IA) system, if sometimes only by a slight margin.
- The DA system necessitates a higher number of reissues.
- Planning with quadratic service cost reduces the number of reissues in comparison to planning with linear service cost.
- Compared with planning with linear service cost, planning with quadratic service cost leads to worse waiting time for low load and better waiting times for high load.

For the specific cost settings, we observe the following.

service The fraction of reissues is almost the same for the IA and the DA system. This fraction becomes rather high for relatively low loads and thus is the limiting factor for the handling capacity.

It is clear that the limited capacity of an elevator becomes an issue when the load increases. Since limited capacity is not considered in any way, the control algorithm allocates too many passengers to the same elevator, thus forcing many reissues.

service + cap The number of reissues can be drastically reduced in comparison to the "service" setting. In addition, the DA system achieves significantly better waiting times than the IA system for higher loads. Moreover, on building A the DA system outperforms the IA system significantly.

The reduction of reissues shows that penalizing exceeding the elevator capacity has the desired effect of avoiding such request assignments. Considering cabin capacity also improves the accordance between planning and the realized future, avoiding long waiting times due to unnecessary reissues. This explains the positive effect of the capacity penalty on the waiting times. Moreover, since building A exhibits much higher traffic intensity (up to 660 passengers for 20% traffic), the advantages of the DA system become more apparent here.

service + cap + RTT For low loads, the waiting time median $\alpha_{0.5}$ is zero. The DA system offers significantly better waiting times than the IA system. However, the fraction of reissues is quite high for the DA system and limits the handling capacity. There are more reissues than with the "service + cap" setting.

The reason for the zero median is that there is always an elevator present at the main floor, which is a side effect of the RTT penalty: If the RTT penalty is not yet high, it is more advantageous to increase it a little bit for one elevator by assigning it an additional destination floor than to increase it much to send an idle elevator there. This means that an elevator is assigned several floors before any of the other elevators is used, which is desirable.

service + cap + mmf The waiting times are worse than in the "service + cap" setting for low load, but better for high load. The number of reissues in the DA system is lower than for both the "service + cap" and the "service + cap + RTT" settings, thus enabling higher handling capacities.

The increase of the waiting times for low load is due to avoiding many destination floors served from the main floor, which does not hurt in low load. For high load, however, this pays off and reduces the round trip times and thus also the waiting times. In contrast to the RTT penalty, there is no mechanism that encourages leaving elevators at the main floor.

Using our definition of "reasonable" service quality meaning that the waiting time median $\alpha_{0.5}$ is at most 25 seconds, the 0.9-quantile $\alpha_{0.9}$ is at most 55 seconds, and at most 10% of the passengers have to reissue their call, we obtain from Tables A.1 to A.8 the handling capacities shown in Table 3.6. Since these thresholds are sometimes missed by a small margin only and the next precentage level gives much worse results, we were less strict and actually used 58 seconds and less than 11% as the critical thresholds.

From the table we see that incorporating the capacity helps to increase the handling capacity. In both buildings, the highest handling capacities are achieved with the "service + cap + mmf" setting. The second best setting for building A is "service + cap", whereas for building B it is "service + cap + RTT", which performs rather poor in building A. The differences between the best and the worst setting for each system (IA / DA) and traffic (Up / Real Up) are between 2% and 5% of the building population, which is quite significant. It is interesting to observe that the quadratic service costs consistently (except for "service + cap + mmf") lead to a higher handling capacity than the linear service costs. The reason is that quadratic service costs penalize high waiting times stronger, thus helping to reduce the critical 0.9-quantile $\alpha_{0.9}$.

The question is now whether the handling capacity can be further improved by using our exact algorithm ExactReplan. Since the limit on the number of destination floors served from the main floor is not yet supported by our implementation, we decided to investigate the "service + cap" setting instead of the "service + cap + mmf" setting more deeply. We simulated ExactReplan in the critical range of the handling capacity, admitting a computation time of at most one second per reoptimization run. The full results are reported in Tables A.9 and A.10. In summary, we can say that

- For the IA system, the values obtained by ExactReplan are almost the same as those obtained by BestInsert, indicating that BestInsert finds (almost) the same

(a) Building A.

		IA system		DA system	
cost setting		linear	quadratic	linear	quadratic
service	Up	*10%*	*12%*	*10%*	*12%*
	Real Up	*10%*	*11%*	*9%*	*11%*
service + cap	Up	13%	14%	13%	14%
	Real Up	13%	13%	13%	13%
service + cap + RTT	Up	11%	12%	*11%*	*11%*
	Real Up	10%	10%	13%	13%
service + cap + mmf	Up	14%	14%	15%	15%
	Real Up	13%	13%	14%	13%

(b) Building B.

		IA system		DA system	
cost setting		linear	quadratic	linear	quadratic
service	Up	*13%*	*15%*	*14%*	*15%*
	Real Up	*13%*	*14%*	*13%*	*15%*
service + cap	Up	16%	16%	16%	16%
	Real Up	15%	15%	15%	15%
service + cap + RTT	Up	14%	15%	17%	17%
	Real Up	13%	14%	15%	15%
service + cap + mmf	Up	16%	16%	17%	17%
	Real Up	15%	15%	16%	15%

Table 3.6.: *Handling capacities achieved by the* BestInsert *algorithm for various cost function variants. Italic values indicate the number of reissues limits the handling capacity. In all other cases, the handling capacity is limited by the* 0.9*-quantile* $\alpha_{0.9}$ *of the waiting time.*

dispatches. Since ExactReplan manages to find the optimal dispatches within one second, BestInsert's dispatches are almost optimal, too.

- For the DA system, where there are much more decisions to take during each reoptimization run, ExactReplan is able to significantly improve over BestInsert. It even attains a higher handling capacity in four out of eight cases – mainly for quadratic service cost, which reduces $\alpha_{0.9}$ as described above. However, many of the high load snapshot problems cannot be solved to optimality within one second.

Table 3.7 contrasts the handling capacities of ExactReplan with that of BestInsert. Moreover, it shows the handling capacities achieved by the CGC algorithm controlling a conventional system and the best handling capacity of any of the tested algorithms. We see that ExactReplan always attains the best handling capacity, except for the up peak traffic in building A controlled by a DA system, where it is outperformed by BestInsert with setting "service + cap + mmf". This suggests that further improvement is possible by incorporating the destination floor limit in the ExactReplan algorithm. Comparing with CGC, the handling capacity in building A can be more than doubled by using a destination call system, whereas it can be improved by at least 66% in building B. Since CGC is not specifically designed towards up peak traffic, we have to assume that other conventional control algorithms might perform better. Still, improving the handling capacity by something like 50% by replacing a conventional system with a destination control system can be expected from our results. The improvement possible by installing a DA instead of a IA system is not as significant.

There are some questions for further research. For instance, it is unclear why there are more reissues for the DA system and whether they can be avoided. Moreover, so far our simulation neglects the time needed for reoptimization, i. e., the new dispatch becomes effective immediately after a new call. In practice, reoptimization takes some time during which several calls may arrive, thus opening up more optimization potential for the IA system by having more than one request to assign. It remains to see whether BestInsert still delivers almost optimal dispatches in this situation. However, the running times of ExactReplan on DA snapshots with at most five requests are still below one second, which indicates that it will perform well in the IA system when there is more than one unassigned request per snapshot problem, too. We also need to see whether our techniques extend to more complex traffic patterns, e. g., in buildings with additional entrance floors such as parking floors.

Finally, there is the issue of the limited elevator capacity. We saw that by assuming a one-to-one correspondence between passengers and destination calls the handling capacity can be increased significantly. Therefore, one should definitely employ this in practice. An important question for practical use is how robust the scheduling decisions are to violations of this one-to-one assumption, e. g., due to misuse. Both the RTT penalty and the limit on the number of destination floors do to some extent encourage not to fill up a cabin completely, but to assign passengers to other elevators earlier. This effect might increase the robustness of the schedules w. r. t. to misuse.

(a) Building A.

	CGC	IA system					DA system				
		linear		quadratic			linear		quadratic		
		BI	ER	BI	ER	**best**	BI	ER	BI	ER	**best**
Up	*6%*	13%	13%	14%	14%	**14%**	13%	14%	14%	14%	**15%**
Real Up	< *6%*	13%	13%	13%	13%	**13%**	13%	13%	13%	14%	**14%**

(b) Building B.

	CGC	IA system					DA system				
		linear		quadratic			linear		quadratic		
		BI	ER	BI	ER	**best**	BI	ER	BI	ER	**best**
Up	*9%*	16%	16%	16%	16%	**16%**	16%	16%	16%	17%	**17%**
Real Up	*9%*	15%	15%	15%	15%	**15%**	15%	15%	15%	17%	**17%**

Table 3.7.: *Resulting handling capacities for the various considered elevator systems and elevator control algorithms. As before, italics indicate that the number of reissues limits the handling capacity, whereas in all other cases the handling capacity is limited by the* 0.9-*quantile* $\alpha_{0.9}$ *of the waiting time. The column "best" gives the best result of any algorithm studied.*

3.4. Significance

Our research showed that the BestInsert algorithm is performing very well when controlling an immediate assignment system as it is used today. From a practical point of view it is nice that this relatively simple algorithm can effectively exploit the capacity offered by an elevator group, since it is much easier to implement and maintain than an exact reoptimization algorithm. Since delayed assignment systems do not offer a significant advantage for the load levels seen in practice it is unlikely that they will be introduced, given that they would also require some familiarization from the passengers.

It is very valuable to know approximately how far the handling capacity of an existing conventional system can be improved by replacing it with a destination call system. Replacing the control system is a much cheaper alternative than to upgrade the elevator drives to faster ones or install additional elevators and it is sometimes the only option.

A variant of the BestInsert algorithm has been implemented by our industry partner Kollmorgen Steuerungstechnik and is in use today. Our results will help to improve this algorithm further.

Part II.

A stochastic dominance approach to the analysis of online algorithms

Chapter 4.

Stochastic dominance analysis of online algorithms

This chapter introduces a novel approach for the probabilistic analysis of algorithms for combinatorial online optimization problems. The main motivation for designing this approach is that the results obtained using the standard analysis tool, *competitive analysis* [ST85], often do not reflect performance differences observed in practice.

Throughout this chapter, we will use the following three online optimization problems as examples. The first two problems are classical problems, whereas the third one, online bin coloring, arose in application projects at ZIB. A connection of the online bin coloring problem to the control of elevators is discussed in Section 5.1.

Paging Paging models an optimization problem occuring in a computer with a two-level memory system. The *slow memory* stores a fixed set M of pages. To speed up access to the pages, up to k pages can be put in the *fast memory*, also called *cache*. The task is to serve a sequence $\sigma \in M^n$ of n page requests. In order to serve a page request $p \in M$, the page has to be in the cache, otherwise a *page fault* occurs. The requested page must then be loaded into the cache, and, if the cache contains k pages, at least one page must be evicted from the cache. An online paging algorithm needs to decide which page(s) will be evicted from the cache on a page fault without knowing the remaining request sequence or its length. The goal is to minimize the number of page faults.

Bin Packing The input consists of a sequence of items, described by their item sizes $l_1, \dots, l_n \in \{1, \dots, B\}$, that need to be packed in bins of capacity $B \in \mathbb{N}$, i. e., the set of items put in a bin has total size at most B. The goal is to minimize the total number of bins used. In the online version, the items arrive one-by-one and each item has to be put into a partially filled bin that still has sufficient capacity or a new bin has to be opened. Once an item has been packed in one bin, it is not possible to repack it into another bin. An online algorithm does not know the number of items to come nor their sizes.

In the bounded-space variant, at most k bins may be open at the same time. If none of the k open bins provides sufficient capacity for the new item, one of these bins needs to be closed and replaced by an empty bin. Closed bins are no longer available for packing items. Note that it is also allowed to close a bin if there is some bin with enough remaining capacity.

Bin Coloring [KdPSR01] The input consists of a sequence of unit-sized items, each of which has one of C colors. These items need to be packed sequentially into one of m initially empty bins of capacity B. As soon as a bin is *full*, i. e., has exactly B items, it is replaced by an empty one. As for bin packing, repacking is not allowed. An online algorithm must decide upon the bin for each item without knowing future item colors. The goal is to minimize the maximum number of different colors in one bin, which is called *colorfulness*.

Note that all these problems are minimization problems. Without further notice, we will always assume that the goal is to minimize some objective.

In the next section, we give an introduction to competitive analysis and we sketch the main idea of our new stochastic-dominance-based approach to the analysis of online algorithms. We then review related work and other alternative ways for analyzing online algorithms. Section 4.3 introduces our approach in detail, highlighting its main properties. We perform an analysis of paging algorithms in Section 4.4, establishing several interesting stochastic dominance results. As a byproduct, we reprove a number of known results in a very straightforward way, thus simplifying, strengthening and unifying former work.

4.1. Introduction

In standard competitive analysis [ST85, BEY98, FW98], an online algorithm ALG is compared with a hypothetical optimal offline algorithm OPT. OPT has access to the entire request sequence, i. e., it knows the future and is thus not an online algorithm, and is supposed to serve this sequence optimally, i. e., at minimum cost. Competitive analysis measures how much the performance of ALG degrades w. r. t. OPT in the worst case as a consequence of the lack of information about the future. For a request sequence σ, we denote the corresponding optimal offline cost and the cost of ALG by $\mathsf{OPT}(\sigma)$ and $\mathsf{ALG}(\sigma)$, respectively. The algorithm ALG is said to be c-competitive for $c \geq 1$, if

$$\mathsf{ALG}(\sigma) \leq c \cdot \mathsf{OPT}(\sigma) + b \tag{4.1}$$

for all request sequences σ and some $b \geq 0$. The *competitive ratio* of ALG is the infimum value c such that ALG is c-competitive. ALG is called *competitive* if ALG is c-competitive for some constant $c \geq 1$ that does not depend on the length of the sequence. Frequently, only the case $b = 0$ is considered, which makes the results less asymptotic.

We will briefly discuss the results of competitive analysis for the three online optimization problems introduced above.

Paging Well-known paging algorithms are flush-when-full (FWF), which evicts every page on a page fault, first-in-first-out (FIFO), which evicts the page that has been in the cache the longest, and least-recently-used (LRU), which evicts the page whose most recent request was earliest.

Sleator and Tarjan [ST85] showed that the competitive ratio of every online algorithm is at least the cache size k. They also proved that both LRU and FIFO attain this ratio and are thus optimal, whereas some other algorithms are not k-competitive. Later it was shown that any online algorithm from the class of *marking algorithms* is k-competitive [Tor98], which includes FWF and LRU.

These results did not match empirical observations: LRU was observed to be much better than FWF and to outperform FIFO (see e. g., [You94]). Moreover, the "empirical" competitive ratio of LRU was much smaller than k.

Bin Packing Probably the simplest online algorithm for bin packing is NextFit, which looks only at the most recently opened bin. If the next item fits, it is put into this bin; if not, the item is put in a new bin and the other bin is never considered again. It is shown by Johnson [Joh74] that the competitive ratio of NextFit is 2. The algorithm FirstFit scans through the bins in their opening order, putting the item in the first bin with sufficient capacity. If no bin is found, a new bin is opened to accomodate the item. BestFit works similar, but puts the item in the open bin with least remaining capacity that suffices for the item. Both FirstFit and BestFit have competitive ratio 17/10 [JDU+74], but BestFit gives better results in practice.

Well-known algorithms [GW95] for the bounded-space bin packing problem are the algorithms Next-k-Fit (or NF_k for short) and k-bounded BestFit (BBF_k for short). NF_k puts the new item in the first (in the order of opening) of the k bins which has sufficient capacity. If there is no such bin, it closes the first bin and opens a new bin containing the item. BBF_k, in contrast, puts the item in the *fullest* bin in which it will fit; also the fullest bin is closed if there is no bin with sufficient capacity. As shown by Csirik and Imreh [CI89] and Mao [Mao93], NF_k has a competitive ratio of $17/10 + 3/(10k - 10)$. BBF_k has, independently of k, a competitive ratio of 17/10 [CJ01], so BBF_k is better than NF_k, if only by a slight margin. However, Lee and Lee [LL85] established a lower bound of roughly 1.691 for the competitive ratio of *any* bounded-space algorithm. Thus all reasonable algorithms have almost the same competitive ratio.

Bin Coloring A natural algorithm for this problem is the algorithm GreedyFit: it packs an item with an already present color in the bin with that color and otherwise chooses a bin which currently has the least number of different colors. Another simple algorithm, OneBin, packs all items in the same bin. Krumke et al. [KdPSR01] showed that the competitive ratio of GreedyFit is at least $2m$, the competitive ratio of OneBin is at most $2m - 1$. The authors report that this result contrasts the clear dominance of GreedyFit over OneBin observed in simulations.

The common failure of competitive analysis to discriminate between algorithms that perform (even significantly) differently in practice is due to its worst-case nature. An online algorithm has a bad competitive ratio even if there is just a single sequence on which the online algorithm incurs much higher cost than OPT. In order to overcome this drawback, a number of variations of and alternatives to competitive analysis have been proposed, see Section 4.2.1 for a brief overview or e. g., [Alb03, DLO05, BF07] for some

	σ_1	σ_2	σ_3	σ_4	σ_5
A	3	2	3	4	2
B	4	3	4	3	2

(a) Objective values obtained by A and B on some input sequences.

	≤ 1	≤ 2	≤ 3	≤ 4
A	0	2	4	5
B	0	1	3	5

(b) Empirical distribution of objective values obtained by A and B.

Figure 4.1.: *Illustration of the idea of comparing two online algorithms* A *and* B *using stochastic dominance. By looking at the* distribution *of the objective values it becomes apparent that algorithm* A *is in general better than algorithm* B. *The empirical distribution plays in this example the role that the distribution function plays in our probabilistic analysis.*

recent surveys. Much of this work sticks to worst case analysis or considers randomization on the side of the online algorithm in order to improve the competitive ratio.

Another alternative is to drop the worst-case view and assume that the input sequences are generated randomly and to perform a probabilistic analysis. So far all probabilistic analyses of online algorithms are more or less average case analyses, i. e., they focus on the *expected* performance of an algorithm. In our novel approach to probabilistic analysis, we apply stochastic orders, in particular the stochastic dominance order, to derive results that compare the performance of algorithms. A random variable X is said to be *stochastically dominated* by a random variable Y, written $X \leq_{\mathrm{st}} Y$, if

$$\operatorname{Prob}[X \leq x] \geq \operatorname{Prob}[Y \leq x] \quad \text{for all } x \in \mathbb{R}.$$

Intuitively, this means that X has “more probability mass” on small values than Y, so it is in a probabilistic sense smaller than Y. This concept is illustrated in Figure 4.1.

Suppose we can describe the objective values of two online algorithms A and B after n steps by random variables χ_n^{A} and χ_n^{B}, respectively. We can then say that A is *stochastically better* than B w. r. t. to the considered input distribution if $\chi_n^{\mathsf{A}} \leq_{\mathrm{st}} \chi_n^{\mathsf{B}}$ for all $n \in \mathbb{N}$ (assuming a minimization problem). A main feature of this approach is that it judges an algorithm not by a single worst-case or average value, but by its objective value distribution induced by the random sequences. We will see that this approach sometimes admits quite a strong deterministic interpretation as well.

Our distribution-based analysis allows to prove stronger and more realistic results for the paging and the bin coloring problems than the competitive analysis results sketched above. For the paging problem, we show in Section 4.4 that the algorithm LRU is actually optimal on a large class of relevant request sequences. In Chapter 5 we establish that the maximum colorfulness obtained by GreedyFit is stochastically dominated by that of OneBin, indicating that GreedyFit is better than OneBin.

4.2. Related work

In this section we give an overview on related work. There are basically two kinds of research we concentrate on. First we review alternative measures for evaluating and designing online algorithms. Since it turns out that for the purposes of our analysis it is useful to model the working of an online algorithm using a Markov chain, we will also summarize work on the uses of Markov chains and / or stochastic dominance in the analysis of algorithms.

4.2.1. Measures for online algorithms

We already mentioned that competitive analysis results often do not reflect (significant) performance differences between online algorithms that are observed empirically. This observation has stimulated a lot of research on alternative measures. In this overview, we focus on approaches that use randomness to provide more meaningful results.

Competitive analysis of online algorithms can be viewed as a two player game. The *online player*, i. e., the online algorithm, tries to achieve a small competitive ratio, whereas the other player, the *adversary*, tries to generate the request sequence such that he can serve it with small cost, but incurring comparatively high cost for the online player, which gives a large competitive ratio.

Basically there are three approaches to remedy the drawbacks of competitive analysis in order to get more realistic and useful results.

1. Keep the idea of comparing to the offline optimum, but reduce the power of the adversary online algorithms are compared to.
2. Compare online algorithms to a weaker optimum, and use this to compare the algorithms. An example for this approach is the paper by Karlin et al. [KPR00], where an optimal policy of a Markov decision process is used as the reference optimum.
3. Compare online algorithms directly instead of indirectly via the offline optimum.

The first approach is the most prominent one. An obvious way to reduce the power of the adversary is to restrict the set of request sequences it may generate. This problem-specific technique is quite successful. Examples for paging are the restriction to sequences that exhibit locality of reference [AFG05, Tor98] or that are derived from an access graph [BIRS95]. Another possibility is to allow the online algorithm to use more resources (e. g., more bins) than the adversary [KP00]; this is known as resource augmentation. Many more deterministic approaches have been proposed, see e. g., the surveys by Albers [Alb03] and Dorrigiv and López-Ortiz [DLO05] or the overview provided by Boyar et al. [BF07].

Randomness can be used in two ways in the analysis of online algorithms. First, the request sequence may be assumed to be drawn from some probability distribution. This is equivalent to replacing the adversary by some random process. Second, the online algorithms themselves may be randomized, which makes it harder for the adversary to come up with bad sequences.

Randomized online algorithms

Using randomization to improve online algorithms has been suggested by Borodin et al. [BLS92]. Usually, analyses are done with respect to an *oblivious adversary*, i. e., this adversary knows the probability distribution employed by the randomized online algorithm, but not the random outcomes, so the adversary cannot predict what the online algorithm is going to do. The competitive ratio is then determined by replacing ALG by its expectation $\mathbb{E}\left[\mathsf{ALG}\right]$ in Equation (4.1).

For the paging problem, Fiat et al. [FKL$^+$91] showed a lower bound of H_k (H_k is the kth harmonic number) on the competitive ratio of every randomized paging algorithm. Moreover, they provided a randomized paging algorithm that is $2H_k$-competitive against the oblivious adversary. McGeoch and Sleator [MS91] presented an optimal randomized algorithm with competitive ratio H_k. Since $H_k \in \mathcal{O}(\log k)$, both results show that randomized paging algorithms can achieve a significantly better competitive ratio than deterministic ones. Similar improvements have been achieved for many online optimization problems.

Interestingly, randomized algorithms for bin packing can only be slightly better than deterministic ones. Chandra [Cha92] proves a lower bound of 1.536 for the competitive ratio of any randomized online algorithm, whereas the best-known deterministic algorithms achieves 1.589.

In the bin coloring problem, any randomized algorithm has a competitive ratio of $\Theta(m)$ [KdPSR01], so random algorithms cannot achieve a qualitative improvement over $\mathsf{GreedyFit}$ and OneBin.

Average case analysis

Probably the first alternative for worst case analysis that comes into one's mind is average case analysis. In average case analysis, the request sequence is chosen according to some probability distribution. In most average case competitive analyses, the requests are chosen independently and identically distributed. In this context, the optimal value for a request sequence of length n, OPT_n, becomes a random variable, just as ALG_n, the value obtained by an (online) algorithm. The *expected performance ratio* is defined as

$$R_{\mathsf{ALG}}(n) = \mathbb{E}\left[\frac{\mathsf{ALG}_n}{\mathsf{OPT}_n}\right], \tag{4.2}$$

i. e., as the expectation of the ratio between ALG_n an OPT_n. Note that this is in contrast to the expected competitive ratio defined by (4.4), which uses the ratio of the expectations. Scharbrodt et al. [SSS06] and Souza and Steger [SS06] elaborate on the difference between the two measures. Although the expected competitive ratio as the ratio of the expectations is normally easier to compute, they prefer the expected performance ratio, for several reasons. First, they say that expected competitive ratio favours algorithms that perform well on sequences for which the optimal solution value is large, due to the fact that sequences with small solution value contribute little to the expected value of an algorithm. On the other hand, the expected performance ratio favours algorithms that perform well on many sequences. Furthermore, by Markov's inequality, one can easily derive bounds on the probability that for a randomly generated sequence the ratio $\mathsf{ALG}_n/\mathsf{OPT}_n$ is more than a certain factor away from the expected performance ratio.

There is a vast literature on average case analysis of bin packing algorithms (see e. g., [CGJ97, CCG+02] for surveys). Many results for bin packing focus on the asymptotic expected competitive ratio of ALG, i. e.,

$$R^{\infty}_{\mathsf{ALG}}(F) := \lim_{n\to\infty} R_{\mathsf{ALG}}(n),$$

assuming that the input sequence is generated by choosing each item size i. i. d. from distribution F. In most cases, F is a uniform distribution, e. g., $U[0,1]$, the uniform distribution on $[0,1]$. The first precise average case asymptotic analysis is by Coffman et al. [CSHY80] for the $\mathsf{NextFit}$ algorithm. They showed that the asymptotic expected competitive ratio for $\mathsf{NextFit}$ is $R^{\infty}_{\mathrm{NF}}(U[0,1]) = 4/3$. Lee and Lee [LL85] proved that their Harmonic algorithm achieves $R^{\infty}_{\mathrm{H}}(U[0,1]) < 1.306$. Bentley et al. [BJL+84] were able to show, somewhat suprisingly, $R^{\infty}_{\mathrm{FF}}(U[0,1]) = 1$, i. e., that $\mathsf{FirstFit}$ is asymptotically optimal. The result was surprising since the empirical value is significantly larger than 1 in simulations, even for large numbers of items. This motivated the study of the *expected waste after n items*, i. e., the expected difference between the number of bins used by ALG and the total size of n items. Analyzing the waste, Shor [Sho86] was able to separate the performance of $\mathsf{FirstFit}$ and $\mathsf{BestFit}$: While the expected waste of $\mathsf{FirstFit}$ grows as $\Omega(n^{2/3})$, $\mathsf{BestFit}$ has expected waste $\Theta(\sqrt{n}\log^{3/4} n)$. Both $\mathsf{FirstFit}$ and $\mathsf{BestFit}$ are thus asymptotically optimal, but the performance of $\mathsf{BestFit}$ converges faster.

Diffuse adversaries

As competitive analysis is often criticized for being too pessimistic due to its worst case character, average case analysis is often considered to be too optimistic. Moreover, in many cases the probability distributions analyzed are quite special and / or realistic probabilistic models cannot be analyzed or are not available.

In order to address these issues and to improve upon competitive analysis, Koutsoupias and Papadimitriou [KP94] proposed the *diffuse adversary model*. In this model, the offline adversary is replaced by the *diffuse adversary*, which chooses a probability distribution D out of a family of distributions Δ. The class Δ of distributions may be used to express some structural property of the inputs without sticking to a certain distribution. An algorithm ALG is called c-competitive against class Δ of request sequence distributions, if there is a $b \geq 0$ such that

$$\mathbb{E}_D\left[\mathsf{ALG}(\sigma)\right] \leq c \cdot \mathbb{E}_D\left[\mathsf{OPT}(\sigma)\right] + b$$

for all $D \in \Delta$, where the request sequence σ is drawn according to D. Note that this definition generalizes both competitive analysis and average case analysis for the ratio of expectations.

To apply this approach to the paging problem, Koutsoupias and Papadimitriou propose a class of distributions Δ_ε. This class contains all distributions D such that the conditional probability $D[p|\sigma]$ of page p being requested after request sequence σ satisfies $D[p|\sigma] \leq \varepsilon$ for all p and σ. Clearly, a small ε limits the power of the adversary since he has less control over the next request (note that Δ_1 is equivalent to all request distributions). Koutsoupias and Papadimitriou show that LRU attains the optimal competitive ratio against Δ_ε for any $\varepsilon \in [0, 1]$. However, they cannot determine this ratio. In subsequent work, Young [You00] gives lower and upper bounds for the optimal competitive ratio in terms of a function $\psi(\varepsilon, k)$ that match up to a factor of two. The function $\psi(\varepsilon, k)$ exhibits the following threshold behavior around $\varepsilon = 1/k$:

$$\psi(\varepsilon, k) \text{ is } \begin{cases} \leq 1 + \ln \frac{1}{\delta} & \varepsilon = (1-\delta)/k, \\ \approx \ln k & \varepsilon = 1/k, \\ \leq k \frac{\delta}{1+\delta} & \varepsilon = (1+\delta)/k. \end{cases}$$

Thus the optimal competitive ratio (that of LRU) is constant for small ε and almost k for large ε. This bound holds for randomized algorithms, too, except for the case $\varepsilon \geq 1/k$ where both the lower and upper bounds are $\mathcal{O}(\log k)$. Moreover, Young shows that FIFO and FWF have competitive ratio k for $\varepsilon \geq 1/k$. Hence, these results generalize the standard competitive analysis results and those for randomized algorithms and are able to discriminate between FWF or FIFO and LRU.

Becchetti [Bec04] proposes a different diffuse adversary for the paging problem. The class Δ comprises distributions D whose conditional distributions $D[p|\sigma]$ favor pages p that are more recent w. r. t. σ in order to model the locality of reference often encountered in practice. He then shows that LRU achieves a constant competitive ratio against Δ, whereas that of FWF is $\Omega(k)$ if locality of reference is high.

Smoothed competitive analysis

An alternative compromise between worst case and average case analysis is to consider *smoothed* inputs. The notion of *smoothed complexity* was introduced by Spielman and Teng [ST04] in an attempt to explain the success of algorithms that are known to work well in practice while having a poor worst case performance. It can be seen as an hybrid between worst case and average case complexity. The basic idea is to randomly perturb the initial input sequences and to analyze the expected performance of the algorithm on the perturbed sequences.

Becchetti et al. [BLMS$^+$06] extended the idea of smoothed complexity to *smoothed competitive ratio*. Following the idea of Spielman and Teng, they smoothen the input sequence according to some probability distribution f. Given an input sequence $\bar{I}$, let $N(\bar{I})$ denote the set of sequences that can be obtained by smoothing the input sequence I according to f. The *smoothed competitive ratio* is defined as

$$\sup_{\bar{I}} \mathbb{E}_{I \xleftarrow{f} N(\bar{I})} \left[\frac{\mathsf{ALG}(I)}{\mathsf{OPT}(I)} \right],$$

where the supremum is taken over all input sequences $\bar{I}$ and the expectation is taken according to f over all sequences I in $N(\bar{I})$.

Becchetti et al. [BLMS$^+$06] considered the so-called multilevel feedback (MLF) algorithm for preemptive scheduling on a single machine so as to minimize the total flow-time. When all processing times are between 1 and 2^K, any nonclairvoyant deterministic online algorithm has a competitive ratio of $\Omega(2^K)$ and $\Omega(n^{1/3})$. Becchetti et al. show that when the processing times are smoothed according to a suitable natural randomization model with standard deviation σ, the smoothed competitive ratio of MLF is $\mathcal{O}\left((2^k/\sigma)^3 + (2^k/\sigma)^2 2^{K-k}\right)$. Whenever f is the uniform distribution over $[0, 2^k - 1]$, this simplifies to $\mathcal{O}\left(2^{K-k}\right)$.

Other measures

As mentioned before, one weakness of competitive analysis is the comparison to the offline optimum, which is due to the fact that in a deterministic setting there is no reasonable concept of "optimal online algorithm" that can serve as a yardstick. In contrast, such an optimal online algorithm can be defined if the input is generated by some random process.

This approach has been applied in the analysis of paging algorithms even before the advent of competitive analysis. Franaszek and Wagner [FW74] studied the paging problem with request sequences generated according to the independent reference model. In this model, each request is generated independently and identically from the same fixed page distribution. They consider the *page fault rate*, i. e., the (asymptotic) expected number of page faults per time unit, instead of the number of page faults as their performance measure for paging algorithms. It is shown that no paging algorithm from a certain class of algorithms including LRU and FIFO achieves a page fault rate that is at most a constant factor larger than the optimal one. However, it turns out that if LRU is allowed to use (slightly) larger cache than the optimal online algorithm, the ratio of the page fault rates becomes bounded, which is not true for FIFO.

Karlin et al. [KPR00] generalize this approach by considering a request sequence generated by a fixed Markov chain, which is a probabilistic version of the access graph model. Using the theory of Markov decision processes, they are able to characterize the optimal online algorithm for the given Markov chain, which is a deterministic algorithm whose decisions depend only on the current request and the state of the cache. They show that there are Markov chains such that all marking algorithms exhibit a page fault rate that is $\Omega(k)$ times the optimal one. Surprisingly, this includes LRU, which performs well under deterministic locality of reference models. The authors describe a polynomial-time algorithm whose page fault rate is not more than a constant times the optimal one. In contrast to most paging algorithms, this algorithm is not independent of the input, but depends crucially on the Markov chain generating the request sequence.

All the measures discussed so far use a single number to evaluate the performance of an algorithm, e. g., the maximum or the average. The only approach we are aware of that looks at the *distribution* of the objective values of an algorithm for various inputs to assess the algorithm is Bijective Analysis [ADLO07, AS09], which can be viewed as a special case of our stochastic dominance approach.

Very recently, Angelopoulos and Schweitzer [AS09] applied Bijective Analysis to paging and showed that LRU is the unique optimal algorithm w. r. t. to Bijective Analysis for a restricted class of sequences exhibiting locality of reference. In Section 4.4 we present an alternative proof of this result based on stochastic dominance.

4.2.2. Related applications of Markov chains and stochastic dominance

Although Markov chains are a natural tool for the study of online algorithms, they have, to the best of our knowledge, been applied only to study paging and bin packing algorithms. For paging, Karlin et al. [KPR00] studied request sequences generated by a Markov chain. This work uses Markov chains to model the input sequences and not for analyzing algorithms, but employs the theory of Markov decision processes to derive lower bounds for all online algorithms.

Coffman et al. [CSHY80] and Karmarkar [Kar82] used Markov chains to analyze the expected performance of NextFit with continuous item size distributions. They determined the expected number of bins needed to pack n items of sizes distributed according to $U(0, u)$ for $1/2 < u \leq 1$. Ramanan [Ram89] applies a similar analysis to a variant of NextFit called SmartNextFit. Recently, Naaman and Rom [NR08] used a Markov chain approach to analyze the asymptotic expected performance ratio for several bounded-space bin packing algorithms. An alternative way of measuring the performance of bin packing algorithms is to consider the growth of the *waste*, which is the bin capacity that remains unutilized by the bin packing algorithm. Coffman et al. [CJSW93] study the stability of Markov chains in order to discriminate parameters (u, B) for which the asymptotic expected waste of BestFit on $U\{u, B\}$-sequences remains bounded or grows linearly. Kenyon et al. [KRS98] and Albers and Mitzenmacher [AM00] extended their results. In contrast to our approach all mentioned results are only asymptotic and expectation-based.

Other uses of Markov chains in the analysis of algorithms are in the field of approximate sampling [Sin93]. Some of the techniques used there are similar to ours since they are based on the concept of coupling, which is very useful to compare probability distributions.

Our approach is also related to the theory of *Markov decisions processes (MDPs)* (see e. g., [Put05]). A Markov decision process is a stochastic process that can partially be controlled, i. e., at certain points in time a control algorithm can choose from a set of actions influencing the future behavior of the process. The goal is to identify an algorithm that minimizes the expected cost w. r. t. to an associated cost function.

There are many applications of the rich theory of stochastic comparison and stochastic dominance, see e. g., [SS94, MS02]. However, there are only few papers applying them to analyze or develop algorithms. The papers by Mitzenmacher [BM01, Mit96] employ these methods to analyze routing algorithms.

4.3. Stochastic dominance and Markov chain models for online algorithms

From now on we consider random input sequences drawn according to some probability distribution and aim to analyze the performance of an online algorithm probabilistically. This approach dates back at least to the 1970s (see e. g., [CD73] and the references therein) and has been applied to various online optimization problems. Our contribution is to introduce stochastic orders, in particular the stochastic dominance order, to the performance analysis of online algorithms.

4.3.1. Stochastic dominance and online algorithms

So far all probabilistic analyses of online algorithms are more or less average case analyses, i. e., they focus on the *expected* performance. Sometimes results on the expected performance are used to obtain "with high probability" or "asymptotically almost surely" results. However, different algorithms are compared by considering their expected performance, e. g., their expected competitive ratio. In this thesis we propose to compare algorithms by looking at their *objective value distributions* using *stochastic orders*, in particular the stochastic dominance order. A random variable X is said to be *stochastically dominated* by a random variable Y, written $X \leq_{\mathrm{st}} Y$, if

$$\mathrm{Prob}\,[X \geq x] \leq \mathrm{Prob}\,[Y \geq x] \quad \text{for all } x \in \mathbb{R}. \tag{4.3}$$

Suppose we can describe the objective values of two online algorithms A and B after n steps by random variables χ_n^{A} and χ_n^{B}, respectively. We can then say that A is *stochastically better* than B w. r. t. to the considered input distribution if $\chi_n^{\mathsf{A}} \leq_{\mathrm{st}} \chi_n^{\mathsf{B}}$ for all $n \in \mathbb{N}$ (assuming a minimization problem).

Example 4.3.1 Consider the bounded-space bin packing problem with at most k open bins, bin capacity B and item sizes drawn uniformly at random from $\{1, \dots, B\}$. The "performance" χ_n^{A} of a bin packing algorithm A after processing n items is the random variable giving the total number of bins used to pack those items. Algorithm A is considered to be superior to algorithm B if the number of bins used by A is stochastically dominated by the number of bins required by B. ■

Having a stochastic dominance result for two online algorithms has some nice consequences due to the properties of this stochastic order.

Theorem 4.3.2 (see e. g., [MS02]) *Let X and Y be two random variables with $X \leq_{\mathrm{st}} Y$ and $f\colon \mathbb{R} \to \mathbb{R}$ be some nondecreasing function. Then*

- $\mathbb{E}\,[X] \leq \mathbb{E}\,[Y]$ *and*
- $f(X) \leq_{\mathrm{st}} f(Y)$.

Suppose $(X_n)_{n \in \mathbb{N}_0}$ and $(Y_n)_{n \in \mathbb{N}_0}$ are sequences of random variables that converge in distribution to X and Y, respectively, and that satisfy $X_n \leq_{\mathrm{st}} Y_n$. Then

- $X \leq_{\mathrm{st}} Y$.

Assuming that the random variables X and Y measure the performance of algorithms A and B, respectively, the first part of Theorem 4.3.2 states that if algorithm A is stochastically better than B it is also better in expectation. It also implies that A has a better expected competitive ratio than B, where the *expected competitive ratio* of an online algorithm ALG is defined as the smallest c such that

$$\mathbb{E}\left[\mathsf{ALG}\right] \leq c \cdot \mathbb{E}\left[\mathsf{OPT}\right]. \tag{4.4}$$

The second part of the theorem states that A remains superior to B for all other performance measures that are nondecreasing functions of the original performance measure. Finally, the third part says that if A is stochastically better than B for any point in time, then it is also better in the limit.

Stochastic dominance results admit an interesting interpretation if the input sequences are uniformly distributed. It is then possible to view the probabilistic result deterministically as a counting result. Stochastic dominance then implies that the better algorithm achieves a low cost on more sequences than the worse one, i. e., it is more robust w. r. t. the unknown future, a property that is certainly desirable for an online algorithm. In fact, stochastic dominance in the special case of the uniform distribution is equivalent to a recent deterministic way to compare online algorithms, known as *Bijective Analysis* [ADLO07, AS09]. Let S_n denote the input sequences of length n and consider two online algorithms ALG_1 and ALG_2. ALG_1 is said to dominate ALG_2 w. r. t. Bijective Analysis, if there is a bijective mapping $\phi\colon S_n \to S_n$ such that $\mathsf{ALG}_1(\sigma) \leq \mathsf{ALG}_2(\phi(\sigma))$ for any $\sigma \in S_n$. Intuitively, the input sequences are "remapped" such that ALG_1 is "pointwise" better than ALG_2. A similar statement is indeed a characterization of the stochastic dominance order, see Theorem 5.2.12 and Remark 5.2.13.

It is quite natural to model the working of an online algorithm on random input as a (time homogeneous, discrete time) Markov chain, where the state space is given by the configurations an online algorithm generates while processing the input sequence. Formally, we will consider the following Markov chain model for online algorithms for minimization problems.

Definition 4.3.3 A *valued Markov chain* (X, χ) is a Markov chain $X = (X_n)_{n\in\mathbb{N}_0}$ on state space $\mathcal{S}$ together with a valuation function $\chi\colon \mathcal{S} \to V$ for some $V \subseteq \mathbb{N}_0$. The random successor state of state $x \in \mathcal{S}$ is denoted by $X(x)$.

Example 4.3.4 Consider again the bounded-space bin packing problem as above. The working of many online bin packing algorithms can be modelled as follows as a valued Markov chain. The state space $\mathcal{S}$ is given by

$$\mathcal{S} = \{(r_1, \ldots, r_k, n) \mid 0 \leq r_i \leq B - 1, n \in \mathbb{N}_0\},$$

where r_i denotes the remaining capacity of bin i and n is the total number of bins used so far. Let A be an online bin packing algorithm that decides in which bin to put the new item or which bin to close by using the information of the remaining capacities and the size of the new item only. Such an algorithm is fully described by the function $\mathsf{A}\colon \mathcal{S} \times \{1, \ldots, B\} \to \{1, \ldots, k\}$ giving for each state s and item size l the bin $\mathsf{A}(s, l)$ in which the new item is put. Note that if the capacity of that bin is not sufficient, it is closed and replaced by a new empty bin, which then accomodates the new item. The successor state of $s = (r_1, \dots, r_k, n)$ for a new item of size l is then $s' = (r'_1, \dots, r'_k, n')$ with

$$r'_i = \begin{cases} r_i - l & i = \mathsf{A}(s,l), l \le r_i, \\ B - l & i = \mathsf{A}(s,l), l > r_i, \\ r_i & \text{otherwise}, \end{cases}$$

$$n' = \begin{cases} n + 1 & i = \mathsf{A}(s,l), l > r_i, \\ n & \text{otherwise}. \end{cases}$$

These successor relations together with the uniform item size distribution define the transitions of a Markov chain X^{A} on state space $\mathcal{S}$ with initial state $(0, \dots, 0, 0)$. By defining $\chi\big((r_1, \dots, r_k, n)\big) = n$ we get a valued Markov chain (X^{A}, χ) corresponding to A. ■

In our setting there are basically two ways to measure the performance of an online algorithm. The first, more natural one, was already introduced: We look at the evolution of the valuation over time, i. e., at $\chi(X_n)$ if (X, χ) is a valued Markov chain modelling the algorithm. But we may also ask: How fast does the valuation grow, i. e., for how many steps can the algorithm "guarantee" that the valuation is at most $v \in V$? Of course, an algorithm is regarded to be good if it keeps a low valuation for a long time. Formally, we look at the stopping time T^v_X, a random variable that gives the first time the Markov chain X reaches a state with valuation at least v. The following result states that both performance measures are actually equivalent if we compare algorithms with the stochastic dominance order. The theorem is an extension of a result in [bMBFP06].

Theorem 4.3.5 *Let (X, ϕ) and (Y, ψ) be valued Markov chains on state spaces $\mathcal{S}_X$ and $\mathcal{S}_Y$ with common valuation space $V \subseteq \mathbb{N}_0$. Assume that the transitions of X and Y are such that the value of a state is nondecreasing in each step and that $\phi(X_0) = \psi(Y_0)$. Then the following are equivalent:*

1. $T^v_Y \le_{\text{st}} T^v_X \quad \forall v \in V$.
2. $\phi(X_n) \le_{\text{st}} \psi(Y_n) \quad \forall n \in \mathbb{N}_0$.

Proof. Let the Markov chain X be defined on the probability space $(\Omega, \mathcal{A}, \text{Prob})$. The stopping time T^v_X is then a random variable $T^v_X \colon \Omega \to \mathbb{N}_0$ that is defined by

$$T^v_X(\omega) := \min \big\{ n \,\big|\, \phi\big(X_n(\omega)\big) \ge v \big\}$$

for each $\omega \in \Omega$. Since $\phi(X_n(\omega)) \ge \phi(X_{n'}(\omega))$ whenever $n' \le n$, we have the equivalence

$$T^v_X(\omega) \le n \iff \phi(X_n(\omega)) \ge v,$$

which implies

$$\text{Prob}\,[T^v_X \le n] = \text{Prob}\,[\phi(X_n) \ge v]\,.$$

Of course, analogous statements hold for Y as well.

We now have the following chain of equivalences.

$$\begin{aligned} & \phi(X_n) \leq_{\mathrm{st}} \psi(Y_n) \quad \forall n \in \mathbb{N}_0 \\ \iff & \operatorname{Prob}[\phi(X_n) \geq v] \leq \operatorname{Prob}[\psi(Y_n) \geq v] \quad \forall n \in \mathbb{N}_0, v \in V \\ \iff & \operatorname{Prob}[T_X^v \leq n] \leq \operatorname{Prob}[T_Y^v \leq n] \quad \forall n \in \mathbb{N}_0, v \in V \\ \iff & T_Y^v \leq_{\mathrm{st}} T_X^v \quad \forall v \in V. \end{aligned}$$

□

4.3.2. Simulation results for bounded-space bin packing and bin coloring

We already mentioned that having a stochastic dominance relation is equivalent to having an “almost pointwise” dominance relation (see Theorem 5.2.12 and Remark 5.2.13 on page 127 for a precise statement). Therefore, it is a rather strong statement. It is thus reasonable to ask whether such strong relations hold between interesting algorithms. To support this, we present some Markov chain simulation results for bounded-space bin packing and bin coloring algorithms. *Markov chain simulation* means here that we start with some initial distribution p_0 and then successively compute the distribution p_n of the Markov chain after n steps using its transition matrix. Markov chain simulation is stronger than simulating an online algorithm on a random sample of input sequences, since simulating a Markov chain for n steps implicitly considers *all* sequences of length n. To ensure that stochastic dominance does not show up or vanish due to numerical inaccuracies, the Markov chain simulations described in the following were done with exact rational arithmetic provided by the GNU Multiple Precision Arithmetic Library [GMP06].

Bounded-space bin packing In order to be able to simulate the Markov chains corresponding to bin packing algorithms we assumed that item sizes are discrete, which lead to the countable state space Markov chains described in Example 4.3.4. We remark that historically, the bin packing problem has been studied for continuous item sizes (see [CGJ97] for a survey) and discrete item sizes were only considered much later. In our simulations, we assume that the item sizes are drawn from the uniform distribution on $\{1, \dots, u\}$ for some $u \leq B$. It turns out that the number of bins used by BBF_k is stochastically dominated by that needed by NF_k, at least for the parameters considered here, meaning that BBF_k performs better than NF_k. Detailed simulation results including the expected number of bins used can be found in Table A.12 in the Appendix.

Observation 4.3.6 Let $\mathrm{BBF}_n(k, B, u)$ and $\mathrm{NF}_n(k, B, u)$ denote the state of the BBF_k and NF_k Markov chains for parameters k, B, u after n steps, respectively. For the bin packing parameters in the set

$$\{(k, B, u) \mid 2 \leq k \leq 5, 5 \leq B \leq 14, 4 \leq u \leq B - 1\}$$

we have that

$$\chi(\mathrm{BBF}_n(k, B, u)) \leq_{\mathrm{st}} \chi(\mathrm{NF}_n(k, B, u))$$

for all $n = 1, \dots, 200$.

Ramanan [Ram89] introduced the SmartNextFit algorithm, which uses only one bin. However it employs a more clever closing rule than NextFit: It tries to put the new item in a bin of its own and closes the original bin only if it has less remaining capacity than the bin with the new item. Ramanan shows that SmartNextFit is at least as good as NextFit on *every* input sequence, which is a (stronger) special case of stochastic dominance. Note that the idea of the "safe" closing rule can be applied to BBF_k and NF_k as well. By a straightforward extension of Ramanan's induction argument one can see that the resulting variants are better than the original algorithm on any input sequence, too.

Bin coloring We assume that the color sequence is generated by choosing each color independently and uniformly at random from a set of C colors. An online bin coloring algorithm is called *partitioning algorithm*, if it never puts the same color in two different bins at the same time, i. e., if the color of the new item is already present in one of the bins, the new item is put in this bin. Note that both OneBin and GreedyFit are partitioning algorithms. Due to the uniform color distribution it does not matter exactly which colors are already in a bin. Therefore the operation of any partitioning algorithm ALG can be described on a state space which encodes for every bin i its current number of items f_i and the number of colors in that bin c_i. The state also keeps track of the maximum colorfulness attained so far. Moreover, let $\chi_{\max}$ be the maximum colorfulness that ALG attains on any color sequence. Formally, we have

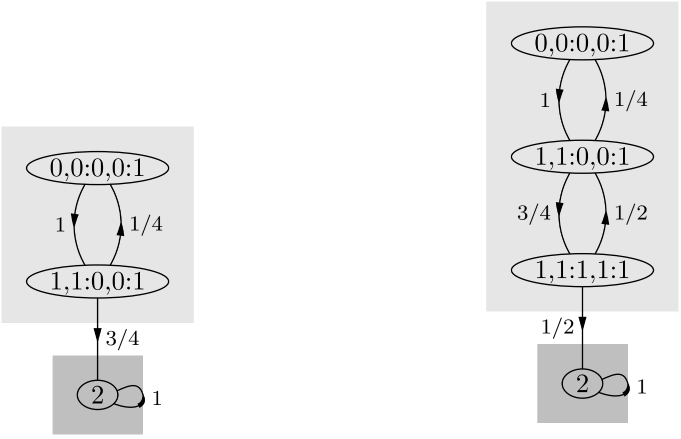
$$\mathcal{S} := \left\{ (f_1, c_1, \dots, f_m, c_m, \chi) \;\middle|\; 0 \le c_i \le f_i \le B, \; c_i \le \chi, 1 \le \chi < \chi_{\max} \right\} \cup \{(\chi_{\max})\}.$$

For convenience, we assume that $\chi \ge 1$ and we collapsed all states with $\chi = \chi_{\max}$ to a single state since we are only interested in the correct distribution of the maximum colorfulness. We use the notation $f_i(s)$, $c_i(s)$, and $\chi(s)$ to refer to the components of state s, the state $(0, 0, \dots, 0, 0, 1)$ is called the *initial empty state*. Note that the states reachable by the operation of an algorithm may be a proper subset of $\mathcal{S}$. In order to reduce the size of the state space, we break the symmetry by requiring that $\big(f_i(s), c_i(s)\big)$ is lexicographical not smaller than $\big(f_{i+1}(s), c_{i+1}(s)\big)$.

If a new item arrives, there are two cases: Either the color is already contained in bin i, i. e., the color is *known (in bin i)*, or it is not, so the color is *new (in bin i)*. In the first case, only f_i is incremented (modulo B), in the second case both f_i and c_i and maybe also χ are incremented. We will denote the successor state of a known-transition in bin i by $s^{\mathrm{known}(i)}$ and for a new-transition in bin i by $s^{\mathrm{new}(i)}$. The OneBin algorithm can then be described by the transitions

$$s' = \begin{cases} s^{\mathrm{known}(1)} & \text{with probability } c_1(s)/C, \\ s^{\mathrm{new}(1)} & \text{with probability } (1 - c_1(s))/C. \end{cases}$$

Similarly, any partitioning algorithm can be modelled. We denote the Markov chains modelling OneBin and GreedyFit for parameters m, B, and C by $\mathsf{OB}(m, B, C)$ and $\mathsf{GF}(m, B, C)$, respectively. Two small examples are given in Figure 4.2.



(a) OneBin Markov chain OB(2, 2, 4).

(b) GreedyFit Markov chain GF(2, 2, 4).

Figure 4.2.: *Small example Markov chains for* OneBin *and* GreedyFit *for the parameters $m = B = 2$ and $C = 4$. The lightly shaded states are the states of maximum colorfulness 1, the darker ones those with maximum colorfulness 2.*

We computed the colorfulness distributions of processing up to 1000 items with OneBin and GreedyFit by simulating the evolution of the corresponding Markov chains for some set of parameters. More precisely, we considered two parameter sets:

1. $C = mB$, m, B s. t. the resulting Markov chains have at most 1000 states, and
2. $m = 3$, $B = 5$, $6 \leq C \leq 45$ and $C \bmod 3 \equiv 0$.

Table 4.1 shows an example of the evolution of the colorfulness distributions of OneBin and GreedyFit, clearly exhibiting stochastic dominance. The simulation results are summarized in the following observation.

Observation 4.3.7 For the parameters given above we have

$$\chi(\mathsf{GF}_n(m, B, C)) \leq_{\text{st}} \chi(\mathsf{OB}_n(m, B, C))$$

for all $n = 1, \ldots, 1000$.

This observation states that w. r. t. stochastic dominance the algorithms are ordered as one would conjecture. Stochastic dominance also explains the superiority of GreedyFit to OneBin observed in earlier simulations on randomly generated sequences. We will provide a formal proof of this result in Section 5.3.

4.4. Optimality of LRU for paging with locality of reference

In this section we apply stochastic dominance analysis to show that in the paging problem, LRU is optimal w. r. t. stochastic dominance for a certain natural class of input sequences.

#items	OneBin	GreedyFit
5	0.000, 0.004, 0.094, 0.525, 1.000	0.094, 1.000, 1.000, 1.000, 1.000
10	0.009, 0.276, 1.000	0.189, 0.981, 1.000, 1.000
20	0.076, 1.000	0.009, 0.725, 0.999, 1.000
40	0.006, 1.000	0.429, 0.999, 1.000
80	1.000	0.149, 0.998, 1.000
160	1.000	0.018, 0.993, 1.000
1000	1.000	0.956, 1.000

Table 4.1.: *Example of the evolution of the colorfulness distribution of* OneBin *and* GreedyFit *for $m = 3$, $B = 5$, and $C = 15$. Shown are the cumulative distribution functions of the colorfulness distributions.*

As byproducts we obtain several results from the literature as corollaries of our optimality results.

4.4.1. Paging with locality of reference

We start by recalling some basic notions for paging algorithms, see e. g., [BEY98]. A standard tool is the partitioning of a sequence into *phases*. The first phase starts with the first request. Phase ℓ starts with the $(k+1)$st *distinct* request after the start of phase $\ell - 1$. Each phase ends just before the start of the next phase or at the end of the sequence, whichever comes first.

Given a request sequence σ, we say that a page p is *marked w. r. t. σ* if it has been requested in the final phase of σ; otherwise, we say that p is *unmarked w. r. t. σ*. Note that by definition of the phases, there cannot be more than k marked pages w. r. t. a request sequence. Also note that all pages are unmarked w. r. t. the empty sequence. Moreover, the partition into phases and the set of marked pages at any point in the sequence do not depend on the algorithm. An algorithm is called *marking algorithm* if, for any request sequence σ, it never evicts a marked page. Observe that a marking algorithm has, at any point in time, all marked pages in its cache, which justifies the name.

A paging algorithm is called *lazy* if it evicts a page only on a page fault and never evicts more than one page. We remark that both FIFO and LRU are lazy algorithms, whereas LRU is a marking algorithm and FIFO is not.

One of the reasons that competitive analysis is not able to make a distinction between the performance of several paging algorithms is that it considers arbitrary request sequences. In practice, however, request sequences have some structure, for instance they often feature *locality of reference*. In a strict sense this means that if a page is referenced, it is likely to be referenced again in the near future, a fact that has been observed early and formalized by Denning with his working set concept [Den68, Den80].

There is also the broader sense of locality of reference, meaning that usually "each time a page is referenced by a program, the next page to be referenced is very likely to come from a small set of pages" [BIRS95]. To formalize this, Borodin et al. [BIRS95] presented the *access graph model*, in which a graph models which pages can be requested after a certain page has been asked. Using this model, they showed that LRU is at least as good as FIFO. This approach was extended in several ways in [IKP96, KPR00, FM97]. Panagiotou and Souza [PS06] introduce another model, restricting the sequences such that successive references to the same page are mostly close together or far apart.

We consider only locality of reference in the strict sense and focus on three models for paging with locality of reference proposed in the literature. Our main result states that LRU is optimal w. r. t. stochastic dominance for sequences exhibiting this kind of locality of reference. It is well known that LRU is not an optimal online algorithms for sequences with locality of reference in the broader sense, since e. g., the access graph model allows the $k + 1$ cycle as an access graph, for which LRU faults on every page request.

The age model Coffman and Denning [CD73] introduced the following probabilistic model which we call the *age model*. In the age model, the next request for a prefix sequence σ is generated based on the age of the pages. For a prefix sequence σ, the age of a page $p \in M$ is defined by

$$\operatorname{age}(p, \sigma) := \begin{cases} l & \text{if } p \text{ is the } l\text{th most recently requested page,} \\ \infty & \text{if } p \text{ does not appear in } \sigma. \end{cases}$$

We say that a probability distribution over the request sequences is an *age model distribution* if it arises in the following way. Let Δ be the set of probability distributions over $\{1, \dots, |M|\}$. Given a prefix sequence σ, the probability $\text{Prob}\,[p \mid \sigma]$ that the next request page is p is determined by an *age distribution* $\delta \in \Delta$ as follows. The age distribution δ gives the age of the new request page p, i. e., if a is a realization according to δ, the next page is $p \in M$ with $\operatorname{age}(p, \sigma) = a$. If there is no page with age a one of the pages with age ∞ is chosen arbitrarily.

Let $\mathcal{D} \subseteq \Delta$ be the set of probability distributions with nonincreasing distribution functions $f\colon \{1, \dots, |M|\} \to [0, 1]$. Note that considering age distributions from $\mathcal{D}$ models locality of reference: Pages requested more recently have a high probability to be requested next. Note that $\mathcal{D}$ contains two out of three classes of age distributions studied by Becchetti [Bec04]. In his diffuse adversary analysis, Becchetti considers probability distributions favoring recently over less recently requested pages and shows that in this model, LRU outperforms FWF.

The concave function model and the a-locality model In contrast to the age model, the concave function model [AFG05] and the a-locality model [Tor98] are deterministic models which restrict the set of request sequences. Both models are based on Denning's working set concept [Den68, Den80].

Albers et al. [AFG05] propose the *concave function model* which models working sets. Locality of reference is modeled by an increasing concave function $f\colon \mathbb{N} \to \mathbb{R}$, which specifies the maximum number $f(l)$ of distinct pages in a (contiguous) subsequence of length l for any $l \in \mathbb{N}$. A request sequence for which each subsequence of length l has at most $f(l)$ distinct pages is called *f-consistent*. They showed that LRU is an optimal online algorithm in their model and that FIFO and marking algorithms are not optimal in general.

Torng [Tor98] generalizes the partitioning of a sequence to *m-phases*, i. e., a phase consists of m distinct pages. For a sequence σ, we denote the subsequence $\sigma_i, \ldots, \sigma_j$ by $\sigma[i, j]$. Given a sequence σ of length n, define the phase indices $I_{i,m}(\sigma)$ recursively by

$$\begin{aligned} I_{0,m}(\sigma) &= 0, \\ I_{i+1,m}(\sigma) &= \max\{j \leq n \mid \sigma[I_{i,m}(\sigma), j] \text{ contains at most } m \text{ distinct pages}\}. \end{aligned}$$

The m-phases of σ are then given by $\sigma[I_{0,m}(\sigma)+1, I_1(\sigma)], \sigma[I_{1,m}(\sigma)+1, I_{2,m}(\sigma)], \ldots$ and so on. Let $A(\sigma, m)$ be the average length of the m-phases in σ. Torng argues that σ exhibits significant locality of reference if $A(\sigma, m) \gg m$. To capture this formally, we introduce the notion of *a-locality* for a function $a\colon \mathbb{N} \to \mathbb{R} \in \Omega(1)$. A sequence σ is called *a-local* if $A(\sigma, m) \geq a(m)m$ holds for all $m = 1, \ldots, |\sigma|$. Note that this generalizes the notion of "a-local" used in [DLO08]. Torng [Tor98] showed that, among other algorithms, LRU achieves a constant competitive ratio for sequences σ with $A(\sigma, k) \gg k$.

4.4.2. Optimality of LRU for paging with locality of reference

We now show that LRU is optimal w. r. t. to stochastic dominance, i. e., incurs stochastically fewer page faults than any other paging algorithm, for a certain class of request sequence distributions. This class includes all distributions according to the age model as well as the uniform distribution over all f-consistent sequences for any increasing concave function f and the uniform distribution over all a-local sequences for any function $a\colon \mathbb{N} \to \mathbb{R} \in \Omega(1)$.

Preliminaries

For an online algorithm ALG, an integer $j \in \{1, \ldots, n\}$ and a sequence σ of length $|\sigma| = j - 1$, the random variable $X_j^{\mathsf{ALG}}(\sigma) = 1$ if the first $j-1$ requests of the random sequence are given by σ and ALG encounters a page fault on the jth request; otherwise $X_j^{\mathsf{ALG}}(\sigma) = 0$. The random variable $Y_j(\sigma) = 1$ if $|\sigma| = j$ and the first j requests are as in σ; otherwise $Y_j(\sigma) = 0$. Moreover, we define the random variable $W^{\mathsf{ALG}}(\sigma) = \big(X_j^{\mathsf{ALG}}(\sigma) \mid Y_{j-1}(\sigma) = 1\big)$, i. e., $W^{\mathsf{ALG}}(\sigma) = 1$ if the next request after processing $j-1$ requests from σ leads to a page fault in ALG, and $W^{\mathsf{ALG}}(\sigma) = 0$ otherwise.

Given a sequence σ of length $|\sigma| \geq j$ and an online algorithm ALG, the variable $Z_j^{\mathsf{ALG}}(\sigma) = 1$ if the jth request leads to a page fault when ALG operates on σ and $Z_j^{\mathsf{ALG}}(\sigma) = 0$ otherwise. Note that $Z_j^{\mathsf{ALG}}(\sigma)$ is deterministically determined by ALG, j, and σ. On a sequence $\sigma \in M^n$, an online algorithm ALG has $\sum_{j=1}^{n} Z_j^{\mathsf{ALG}}(\sigma)$ page faults. In analogy to the notation ALG(σ) for the number of page faults of ALG on σ, we denote by ALG the random variable giving the number of page faults on a random sequence. We can write this random variable for a sequence of length n as

$$\mathsf{ALG} = \sum_{\sigma \in M^n} \sum_{j=1}^{n} Z_j^{\mathsf{ALG}}(\sigma) Y_n(\sigma).$$

Finally, we denote by $C^{\mathsf{ALG}}(\sigma)$ the set of pages in the cache after the sequence σ has been processed by algorithm ALG.

Lemma 4.4.1 *The random variable* ALG *can be expressed as*

$$\mathsf{ALG} = \sum_{j=1}^{n} \sum_{\sigma \in M^{j-1}} X_j^{\mathsf{ALG}}(\sigma).$$

Proof. We have

$$\begin{aligned}\mathsf{ALG} &= \sum_{\sigma \in M^n} \sum_{j=1}^{n} Z_j^{\mathsf{ALG}}(\sigma) Y_n(\sigma) = \sum_{j=1}^{n} \sum_{\sigma \in M^n} Z_j^{\mathsf{ALG}}(\sigma) Y_n(\sigma) \\ &= \sum_{j=1}^{n} \sum_{\sigma_1 \in M^{j-1}} \sum_{p \in M} \sum_{\sigma_2 \in M^{n-j}} Z_j^{\mathsf{ALG}}(\sigma_1 p \sigma_2) Y_n(\sigma_1 p \sigma_2).\end{aligned}$$

As ALG is an online algorithm, we can write this as

$$\mathsf{ALG} = \sum_{j=1}^{n} \sum_{\sigma_1 \in M^{j-1}} \sum_{p \in M} Z_j^{\mathsf{ALG}}(\sigma_1 p) \sum_{\sigma_2 \in M^{n-j}} Y_n(\sigma_1 p \sigma_2).$$

Due to the fact that $Y_j(\sigma_1 p) = \sum_{\sigma_2 \in M^{n-j}} Y_n(\sigma_1 p \sigma_2)$, we get

$$\begin{aligned}\mathsf{ALG} &= \sum_{j=1}^{n} \sum_{\sigma_1 \in M^{j-1}} \sum_{p \in M} Z_j^{\mathsf{ALG}}(\sigma_1 p) Y_j(\sigma_1 p) \\ &= \sum_{j=1}^{n} \sum_{\sigma_1 \in M^{j-1}} X_j^{\mathsf{ALG}}(\sigma_1),\end{aligned}$$

where the last equality follows from the fact that $X_j^{\mathsf{ALG}}(\sigma_1) = 1$ for all realizations of the request sequence that start with σ_1 and the jth request p leads to a page fault, i. e., $Z_j^{\mathsf{ALG}}(\sigma_1 p) = 1$. □

The following theorem is the main result underlying our stochastic dominance proofs. We call a probability distribution over request sequences a *prefix distribution* if it is completely described by the probability that page p is requested given that the sequence up to this page is σ, $\mathrm{Prob}\,[p \mid \sigma]$. Note that all age model distributions are of this type.

Theorem 4.4.2 *Let* ALG_1 *and* ALG_2 *be two online paging algorithms. Assume that the request sequence is drawn according to a prefix distribution and denote by* P *the random next request after prefix* σ*. Suppose that*

$$\mathrm{Prob}\left[P \in C^{\mathsf{ALG}_1}(\sigma)\right] \geq \mathrm{Prob}\left[P \in C^{\mathsf{ALG}_2}(\sigma)\right] \tag{4.5}$$

for any prefix sequence σ*. Then* $\mathsf{ALG}_1 \leq_{\mathrm{st}} \mathsf{ALG}_2$.

Proof. By Lemma 4.4.1, it is sufficient to show

$$X_j^{\mathsf{ALG}_1}(\sigma) \leq_{\text{st}} X_j^{\mathsf{ALG}_2}(\sigma),$$

for all $j = 1, \dots, n$, and all sequences $\sigma \in M^{j-1}$. As the variables $X_j^{\mathsf{ALG}}(\sigma)$ are binary random variables, this is equivalent to

$$\text{Prob}\left[X_j^{\mathsf{ALG}_1}(\sigma) = 1\right] \leq \text{Prob}\left[X_j^{\mathsf{ALG}_2}(\sigma) = 1\right]. \tag{4.6}$$

For any online algorithm ALG, $j \in \{1, \dots, n\}$, and $\sigma \in M^{j-1}$, we can write

$$\begin{aligned}\text{Prob}\left[X_j^{\mathsf{ALG}}(\sigma) = 1\right] &= \text{Prob}\left[Y_{j-1}(\sigma) = 1 \wedge X_j^{\mathsf{ALG}}(\sigma) = 1\right]\\ &= \text{Prob}\left[X_j^{\mathsf{ALG}}(\sigma) = 1 \mid Y_{j-1}(\sigma) = 1\right] \cdot \text{Prob}\left[Y_{j-1}(\sigma) = 1\right]\\ &= \text{Prob}\left[W^{\mathsf{ALG}}(\sigma) = 1\right] \cdot \text{Prob}\left[Y_{j-1}(\sigma) = 1\right].\end{aligned}$$

Since $Y_{j-1}(\sigma)$ does not depend on the algorithm, (4.6) is equivalent to

$$\text{Prob}\left[W^{\mathsf{ALG}_1}(\sigma) = 1\right] \leq \text{Prob}\left[W^{\mathsf{ALG}_2}(\sigma) = 1\right],$$

which in turn is equivalent to (4.5). □

Optimality results

Theorem 4.4.3 *Suppose the request sequence is chosen according to the age model with age distributions from $\mathcal{D}$. Then the number of page faults of* LRU *is stochastically dominated by that of any online algorithm.*

Proof. Let ALG be any online paging algorithm. We show that condition (4.5) of Theorem 4.4.2 is satisfied. Note that always $|C^{\mathsf{LRU}}(\sigma)| \geq |C^{\mathsf{ALG}}(\sigma)|$ holds. Therefore by definition of LRU, there is an injective mapping $\phi\colon C^{\mathsf{ALG}}(\sigma) \to C^{\mathsf{LRU}}(\sigma)$ that maps a page q from $C^{\mathsf{ALG}}(\sigma)$ to a page p in $C^{\mathsf{LRU}}(\sigma)$ with $\text{age}(p, \sigma) \leq \text{age}(q, \sigma)$. Let p be some page that ALG has in the cache and denote by P the random next request generated according to the age model distribution. Clearly, $\text{Prob}\left[P = \phi(p)\right] \geq \text{Prob}\left[P = p\right]$, which implies condition (4.5). □

Remark 4.4.4

1. Coffman and Denning [CD73, p. 276] show that LRU achieves an optimal expected number of page faults if the request sequence is generated as in Theorem 4.4.3. This result is implied by Theorem 4.4.3.
2. The result actually holds for all age distributions where the age probabilities $(p_1, \dots, p_{|M|})$ satisfy $\min\{p_1, \dots, p_k\} \geq \max\{p_{k+1}, \dots, p_{|M|}\}$. As the uniform distribution over all sequences belongs to the family of age model distributions, we also have that LRU, as any other lazy paging algorithm, is an optimal algorithm w. r. t. Bijective Analysis for the set of all request sequences.

We will now prove similar optimality results for the deterministic locality of reference models mentioned in Section 4.4.1, assuming the uniform distribution over the feasible sequences. We start by explaining how we can use a prefix distribution to generate a uniform distribution. Denote by S_n the set of sequences of length n that are feasible for the locality of reference model (i. e., f-consistent or a-local). Moreover, for any σ, $0 \le |\sigma| < n$, and $p \in M$ let $S_n(\sigma, p)$ be the set of extensions of σp to a feasible sequence of length n, i. e.,

$$S_n(\sigma, p) = \left\{ \sigma' \in M^{n-|\sigma|} \;\middle|\; \sigma p \sigma' \in S_n \right\}.$$

Finally define $L_n(\sigma, p) = |S_n(\sigma, p)|$ and $L_n(\sigma) = \sum_{p \in M} L_n(\sigma, p)$. A uniformly distributed random sequence $\Sigma \in S_n$ may be generated as follows: after $\sigma = (\sigma_1, \ldots, \sigma_{i-1})$ has been choosen, we set $\sigma_i = p$ with probability $L_n(\sigma, p)/L_n(\sigma)$ for each $p \in M$.

Lemma 4.4.5 *Consider any online paging algorithm* ALG *and suppose that for any sequence* σ, $0 \le |\sigma| < n$, *and for any pages* $p \in C^{\mathsf{LRU}}(\sigma) \setminus C^{\mathsf{ALG}}(\sigma) =: M_p$ *and* $q \in C^{\mathsf{ALG}}(\sigma) \setminus C^{\mathsf{LRU}}(\sigma) =: M_q$ *we have*

$$L_n(\sigma, p) \ge L_n(\sigma, q). \tag{4.7}$$

Then $\mathsf{LRU}(\Sigma) \le_{\mathrm{st}} \mathsf{ALG}(\Sigma)$, *where* Σ *is a random sequence distributed uniformly over* S_n.

Proof. For a sequence σ with $|\sigma| = j < n$ let P denote the random request that follows σ according to the distribution defined above, i. e., $\mathrm{Prob}\,[P = p] = L_n(\sigma, p)/L_n(\sigma)$. By Theorem 4.4.2, it is sufficient to show $\mathrm{Prob}\left[P \in C^{\mathsf{LRU}}(\sigma)\right] \ge \mathrm{Prob}\left[P \in C^{\mathsf{ALG}}(\sigma)\right]$ for any sequence σ. We have

$$\begin{aligned}
\mathrm{Prob}\left[P \in C^{\mathsf{LRU}}(\sigma)\right] &= \frac{\sum_{p \in C^{\mathsf{LRU}}(\sigma)} L_n(\sigma, p)}{L_n(\sigma)} \\
&= \frac{\sum_{p \in C^{\mathsf{LRU}}(\sigma) \cap C^{\mathsf{ALG}}(\sigma)} L_n(\sigma, p) + \sum_{p \in M_p} L_n(\sigma, p)}{L_n(\sigma)} \\
&\ge \frac{\sum_{p \in C^{\mathsf{LRU}}(\sigma) \cap C^{\mathsf{ALG}}(\sigma)} L_n(\sigma, p) + \sum_{q \in M_q} L_n(\sigma, q)}{L_n(\sigma)} \\
&= \mathrm{Prob}\left[P \in C^{\mathsf{ALG}}(\sigma)\right],
\end{aligned}$$

where we used (4.7) and the fact that $|C^{\mathsf{LRU}}(\sigma)| \ge |C^{\mathsf{ALG}}(\sigma)|$ which implies $|M_p| \ge |M_q|$. □

In the subsequent proofs we need the following notation borrowed from [AS09]. For a sequence σ and two distinctive pages $p, q \in M$, the *complement sequence of* σ *w. r. t.* p *and* q, denoted by $\overline{\sigma}^{(p,q)}$, arises from σ by exchanging each occurence of p by q and vice versa and keeping all other requests. Note that the mapping $\sigma \mapsto \overline{\sigma}^{(p,q)}$ is bijective since it is self-inverse.

Theorem 4.4.6 *Let* $f \colon \mathbb{N} \to \mathbb{R}$ *be an increasing concave function and let* Σ *be a request sequence drawn uniformly at random from all* f*-consistent sequences of length* n. *Then* $\mathsf{LRU}(\Sigma) \le_{\mathrm{st}} \mathsf{ALG}(\Sigma)$.

Proof. We apply Lemma 4.4.5 and show $L_n(\sigma, p) \geq L_n(\sigma, q)$ for any sequence σ, $0 \leq |\sigma| < n$ and for any pages $p \in M_p$ and $q \in M_q$ by giving an injective map from $S_n(\sigma, q)$ to $S_n(\sigma, p)$. In particular, we claim that for $\sigma' \in S_n(\sigma, q)$, $\sigma' \mapsto \overline{\sigma'}^{(p,q)}$ is such a map.

By definition of p and q, the last request for page p in σ was after the last request for page q. Let $\omega = \sigma q \sigma'$ be the f-consistent extended sequence corresponding to σ'. Lemma 1 from [AS09] states that either the sequence $\sigma p \overline{\sigma'}^{(p,q)}$ is f-consistent or the first access to p after σ in ω is before the first access to q. Since in ω, q is the first requested page after σ, the second case obviously cannot be true and therefore $\overline{\sigma'}^{(p,q)}$ must be an element of $S_n(\sigma, p)$. ☐

Theorem 4.4.7 *Consider a function $a\colon \mathbb{N} \to \mathbb{R} \in \Omega(1)$ and let Σ be a request sequence drawn uniformly at random from all a-local sequences of length n. Then* $\mathsf{LRU}(\Sigma) \leq_{\mathrm{st}} \mathsf{ALG}(\Sigma)$.

Proof. We use the same proof technique as in the last theorem, using again the mapping $\sigma' \mapsto \overline{\sigma'}^{(p,q)}$ for each $\sigma' \in S_n(\sigma, q)$. Let $\omega := \sigma q \sigma'$ be the original sequence and $\bar{\omega} := \sigma p \overline{\sigma'}^{(p,q)}$ be its mapping image. We establish $A(\bar{\omega}, m) \geq A(\omega, m)$ for any m by showing $I_{i,m}(\bar{\omega}) \geq I_{i,m}(\omega)$ by induction on i. Let ℓ be the number of m-phases of σ. It is obvious that $I_{i,m}(\bar{\omega}) = I_{i,m}(\omega)$ for all $i < \ell$.

Suppose $|\sigma| = l$ and denote by $D(\sigma, i, j)$ the set of distinct pages in the subsequence $\sigma_i, \ldots, \sigma_j$. We claim $I_{\ell,m}(\bar{\omega}) \geq I_{\ell,m}(\omega)$, or equivalently,

$$|D(\bar{\omega}, I_{\ell-1,m}(\sigma) + 1, I_{\ell,m}(\omega))| \leq |D(\omega, I_{\ell-1,m}(\sigma) + 1, I_{\ell,m}(\omega))|. \qquad (4.8)$$

If q is referenced in $\sigma[I_{\ell-1,m} + 1, l]$, then so is p since $p \in C^{\mathsf{LRU}}(\sigma)$ and $q \notin C^{\mathsf{LRU}}(\sigma)$. This implies that (4.8) holds with equality in this case. The same is true if both p and q do not occur in $\sigma[I_{\ell-1,m} + 1, l]$. In the case that p occurs in $\sigma[I_{\ell-1,m} + 1, l]$ but q does not we have $|D(\bar{\omega}, I_{\ell-1,m}(\sigma) + 1, l + 1)| < |D(\omega, I_{\ell-1,m}(\sigma) + 1, l + 1)|$ and (4.8) holds as well.

To finish the induction, assume we already established $I_{i,m}(\bar{\omega}) \geq I_{i,m}(\omega)$. Then $I_{i+1,m}(\bar{\omega}) \geq I_{i+1,m}(\omega)$ follows by observing that

$$|D(\bar{\omega}, I_{i,m}(\bar{\omega}) + 1, I_{i+1,m}(\omega))| \leq |D(\omega, I_{i,m}(\omega) + 1, I_{i+1,m}(\omega))|,$$

which follows from the fact that $|D(\sigma, i, j)| = |D(\overline{\sigma}^{(p,q)}, i, j)|$ for any sequence σ and $1 \leq i < j \leq |\sigma|$. ☐

Remark 4.4.8

1. Theorem 4.4.6 is actually equivalent to Theorem 1 in [AS09], but we believe that our proof is simpler and more straightforward; it is only half as long, too.

2. Theorem 4.4.6 also holds for the more general average concave function model by Albers et al. [AFG05], as Lemma 1 from [AS09] holds for this as well.

3. As observed by Angelopoulos and Schweitzer [AS09], LRU is actually optimal for each locality of reference model with the following property: If the sequence $\sigma_1 q \sigma_2 p \sigma_3 q \sigma_4 p \sigma_5$ with sequence σ_3 containing neither p nor q is feasible, then so is the sequence $\sigma_1 q \sigma_2 p \sigma_3 p \sigma_4 q \sigma_5$, which is arguably more "local" than the original one.

Remark 4.4.9 Aho et al. [ADU71] study random request sequences that are *almost stationary* in the sense that the page request probabilities maintain their relative orders, i. e., the probability distribution is such that for pages p and q

$$\text{Prob}\,[p \mid \sigma] \geq \text{Prob}\,[q \mid \sigma] \quad \Longrightarrow \quad \text{Prob}\,\left[p \mid \sigma\sigma'\right] \geq \text{Prob}\,\left[q \mid \sigma\sigma'\right] \tag{4.9}$$

for all request sequences σ and σ'. They define a *ranking relation* $<$ on the pages with $q < p$ meaning $\text{Prob}\,[q \mid \sigma] \leq \text{Prob}\,[p \mid \sigma]$ for all request sequences σ. Aho et al. show that the algorithm A_0 which on a page fault evicts a $<$-minimal page from the cache is optimal w. r. t. to the expected number of page faults. From (4.9) and Theorem 4.4.2 it is clear that A_0 is optimal w. r. t. to stochastic dominance of page faults, implying the result from [ADU71]. Note that the almost stationary random sequences include the *independent reference model* [CD73], in which each requested page is drawn i. i. d. from a fixed distribution.

4.5. Conclusion

We saw in Sections 4.3.2 and 4.4 that stochastic dominance is useful to compare online algorithms, yielding results that are more in line with their actually observed behavior. The proofs for the optimality of LRU presented in Section 4.4 are relatively straightforward, which is mainly due to the following fact. In the presence of locality of reference, we can prove that LRU's cache configuration after *any sequence* σ has higher probability of *not increasing* the objective function than that of any other algorithm on the same sequence (Theorem 4.4.2), which is sufficient for stochastic dominance.

In general, however, such a condition is obviously not necessary: Algorithm ALG_1 may be in a state with higher probability of increasing the objective function after processing a certain sequence σ of length n than algorithm ALG_2. Nevertheless ALG_1 may still be better than ALG_2 w. r. t. to stochastic dominance for random sequences of length $n + 1$. As an example, consider the bounded-space bin packing problem with $B = 6$ and size sequence $\sigma = (3, 4, 1, 2, 2)$. After packing the item of size 1 BBF_2 has used two bins with remaining capacities 3 and 1 and thus cannot accomodate the two items of size 2 without opening a new bin. In contrast, NF_2 has used two bins which both have remaining capacity 2, meaning it can pack both items without opening a new bin. Nevertheless the simulation results of Section 4.3.2 indicate that $\chi(\text{BBF}_2(\Sigma)) \leq_{\text{st}} \chi(\text{NF}_2(\Sigma))$ for a sequence Σ chosen uniformly at random from $\{1, \ldots, B - 1\}^6$. For such cases more powerful proof techniques are needed. In the next chapter, we develop such techniques and apply them to a detailed analysis of online bin coloring algorithms.

Chapter 5.

Analysis of bin coloring algorithms

In this chapter we conduct a detailed study of online algorithms for the bin coloring problem, applying the stochastic dominance approach developed in Chapter 4. We start by introducing the online bin coloring problem and discussing an application to estimating the handling capacity of an elevator system. Section 5.2 deals with general methods for proving stochastic dominance results and how they can be applied to analyzing bin coloring algorithms. Finally, we present a proof showing that GreedyFit is better than OneBin w. r. t. stochastic dominance in Section 5.3. This result appeared as [HV08].

5.1. Problem definition and an application to elevator control

An instance of the bin coloring problem is described by the number of simultaneously open bins m, the bin capacity B and a sequence of n unit sized items, each of which has one of C colors. The items need to be packed in the open bins and whenever a bin has B items, it is closed and replaced by a new empty bin. The colorfulness of a bin is the number of different colors in this bin and the goal is to pack the items in the (open) bins such that the maximum colorfulness over all bins is minimized. For probabilistic analysis, we assume that the number of open bins m, the bin capacity B and the number of colors C is given deterministically. The color sequence, however, is generated by chosing each color independently according to a probability distribution function γ over the colors.

Note that in this model, all online algorithms eventually have to produce a bin with colorfulness B if the number of colors is sufficiently high, say $C \geq 2mB$. This implies that in this case, all online algorithms are asymptotically equally bad. Moreover, since eventually there will be a color subsequence of length $2mB$ with all colors different, the asymptotic competitive ratio is 1 with probability 1. Both issues indicate that asymptotic probabilistic analysis does not give meaningful results. Instead it is necessary to study the *transient* behavior instead of the asymptotic behavior of the algorithms. Let the random variables χ_n^{GF} and χ_n^{OB} denote the maximum colorfulness attained after processing n items using GreedyFit and OneBin, respectively. We will show that GreedyFit is stochastically better than OneBin after n items, i. e., that $\chi_n^{\mathsf{GF}} \leq_{\mathrm{st}} \chi_n^{\mathsf{OB}}$. The competitive analysis results of Krumke et al. [KdPSR01] imply that there is a sequence on which OneBin gives better overall colorfulness than GreedyFit, i. e., GreedyFit is not sequence-wise at least as good as OneBin.

We consider the following bin coloring algorithms.

GreedyFit packs an item with an already present color in the bin with that color and otherwise chooses a bin which currently has the least number of different colors.

OneBin uses only a single bin, i. e., it packs all items in the same bin until it is full and replaced.

FixedColors puts the items into bins according to their colors and a prescribed color-to-bin-assignment.

Observe that FixedColors is somewhere between OneBin and GreedyFit: it takes advantage of all available bins, but uses a static assignment rule of colors to the bins.

Application to elevator control Before starting with stochastic dominance analysis, we briefly explain an interesting connection between the online bin coloring problem and the relative performance of conventional elevator control and elevator control based on *destination calls* as discussed in Chapter 2.

In Section 2.2 we mentioned that the up peak handling capacity, a performance measure often used for dimensioning elevator systems, is inversely related to RTT, the time needed for a single roundtrip. Elevator engineers estimate RTT using the formula

$$RTT = 2Ht_v + (S+1)t_s + Pt_l,$$

where H is the highest floor reached, t_v the drive time needed to pass one floor, S is the number of stops needed to unload the passengers, t_s the time per stop, and t_l the time to unload one passenger. Note that only H and S depend on the actual load of the lift. For calculations, H is usually assumed to be N or $N-1$, which is justified if the lift capacity P is high. The number of stops S is calculated by assuming some distribution of the destination floors.

We now want to compare the RTT for a conventional system to that of a destination call system using our bin coloring analysis. Let us assume that the passenger arrival rate is so high that the elevators leaving the main entrance floor are always full (this is also assumed in the formula for the handling capacity above). Since we are only interested in the number of stops S for a roundtrip, we can model the passenger sequence by a sequence of destinations only.

In a conventional system, passengers board the lifts in the order of their arrival at the main entrance floor. The stops of the resulting roundtrips are determined by the sub-sequences of size P of the destination floor sequence. Regarding the destination floors as colors and the round trips as bins, this can be viewed as OneBin working on sequences with N distinct colors and bin capacity P.

For a destination call system the elevator control has the possibility to reduce stops by assigning passengers with the same destination floor to the same lift and balancing the number of stops between the L lifts. A natural strategy to do this is the GreedyFit algorithm, using up to L bins.

As we will see in Theorem 5.3.9, the average number of colors achieved by GreedyFit is stochastically dominated by that of OneBin. Applied to our elevator setting this means that the total number of stops in the destination call system is stochastically dominated by that of the conventional system, which implies the same relation for the *RTT* of both systems. Since $X \leq_{\mathrm{st}} Y$ implies $1/X \geq_{\mathrm{st}} 1/Y$, we get that the handling capacity of a destination call system is higher than that of a conventional system, independent of the destination floor distribution. Note that we could not conclude that the expected handling capacity is larger if we had shown only that the expected *RTT* is smaller.

5.2. Comparison methods for Markov chains

We already saw in Sections 4.3.2 and 4.4 that stochastic dominance is useful to compare online algorithms. In particular, Observation 4.3.7 suggests that there is a general stochastic dominance relation between GreedyFit and OneBin. However, ist is not as easy as for the paging algorithm LRU to prove stochastic dominance here. We will therefore consider several general methods for proving stochastic dominance. Results of this type are known as "comparison results" [MS02].

5.2.1. Monotonicity-based Methods

The first comparison result for Markov chains was provided by Daley [Dal68]. It applies to the case that the state space $\mathcal{S}$ is a subset of $\mathbb{N}_0$ and the valuation function χ is the identity, i. e., the value of a state is the number of the state itself. Daley's result is based on the notion of $\leq_{\mathrm{st}}$-monotonicity. A transition matrix P of a Markov chain on state space $\mathcal{S} \subseteq \mathbb{N}_0$ is called $\leq_{\mathrm{st}}$-*monotone*, if

$$s_1, s_2 \in \mathcal{S}, s_1 \leq s_2 \quad \Longrightarrow \quad P(s_1, \cdot) \leq_{\mathrm{st}} P(s_2, \cdot). \tag{5.1}$$

For a probability distribution λ and a transition matrix P on $\mathcal{S}$ we denote by λP the probability distribution resulting from starting with λ and doing a one-step transition according to P (this probability is given by the vector-matrix product). Daley proved that (5.1) is equivalent to requiring that for all distributions λ and μ over $\mathcal{S}$ we have

$$\lambda \leq_{\mathrm{st}} \mu \quad \Longrightarrow \quad \lambda P \leq_{\mathrm{st}} \mu P.$$

Theorem 5.2.1 ([Dal68]) *If the Markov chains X and Y with transition matrices P and Q and state space $\mathcal{S} \subseteq \mathbb{N}_0$ satisfy*

1. $X_0 \leq_{\mathrm{st}} Y_0$,
2. $P(s, \cdot) \leq_{\mathrm{st}} Q(s, \cdot)$ *for all* $s \in \mathcal{S}$,
3. *at least one of the matrices P and Q is $\leq_{\mathrm{st}}$-monotone,*

then this implies $X_n \leq_{\mathrm{st}} Y_n$ for all $n \in \mathbb{N}_0$.

Obviously, this result is not sufficient to compare valued Markov chains with more complex countable state spaces, since in general different states may have the same value. In this case, χ induces a partial order $\leq_\chi$ on the state space $\mathcal{S}$ by

$$s \leq_\chi s' \quad :\Longleftrightarrow \quad \chi(s) \leq \chi(s'). \tag{5.2}$$

Using the theory of *integral stochastic orders* (see e. g., [MS02]) $\leq_{\text{st}}$ can be generalized to partially ordered spaces. We will briefly recapture the notions and results needed to apply monotonicity techniques to valued Markov chains.

Definition 5.2.2 Let $\mathcal{F}$ be a class of functions from $\mathcal{S}$ to $\mathbb{R}$. The class $\mathcal{F}$ induces a stochastic order $\leq_\mathcal{F}$ among $\mathcal{S}$-valued random variables X and Y by

$$X \leq_\mathcal{F} Y \quad :\Longleftrightarrow \quad \mathbb{E}\left[f(X)\right] \leq \mathbb{E}\left[f(Y)\right] \quad \forall f \in \mathcal{F}. \tag{5.3}$$

A stochastic order arising in this way is called an *integral stochastic order* with *generator* $\mathcal{F}$.

Definition 5.2.2 defines a very broad class of stochastic orders that includes the stochastic dominance order on arbitrary partially ordered sets. Consider a partial order $\prec$ on $\mathcal{S}$. A function $f\colon \mathcal{S} \to \mathbb{R}$ is called nondecreasing w. r. t. $\prec$ if it satisfies $f(s) \leq f(s')$ whenever $s \prec s'$ for all $s, s' \in \mathcal{S}$. Then the stochastic dominance order $\leq_{\text{st}}$ induced by $\prec$ is generated via (5.3) by the set of all functions that are nondecreasing w. r. t. $\prec$. The stochastic dominance order $\leq_{\text{st}}$ used so far and defined by inequality (4.3) is just the special case $\mathcal{S} = \mathbb{R}$ and $\prec = \leq$.

It is easier to check (4.3) if the class $\mathcal{F}$ is small and has a simple structure. The stochastic dominance order $\leq_{\text{st}}$ induced by a partial order $\prec$ has also a nice small generator. A subset $S \subseteq \mathcal{S}$ is called a *$\prec$-increasing set* if $s \in S$ and $s \prec s'$ imply $s' \in S$. It is not hard to see that $\leq_{\text{st}}$ induced by $\prec$ on $\mathcal{S}$ is generated by the indicator functions of all $\prec$-increasing subsets of $\mathcal{S}$, which are of course $\prec$-nondecreasing functions.

The theorem of Daley can be generalized to any integral stochastic order by generalizing the equivalent characterization of $\leq_{\text{st}}$-monotonicity. We call a stochastic matrix P over $\mathcal{S}$ *$\leq_\mathcal{F}$-monotone*, if for all distributions λ and μ over $\mathcal{S}$ we have

$$\lambda \leq_\mathcal{F} \mu \quad \Longrightarrow \quad \lambda P \leq_\mathcal{F} \mu P.$$

Theorem 5.2.3 ([MS02, p. 180]) *If the Markov chains X and Y with transition matrices P and Q satisfy*

1. $X_0 \leq_\mathcal{F} Y_0$,
2. $P(s,\cdot) \leq_\mathcal{F} Q(s,\cdot)$ *for all* $s \in \mathcal{S}$,
3. *at least one of the matrices P and Q is $\leq_\mathcal{F}$-monotone,*

then this implies $X_n \leq_\mathcal{F} Y_n$ for all $n \in \mathbb{N}_0$.

Note that the second requirement $P(s,\cdot) \leq_{\mathcal{F}} Q(s,\cdot)$ can be interpreted as "P is statewise better than Q". Applied to online algorithms, we can view monotonicity as an additional property that makes an algorithm superior to all algorithms which are not better in any state. This criterion might thus provide guidance for constructing good online algorithms.

Unfortunately, it turns out that this result is not strong enough to explain e. g., the stochastic dominance observed between GreedyFit and OneBin. In the remainder of this section we will show this.

We say that an online bin coloring algorithm is a *partitioning algorithm*, if it uses at most one bin for the same color at any point in time. Observe that OneBin, FixedColors, and GreedyFit are all partitioning algorithms. We briefly recall the valued Markov chain model from Section 4.3.2 which is suitable for modelling partitioning algorithms, assuming the color sequence is generated by the uniform color distribution. For each bin i of the m bins the state space has a component f_i giving the number of items contained in that bin and a component c_i indicating the number of distinct colors in the bin. Moreover there is a state space component χ that keeps track of the maximum colorfulness attained so far, so the state space is

$$\mathcal{S} := \Big\{ (f_1, c_1, \ldots, f_m, c_m, \chi) \;\Big|\; 0 \leq c_i \leq f_i \leq B, \\ c_i \leq \chi, 1 \leq \chi < \chi_{\max} \Big\}.$$

The key point here is that all partitioning algorithms can be modeled using the *same* state space (some algorithms may not reach all states) and the same valuation function χ, which gives just the χ-component of the state.

Let $\mathsf{GF} = (\mathsf{GF}_n)_{n\in\mathbb{N}_0}$ and $\mathsf{OB} = (\mathsf{OB}_n)_{n\in\mathbb{N}_0}$ be the valued Markov chains modelling the working of GreedyFit and OneBin, respectively, for some fixed parameters $m \geq 2$, B, and number of colors C and denote by P_{OB} and P_{GF} the corresponding transition matrices. Moreover, let $\leq_{\chi-\mathrm{st}}$ be the stochastic dominance order $\leq_{\mathrm{st}}$ induced by the partial order $\leq_\chi$ on $\mathcal{S}$. The following result states that GF and OB satisfy requirements 1 and 2 of Theorem 5.2.3 w. r. t. $\leq_{\chi-\mathrm{st}}$.

Proposition 5.2.4 *Let* GF *and* OB *be valued Markov chains as above.*

1. $\mathsf{GF}_0 \leq_{\chi-\mathrm{st}} \mathsf{OB}_0$.
2. $P_{\mathsf{GF}}(s,\cdot) \leq_{\chi-\mathrm{st}} P_{\mathsf{OB}}(s,\cdot)$ *for every state* $s \in \mathcal{S}$.

Proof. The first assertion follows from the fact that both GF and OB start in the same state, i. e., $\mathsf{GF}_0 = \mathsf{OB}_0$.

To establish the second assertion, consider a state $s \in \mathcal{S}$ and let S^{GF} and S^{OB} be random states distributed according to $P_{\mathsf{GF}}(s,\cdot)$ and $P_{\mathsf{OB}}(s,\cdot)$, respectively. We need to show $\mathbb{E}\left[f(S^{\mathsf{GF}})\right] \leq \mathbb{E}\left[f(S^{\mathsf{OB}})\right]$ for all $\leq_\chi$-nondecreasing functions f. It is sufficient to consider the indicator functions $\mathbb{1}_M$ of all $\leq_\chi$-increasing sets M only. As the valuation χ is increasing and can increase at most by one in each step, we have $\mathbb{E}\left[\mathbb{1}_M(S^{\mathsf{GF}})\right] =$

$\mathbb{E}\left[\mathbb{1}_M(S^{\mathsf{OB}})\right]$ whenever there is a state $s' \in M$ with $\chi(s') \leq \chi(s)$ or all states in M have valuation at least $\chi(s)+2$. We therefore can restrict ourselves to the remaining sets M only. Finally, we can assume that state s is reachable in OB, since otherwise we can modify $P_{\mathsf{OB}}(s,\cdot)$ to be the same as $P_{\mathsf{GF}}(s,\cdot)$ without affecting the observed behavior of the OB Markov chain.

A state s reachable by OB is of the form $s=(f_1,c_1,0,\dots,0,\chi)$. Consider a set $M \subseteq \mathcal{S}$ where the lowest valuation of any state is $\chi+1$. If $f_1=c_1=\chi=0$, we trivially have $\mathbb{E}\left[\mathbb{1}_M(S^{\mathsf{GF}})\right]=\mathbb{E}\left[\mathbb{1}_M(S^{\mathsf{OB}})\right]$ since both algorithms behave identically. In any other case, GF in state s will put an item with a new color in the second bin, thus reaching a state with valuation χ in any case. However, OB will by definition always use the first bin, thus possibly reaching a state in M. Hence, we have

$$\mathbb{E}\left[\mathbb{1}_M(S^{\mathsf{GF}})\right]=\text{Prob}\left[S^{\mathsf{GF}}\in M\right]=0\leq\text{Prob}\left[S^{\mathsf{GF}}\in M\right]=\mathbb{E}\left[\mathbb{1}_M(S^{\mathsf{OB}})\right]$$

and the claim follows. □

All we still need to do in order to prove $\chi(\mathsf{GF}_n)\leq_{\text{st}}\chi(\mathsf{OB}_n)$ by applying Theorem 5.2.3 is to show that P_{OB} or P_{GF} is $\leq_{\chi-\text{st}}$-monotone. However, this is not necessary and we can get along with a weaker result. Instead, it would be sufficient to show that one of P_{OB} and P_{GF} is monotone w. r. t. some integral order $\leq_{\mathcal{F}}$, where $\mathcal{F}$ is a set of indicator functions of $\leq_\chi$-increasing sets including the *level sets* $\mathcal{S}_{\geq v}$ defined by

$$\mathcal{S}_{\geq v}:=\{s\in\mathcal{S}\mid\chi(s)\geq v\}$$

for each valuation v. To see this, suppose $\mathcal{F}$ contains the indicator functions g_v of the level sets $\mathcal{S}_{\geq v}$ and one of the transition matrices is $\leq_{\mathcal{F}}$-monotone. Since requirements 1 and 2 of Theorem 5.2.3 are fulfilled for $\leq_{\chi-\text{st}}$, they are also fulfilled for $\leq_{\mathcal{F}}$, as $\mathcal{F}$ is a set of $\leq_\chi$-nondecreasing functions. We can thus apply Theorem 5.2.3 yielding

$$\begin{aligned}\text{Prob}\left[\chi(\mathsf{GF}_n)\geq v\right]&=\text{Prob}\left[\mathsf{GF}_n\in\mathcal{S}_{\geq v}\right]\\&=\mathbb{E}\left[g_v(\mathsf{GF}_n)\right]\\&\leq\mathbb{E}\left[g_v(\mathsf{OB}_n)\right]&&\text{by Theorem 5.2.3 and Definition 5.2.2}\\&=\text{Prob}\left[\chi(\mathsf{OB}_n)\geq v\right]\end{aligned}$$

for all valuations v, which is equivalent to $\chi(\mathsf{GF}_n)\leq_{\text{st}}\chi(\mathsf{OB}_n)$.

However, we can show that this approach does not work. To do this we need some more machinery.

Definition 5.2.5 The *maximal generator* $\mathcal{R}_{\mathcal{F}}$ of an integral stochastic order $\leq_{\mathcal{F}}$ is the set of all functions for which $X\leq_{\mathcal{F}}Y$ implies $\mathbb{E}\left[f(X)\right]\leq\mathbb{E}\left[f(Y)\right]$.

The maximal generator is useful to characterize $\leq_{\mathcal{F}}$-monotone transition matrices. We will use the structure of the maximal generator to show that monotonicity arguments are not suitable for establishing stochastic dominance between online bin coloring algorithms.

Theorem 5.2.6 ([MS02, p. 181]) *A stochastic matrix P is $\leq_{\mathcal{F}}$-monotone if and only if for every $f \in \mathcal{F}$ the function f^P defined by*

$$f^P(s) := \mathbb{E}\left[f(P(s,\cdot))\right]$$

is an element of the maximal generator $\mathcal{R}_{\mathcal{F}}$.

Theorem 5.2.7 ([MS02, p. 72]) *If $\mathcal{F} \subseteq \mathcal{G} \subseteq \mathcal{R}_{\mathcal{F}}$, and $\mathcal{G}$ is a convex cone containing the constant functions and being closed under pointwise convergence, then $\mathcal{G} = \mathcal{R}_{\mathcal{F}}$.*

Corollary 5.2.8 *Consider an integral stochastic order $\leq_{\mathcal{F}}$ defined by a* finite *class $\mathcal{F}$ of functions from $\mathcal{S}$ to $\mathbb{R}$. The maximal generator of $\leq_{\mathcal{F}}$ is the convex cone generated by $\mathcal{F}$.*

Proof. It is easy to see that any function $f \in \operatorname{cone}\mathcal{F}$ satisfies $\mathbb{E}\left[f(X)\right] \leq \mathbb{E}\left[f(Y)\right]$ whenever $X \leq_{\mathcal{F}} Y$. Hence, $\mathcal{F} \subseteq \operatorname{cone}\mathcal{F} \subseteq \mathcal{R}_{\mathcal{F}}$. Since $\operatorname{cone}\mathcal{F}$ is finitely generated, it is closed under pointwise convergence and the result follows from Theorem 5.2.7. □

The following result shows that the transition matrices of the valued Markov chains for GreedyFit and OneBin are not $\leq_{\mathcal{F}}$-monotone for stochastic orders of the type described above. Thus monotonicity methods cannot be applied (that easily) to show stochastic dominance results between these algorithms.

Theorem 5.2.9 *Let (X, χ) be a valued Markov chain corresponding to* GreedyFit *or* OneBin *operating on a color sequence drawn from the uniform distribution on C colors using m bins with capacity $B \geq 3$. Furthermore, let k be the maximum colorfulness attained by the algorithm. If $k \geq 2$, there is no integral stochastic order $\leq_{\mathcal{F}}$ where $\mathcal{F}$ is a (finite) set of indicator functions of $\leq_{\chi}$-increasing sets including the level sets and the transition matrix P of X is $\leq_{\mathcal{F}}$-monotone.*

Proof. It suffices to consider GreedyFit since OneBin is obtained by running GreedyFit with one bin only.

Let g_k be the indicator functions of the level set $\mathcal{S}_{\geq k}$. Denote by $s_1, \dots, s_l$ all predecessor states of $s_{\max}$. According to the definition of GreedyFit, a predecessor state s_j of $s_{\max}$ is of the form $c_i(s_j) = k - 1 \geq 1$ for any bin i. Therefore, the probability α that the next color leads to state $s_{\max}$ is the same for all predecessor states and it is strictly positive. Hence we have that $g_k^P(s) = \operatorname{Prob}\left[\mathcal{S}_{\geq k} \mid s\right]$ (confer Theorem 5.2.6) is

$$g_k^P(s) = \begin{cases} 0 & s \neq \{s_{\max}, s_1, \dots, s_l\}, \\ \alpha & s = s_j \quad 1 \leq j \leq l, \\ 1 & s = s_{\max}. \end{cases}$$

Since $g_k \in \mathcal{F}$, P is $\leq_{\mathcal{F}}$-monotone, and $\operatorname{cone}\mathcal{F}$ is by Corollary 5.2.8 the maximal generator of $\leq_{\mathcal{F}}$, g_k^P is in $\operatorname{cone}\mathcal{F}$ due to Theorem 5.2.6. Thus for any s_j, $1 \leq j \leq l$, there exists a subset $M(s_j) \ni s_j$ of all $s_{\max}$-predecessor states, such that the indicator function $f_{s_j} := \mathbb{1}_{M(s_j) \cup \{s_{\max}\}}$ of $M(s_j) \cup \{s_{\max}\}$ is in $\mathcal{F}$, as such a subset is needed in any conic combination of indicator functions of $\leq_{\chi}$-increasing sets giving g_k^P.

Now consider the special $s_{\max}$-predecessor state $s = (B-1, k-1, \ldots, B-1, k-1, k-1)$ and its predecessor $s' = (B-1, k-1, \ldots, B-1, k-1, B-2, k-2, k-1)$, in which one color less than in s is known. We can now compute $f_s^P(s) = \operatorname{Prob}\left[M(s) \cup \{s_{\max}\} \mid s\right] = \alpha$, as no state in $M(s)$ can be reached from s. Moreover, $f_s^P(s') = \operatorname{Prob}\left[M(s) \cup \{s_{\max}\} \mid s'\right] = \alpha + 1/C$, since s is the only state in $M(S) \cup \{s_{\max}\}$ that can be reached from s'. We thus derived $f_s^P(s') > f_s^P(s)$, but $s' \leq_\chi s$, so f_s^P is not a $\leq_\chi$-nondecreasing function, thus not in cone $\mathcal{F}$. Therefore, P cannot be $\leq_{\mathcal{F}}$-monotone by Theorem 5.2.6. $\square$

5.2.2. Coupling-based Methods

Doisy [Doi00] proposed the following Markov chain comparison criterion, which does not require monotonicity. Recall that given a valued Markov chain (X, χ) and a state $x \in \mathcal{S}$ we use the notation $X(x)$ for the random successor state of state x.

Theorem 5.2.10 ([Doi00]) *Consider valued Markov chains (X, ϕ) and (Y, ψ) on countable state spaces $\mathcal{S}_X$ and $\mathcal{S}_Y$, respectively. Suppose we have $\phi(X_0) \leq_{\text{st}} \psi(Y_0)$ and*

$$\forall x \in \mathcal{S}_X, y \in \mathcal{S}_Y : \phi(x) \leq \psi(y) \quad \Longrightarrow \quad \phi(X(x)) \leq_{\text{st}} \psi(Y(y)).$$

This implies $\phi(X_n) \leq_{\text{st}} \psi(Y_n)$ for all $n \in \mathbb{N}_0$.

Unfortunately, this criterion is too weak for our purposes, too. To see this, consider the Markov chains of OneBin vs. GreedyFit in the case $m = B = 2$ and $C = 4$ (cf. Figure 4.2 on page 110). For this case, the transition matrices are

$$P_{\mathsf{OB}} = \left(\begin{array}{cc|c} 0 & 1 & 0 \\ 1/4 & 0 & 3/4 \\ \hline 0 & 0 & 1 \end{array}\right), \qquad P_{\mathsf{GF}} = \left(\begin{array}{ccc|c} 0 & 1 & 0 & 0 \\ 1/4 & 0 & 3/4 & 0 \\ 0 & 1/2 & 0 & 1/2 \\ \hline 0 & 0 & 0 & 1 \end{array}\right),$$

where the lines separate the states of colorfulness 1 from the state of colorfulness 2. Let $x = (0, 0, 1)$ (row 1) be the state of the OneBin chain and $y = (1, 1, 1, 1, 1)$ (row 3) be the state of the GreedyFit chain. Since $\chi(x) = \chi(y) = 1$ we have $\chi(x) \leq \chi(y)$. On the other hand, we have $P_{\mathsf{OB}}(x, \cdot) = (0, 1, 0)$ and $P_{\mathsf{GF}}(y, \cdot) = (0, 1/2, 0, 1/2)$. Thus $\chi(\mathsf{OB}(x)) = 1$ with probability 1, whereas $\chi(\mathsf{GF}(y)) = 2$ with probability 1/2, so $\chi(P_{\mathsf{GF}}(x, \cdot)) \not\leq_{\text{st}} \chi(P_{\mathsf{OB}}(y, \cdot))$. Hence, the condition of Theorem 5.2.10 is not fulfilled.

Interestingly, the proof of Doisy's criterion lends itself to a generalization. To describe this, we need the notion of a coupling that is well-known in probability theory (and in some areas of algorithmics as well).

Definition 5.2.11 (Coupling) Let X and Y be $\mathcal{S}$-valued random variables on probability spaces $(\Omega_1, \mathcal{A}_1, P_1)$ and $(\Omega_2, \mathcal{A}_2, P_2)$, respectively. A *coupling of X and Y* is an $\mathcal{S} \times \mathcal{S}$-valued random variable $Z = (\tilde{X}, \tilde{Y})$ on some probability space $(\tilde{\Omega}, \tilde{\mathcal{A}}, \tilde{P})$ with

1. $\tilde{X} \sim X$ i. e., $\tilde{P}\left[\tilde{X} \leq x, \tilde{Y} \text{ arbitrary }\right] = P[X \leq x]$ and
2. $\tilde{Y} \sim Y$.

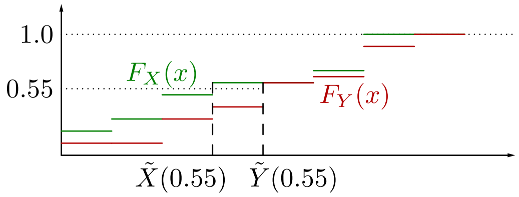


Figure 5.1.: *Illustration of the proof idea for Theorem 5.2.12.*

Couplings are a powerful concept allowing to compare distributions with each other. The following result, which is sometimes refered to as the Theorem of Strassen, gives a well-known connection between stochastic dominance and couplings, see e. g., [Lin92]. It states that stochastic dominance is almost as strong as a pointwise comparison and it is the deeper reason for some of the nice properties of the stochastic dominance order. For instance, the first two statements of Theorem 4.3.2 can easily be derived from it.

Theorem 5.2.12 (e. g., [Lin92]) *For two random variables X and Y the following are equivalent:*

1. $X \leq_{\text{st}} Y$
2. *There is a coupling $Z = (\tilde{X}, \tilde{Y})$ of X and Y such that* $\text{Prob}\left[\tilde{X} \leq \tilde{Y}\right] = 1$.

Proof. This proof is taken from the book of Lindvall [Lin92, p. 127].

1. ⇒ 2. Let F be a distribution function and consider a random variable U distributed uniformly on $[0,1]$. It is well known that the random variable $F^{-1}(U)$ with

$$F^{-1}(u) := \inf\{x \in \mathbb{R} \mid F(x) \geq u\}$$

has distribution function F. We use this fact to construct a coupling Z of X and Y as required (see Figure 5.1 for an illustration). In particular, let $\tilde{X} := F_X^{-1}(U)$ and $\tilde{Y} := F_Y^{-1}(U)$ for a random variable U as above. From the definition of stochastic dominance we have $F_X(x) \geq F_Y(x)$ for all $x \in \mathbb{R}$, which implies $F_X^{-1}(x) \leq F_Y^{-1}(x)$ and thus $\text{Prob}\left[\tilde{X} \leq \tilde{Y}\right] = 1$.

2. ⇒ 1. For all $x \in \mathbb{R}$:

$$\begin{aligned} \text{Prob}\left[X \leq x\right] &= \text{Prob}\left[\tilde{X} \leq x, \tilde{Y} \text{ arbitrary }\right] \\ &\geq \text{Prob}\left[\tilde{X} \leq \tilde{Y} \leq x\right] \\ &= \text{Prob}\left[\tilde{Y} \leq x, \tilde{X} \text{ arbitrary }\right] = \text{Prob}\left[Y \leq x\right]. \end{aligned}$$

□

Remark 5.2.13 For an arbitrary finite set S, consider a uniformly distributed S-valued random variable Σ. Suppose we know $f_1(\Sigma) \leq_{\mathrm{st}} f_2(\Sigma)$ for two real-valued functions f_1 and f_2. The existence of a coupling between $f_1(\Sigma)$ and $f_2(\Sigma)$ according to Theorem 5.2.12 is in this case equivalent to the existence of a bijective mapping $\pi \colon S \to S$ with $f_1(\sigma) \leq f_2(\pi(\sigma))$ for any $\sigma \in S$, which can be obtained by the construction in the proof. Hence, stochastic dominance analysis includes Bijective Analysis as a special case.

The following theorem gives a sufficient criterion for a pairwise comparison between online bin coloring algorithms. It will serve as the theoretical basis for the computer proofs of stochastic dominance between GreedyFit and OneBin obtained in the next section. For a Markov chain $X = (X_n)_{n\in\mathbb{N}_0}$ the l-step Markov chain $X^l = (X^l_n)_{n\in\mathbb{N}_0}$ is defined by

$$X^l_n := X_{ln} \quad \forall n \in \mathbb{N}_0.$$

If P is the transition matrix of X, the transition matrix of X^l is P^l.

Theorem 5.2.14 *Let (X, ϕ) and (Y, ψ) be valued Markov chains on countable state spaces $\mathcal{S}_X$ and $\mathcal{S}_Y$, respectively. Suppose there is a set $S \subseteq \{(x, y) \in \mathcal{S}_X \times \mathcal{S}_Y \mid \phi(x) \leq \psi(y)\}$ and an $l \geq 1$ s. t.*

1. *For $0 \leq i < l$ there exists a coupling $(\tilde{X}_i, \tilde{Y}_i)$ of X_i and Y_i with*

$$\mathrm{Prob}\left[(\tilde{X}_i, \tilde{Y}_i) \in S\right] = 1$$

   and

2. *for every $(x, y) \in S$ there exists a coupling $(\tilde{X}^l, \tilde{Y}^l)$ of $X^l(x)$ and $Y^l(y)$ with*

$$\mathrm{Prob}\left[(\tilde{X}^l, \tilde{Y}^l) \in S\right] = 1.$$

Then we have $\phi(X_n) \leq_{\mathrm{st}} \psi(Y_n)$ for all $n \in \mathbb{N}_0$.

Proof. $0 \leq i < l$ Follows directly using Theorem 5.2.12 and Property 1.

$i \geq l$ We construct a coupling $(\tilde{X}_i, \tilde{Y}_i)$ of X_i and Y_i as follows. First observe that the couplings required by 2. define a transition matrix Z of a Markov chain on state space S where the first component of the state evolves as the l-step Markov chain corresponding to X and the second component as that of Y. Now let $i = ql + r$, $0 \leq r < l$. Let p be the probability distribution of $(\tilde{X}_r, \tilde{Y}_r)$ on S and define $(\tilde{X}_i, \tilde{Y}_i)$ as the random state distributed as pZ^q. By construction, $(\tilde{X}_i, \tilde{Y}_i)$ is in S with probability 1 and has marginal distributions X_i and Y_i, so by Theorem 5.2.12 the claim follows. □

5.2.3. Computer proofs for stochastic dominance relations between bin coloring algorithms

Observe that the set of couplings, or equivalently, the pair (S, l), assumed by Theorem 5.2.14 constitute a *certificate* for the stochastic dominance between the valued Markov chains (X, ϕ) and (Y, ψ). We now describe how such a certificate can be found using a computer-aided construction.

We assume that the colors are generated according to the uniform distribution and consider the Markov chain model of Section 4.3.2. In particular, we study the valued Markov chains for the algorithms OneBin, FixedColors, and GreedyFit. Figure 5.2 gives an example of an explicit construction of the set S for $l = 1$ for a FixedColors and a GreedyFit Markov chain, indicating the obstacles that may occur.

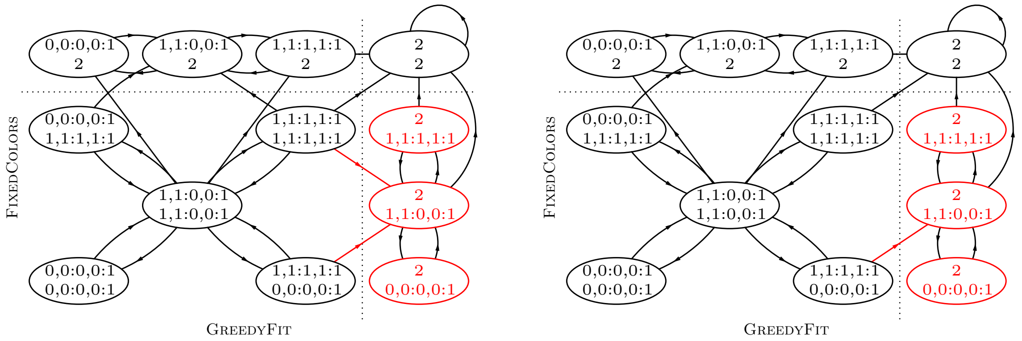
Of course, one can try to algorithmically find a subset S of the product state space $\mathcal{S}_X \times \mathcal{S}_Y$ and an integer l such that the couplings required by Theorem 5.2.14 exist. Our algorithm (see Algorithm 5.1) maintains a set $F \subseteq \mathcal{S}_X \times \mathcal{S}_Y$ of forbidden states, which initially contains those pairs of states $(x, y) \in \mathcal{S}_X \times \mathcal{S}_Y$ with $\phi(x) > \psi(y)$. For a given l, we can assume w. l. o. g. that each Markov chain has l initial states having the i-step distribution as successor distribution, for $0 \leq i < l$. Let $x_0, \ldots, x_{l-1}$ and $y_0, \ldots, y_{l-1}$ be those states for the chains X and Y, respectively. Starting from the states $(x_0, y_0), \ldots, (x_{l-1}, y_{l-1})$, we iteratively try to construct successor-state couplings of all states that can be reached from there such that the probability of reaching a forbidden state is zero. The set G is the set of "good" states that, if the construction is successful, constitute the set S. To keep track of the states for which we still need to find successor-state couplings avoiding F we use a queue Q.

This check is accomplished by procedure Coupling$(X(x), Y(y), F)$ which tries to compute for random states $X(x)$ and $Y(y)$ and a forbidden set $F \subseteq \mathcal{S}_X \times \mathcal{S}_Y$ a coupling $(\tilde{X}, \tilde{Y})$ that avoids states in F. This procedure can be implemented by solving the following Linear Program. Let $p_{x,y}$ be a real variable for the probability to go to state (x, y).

$$\begin{aligned}
\min \quad & 0 & & \\
\text{s.t.} \quad & \sum_{y \in \mathcal{S}_Y} p_{x,y} = \operatorname{Prob}\left[X(x) = x\right] & & \forall x \in \mathcal{S}_X, \\
& \sum_{x \in \mathcal{S}_X} p_{x,y} = \operatorname{Prob}\left[Y(y) = y\right] & & \forall y \in \mathcal{S}_Y, \\
& p_{x,y} = 0 & & \forall (x, y) \in F, \\
& 0 \leq p_{x,y} \leq 1 & & \forall x \in \mathcal{S}_X, y \in \mathcal{S}_Y.
\end{aligned}$$

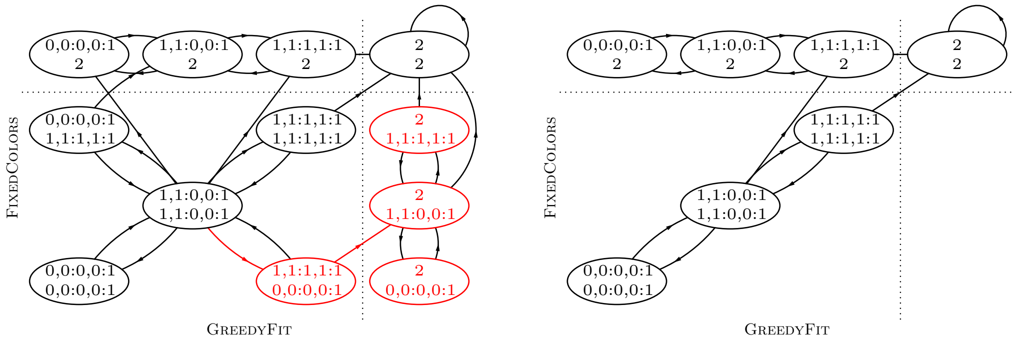
If this Linear Program is feasible, Coupling$(X(x), Y(y), F)$ indicates success and returns the computed probabilities in matrix Z, otherwise it indicates the failure.

If the coupling construction was successful for state (x, y), we need to ensure that we can successfully couple from all states reached via the successor-state coupling. Therefore we add (x, y) to Q and G if it is not in G yet. In the case there is no F-avoiding successor-state coupling from state (x, y), state (x, y) needs to be forbidden, too. Once



(a) First stage: Product coupling with forbidden states (i. e., states that cannot be part of S) and forbidden transitions into the forbidden states.

(b) Second stage: The transition from (1,1;1,1:1 | 1,1;1,1:1) to (1,1;0,0:1 | 2) can be avoided via coupling.



(c) Third stage: The transition from (0,0;0,0:1 | 1,1;1,1:1) to (1,1;0,0:1 | 2) *cannot* be avoided via coupling, so the state (0,0;0,0:1 | 1,1;1,1:1) has to be forbidden as well. The ingoing transition can be avoided via coupling.

(d) Final stage: Resulting coupling consisting of the set S and the transitions staying in S.

Figure 5.2.: *Example construction of a set S that for $l = 1$ shows that* GreedyFit *is superior to* FixedColors *for $m = B = 2$. The lower component of each state is the* GreedyFit *state, the upper the* FixedColors *state.*

```
Input: Valued Markov chains $(X, \phi)$ and $(Y, \psi)$ and their initial states $x_0, \dots, x_{l-1}$ and
    $y_0, \dots, y_{l-1}$
$F \leftarrow \{(x, y) \in \mathcal{S}_X \times \mathcal{S}_Y \mid \phi(x) > \psi(y)\}$
$Q, G \leftarrow \emptyset$
for $i = 0, \dots, l-1$ do
    $G \leftarrow G \cup \{(x_i, y_i)\}$
    if $\neg$UpdateViaCoupling$(x_i, y_i)$ then
        stop; construction failed
    end if
end for
while $Q \neq \emptyset$ do
    remove state $s = (s_1, s_2)$ from $Q$
    if $\neg$UpdateViaCoupling$(s_1, s_2)$ then
        stop; construction failed
    end if
end while
construction successful, $S = G$
procedure UpdateViaCoupling(states $x$ and $y$)
    (success, $(\tilde{X}, \tilde{Y})$) $\leftarrow$ Coupling$(X(x), Y(y), F)$
    if success then
        for $s' \in \{(x, y) \in \mathcal{S}_X \times \mathcal{S}_Y \mid Z(x, y) > 0\}$ do
            if $s' \notin G$ then
                $G \leftarrow G \cup \{s'\}$, $Q \leftarrow Q \cup \{s'\}$
            end if
        end for
    else
        $F \leftarrow F \cup \{(x, y)\}$, $G \leftarrow G \setminus \{(x, y)\}$
        if $(x, y) = (x_i, y_i)$ for some $0 \leq i < l$ (initial state) then
            return FALSE                                        ▷ construction failed
        end if
        $Q \leftarrow Q \cup$ {predecessors of $(x, y)$ according to $Z$}
    end if
    return TRUE
end procedure
```

Algorithm 5.1: *Try to construct a certificate for stochastic dominance.*

one of the initial states has to be forbidden, it is clear that we cannot avoid entering a forbidden state at all and thus no set S exists.

We implemented the algorithm based on exact arithmetic using the GNU Multiple Precision Arithmetic Library [GMP06] (GMP) and the exact LP solver QSopt_ex [ACDE07] in C++. Table 5.1 reports our results for the three bin coloring algorithms, in particular the smallest value of l such that our algorithm found appropriate couplings according to Theorem 5.2.14. The correctness of the constructed couplings does not depend on the construction method, but only on the correct implementation of the (simple) routine which checks the conditions of Theorem 5.2.14 and the correctness of the rational arithmetic provided by GMP used in that routine. Assuming this, we get the following result.

Theorem 5.2.15 *Denote by* GF, FC, *and* OB *the valued Markov chains for the* max-BC *for sequences with the uniform color distributions. For the values of m, B, and C given in Table 5.1, we have*

$$\chi\big(\mathsf{GF}_n(m,B,C)\big) \leq_{\mathrm{st}} \chi\big(\mathsf{FC}_n(m,B,C)\big) \leq_{\mathrm{st}} \chi\big(\mathsf{OB}_n(m,B,C)\big)$$

for all $n \in \mathbb{N}_0$.

Observe that for $B = 2$ and for GreedyFit against OneBin $l = 1$ is sufficient, indicating that the “proof” is particularly simple. In fact, the case $B = 2$ can in general be dealt with similarly to Figure 5.1 and GreedyFit against OneBin admits a rigorous proof presented in the next section.

5.3. Rigorous proofs of the stochastic dominance between OneBin and GreedyFit

We mentioned already in Chapter 4 that the monotonicity technique [MS02] and the comparison criterion of Doisy [Doi00] are not sufficient to prove the observed stochastic dominance between OneBin and GreedyFit. Our analysis is therefore based on Theorem 4.3.5, which states that stochastic dominance at every point in time is equivalent to stochastic dominance between certain stopping times.

So far we assumed that the objective of the bin coloring algorithms is to minimize the *maximum* colorfulness attained in any bin. Another reasonable objective is to minimize the *average* colorfulness of all the bins, or equivalently the sum of the colorfulnesses of all bins. From now on, we refer to the first problem as max-BC and to the second one as sum-BC.

5.3.1. Markov chain models and preliminaries

We start by describing a valued Markov chain model for max-BC algorithms that is a generalization of the Markov chain model given in Section 4.3.2. The operation of any such bin coloring algorithm can be described on a state space which encodes for every

Parameters	GF vs. OB	FC vs. OB	GF vs. FC
$m = 2$, $B = 2$, $C = 4$	1	1	1
$m = 2$, $B = 3$, $C = 6$	1	4	2
$m = 2$, $B = 4$, $C = 8$	1	6	8
$m = 2$, $B = 5$, $C = 10$	1	8	9
$m = 2$, $B = 6$, $C = 12$	1	10	9
$m = 2$, $B = 7$, $C = 14$	1	12	11
$m = 3$, $B = 2$, $C = 6$	1	1	1
$m = 3$, $B = 3$, $C = 9$	1	4	4
$m = 3$, $B = 4$, $C = 12$	1	6	6
$m = 3$, $B = 5$, $C = 15$	1	8	6
$m = 4$, $B = 2$, $C = 8$	1	1	1
$m = 4$, $B = 3$, $C = 12$	1	4	3
$m = 4$, $B = 4$, $C = 16$	1	6	4
$m = 5$, $B = 2$, $C = 10$	1	1	1
$m = 5$, $B = 3$, $C = 15$	1	4	2
$m = 5$, $B = 4$, $C = 20$	1	6	3
$m = 3$, $B = 5$, $C = 6$	1	1	1
$m = 3$, $B = 5$, $C = 9$	1	6	5
$m = 3$, $B = 5$, $C = 12$	1	8	6
$m = 3$, $B = 5$, $C = 15$	1	8	6
$m = 3$, $B = 5$, $C = 18$	1	8	6
$m = 3$, $B = 5$, $C = 21$	1	8	6
$m = 3$, $B = 5$, $C = 24$	1	8	7
$m = 3$, $B = 5$, $C = 27$	1	8	7
$m = 3$, $B = 5$, $C = 30$	1	8	8
$m = 3$, $B = 5$, $C = 33$	1	8	8
$m = 3$, $B = 5$, $C = 36$	1	8	8
$m = 3$, $B = 5$, $C = 39$	1	8	8
$m = 3$, $B = 5$, $C = 42$	1	8	8
$m = 3$, $B = 5$, $C = 45$	1	8	10

Table 5.1.: *Results of our algorithm on the parameter sets defined in Section 4.3.2. The reported numbers are the smallest values l for which couplings proving stochastic dominance via Theorem 5.2.14 could be constructed.*

bin i its current number of items f_i and the set of colors in that bin C_i. Moreover, the state also keeps track of the maximum colorfulness attained so far. Formally, we have

$$\mathcal{S}_{\text{max-BC}} := \mathcal{S}_{\text{max-BC}}(m,B,C) = \Big\{(f_1,C_1,\dots,f_m,C_m,\chi) \;\Big|\; 0 \le |C_i| \le f_i \le B, \\ |C_i| \le \chi \le \min\{B,C\}\Big\}.$$

Note that the states reachable by the operation of an algorithm may be a subset of $\mathcal{S}_{\text{max-BC}}$.

We will use $f_i(s)$, $C_i(s)$, and $\chi(s)$ to refer to the components of state s. Additionally, we define $c_i(s) := |C_i(s)|$. The state $(0,\emptyset,\dots,0,\emptyset,0)$ is called the *initial empty state.*

Suppose an online bin coloring algorithm A is in state s and receives an item of color c. The algorithm then changes to state s' by putting this item in one of the bins, say bin i. There are two cases: Either color c is contained in $C_i(s)$, we say the color is *known (in bin i)*, or it is not, so the color is *new (in bin i).* We will denote the successor state for the first case by $s^{\text{known}(i)}$ (the color c is not needed to determine the successor state), for the second by $s^{\text{new}(i,c)}$. It will be convenient not to consider the new color c, but to deal with the random state resulting from s if any new color distributed according to γ is seen. We will use the notation $s^{\mathbf{new}(i)}$ for this random state.

The OneBin algorithm is then described by the transitions

$$s' = \begin{cases} s^{\text{known}(1)} & \text{with probability } \gamma(C_1(s)), \\ s^{\mathbf{new}(1)} & \text{with probability } 1-\gamma(C_1(s)), \end{cases} \tag{5.4}$$

where we use the shortcut notation $\gamma(S) := \sum_{s\in S}\gamma(s)$. This defines a Markov chain which we denote by $\mathsf{OB}(m,B,C,\gamma)$. Note that although OneBin uses only the first bin, we consider $\mathsf{OB}(m,B,C,\gamma)$ as working on the whole state space with m bins.

Similarly, we can give a Markov chain $\mathsf{GF}(m,B,C,\gamma)$ for $\mathsf{GreedyFit}$. $\mathsf{GF}(s)$ is the bin $\mathsf{GreedyFit}$ selects for an item with a new color in state s. Depending on the tie-breaking rule used by the specific variant of $\mathsf{GreedyFit}$, $\mathsf{GF}(s)$ may or may not be a random variable. We only need that $\mathsf{GF}(s)$ is one of the bins having in state s the smallest number of colors.

$$s' = \begin{cases} s^{\text{known}(i)} & \text{with probability } \gamma(C_i(s)) \quad 1 \le i \le m, \\ s^{\mathbf{new}(\mathsf{GF}(s))} & \text{with probability } 1-\gamma\big(\bigcup_i C_i(s)\big). \end{cases} \tag{5.5}$$

The operation of online algorithms in the sum-BC problem can be captured by a similar Markov chain model. The main difference is that the χ-component is no longer the maximum of the colorfulness seen so far, but the sum. Note that the resulting Markov chains are infinite. We can use the state space

$$\mathcal{S}_{\text{sum-BC}} := \mathcal{S}_{\text{sum-BC}}(m,B,C) = \Big\{(f_1,C_1,\dots,f_m,C_m,\chi) \;\Big|\; 0 \le |C_i| \le f_i \le B, \\ |C_i| \le \min\{B,C\}, \chi \in \mathbb{N}_0\Big\}.$$

The χ-component increases each time a new color for a bin is encountered.

To avoid notational overhead, we will use the same notation for both problem variants. Therefore the sum-BC-Markov chains for OneBin and GreedyFit will be denoted by $\mathsf{OB}(m, B, C, \gamma)$ and $\mathsf{GF}(m, B, C, \gamma)$, too. We use the notations $\mathsf{OB}(m, B, C, \gamma)_n$ and $\mathsf{GF}(m, B, C, \gamma)_n$ for the random state after n steps when OneBin and GreedyFit are started in the initial empty state.

Consider valued Markov chains (X, χ) and (Y, χ) corresponding to bin coloring algorithms, where χ measures either the maximum or the sum of the colorfulnesses encountered. Recall that for a valued Markov chain (X, χ), we denote by T_X^v the first time the Markov chain X reaches a state with valuation at least v. By Theorem 4.3.5, it is suffient to show $T_Y^v \leq_{\mathrm{st}} T_X^v$ for all $v \in V$ to conclude show $\chi(X_n) \leq_{\mathrm{st}} \chi(Y_n)$ for all $n \in \mathbb{N}_0$. We will therefore analyze the stopping times to prove our stochastic dominance result. In the sequel, we denote by $T_X^v(s)$ the stopping time for reaching a state with valuation at least v when started deterministically in state in s.

How can we show $T_Y^v(s_0) \leq_{\mathrm{st}} T_X^v(s_0)$? In order to apply a kind of induction technique we introduce a family of Markov chains $\big(X(n)\big)_{n\in\mathbb{N}}$ derived from a Markov chain X as follows. The state space of $X(n)$ is $\mathcal{S} \times \{0, \ldots, n\}$ and the transitions are defined by

$$\begin{aligned} \operatorname{Prob}\big[X(n)_{i+1} = (s', i+1) \mid X(n)_i = (s, i)\big] &:= \operatorname{Prob}\big[X_{i+1} = s' \mid X_i = s\big] && \forall 0 \leq i < n, \\ \operatorname{Prob}\left[X(n)_{i+1} = (s, n) \mid X(n)_i = (s, n)\right] &:= 1 && \forall i \geq n. \end{aligned}$$

The Markov chain $X(n)$ can be thought of as an time-expanded, acyclic version of the chain X for the first n steps. Clearly, we have

$$\operatorname{Prob}\left[T_X^v(s) = i\right] = \operatorname{Prob}\left[T_{X(n)}^v((s,0)) = i\right] \quad \forall 0 \leq i < n. \tag{5.6}$$

So in order to show $T_Y^v(s_0) \leq_{\mathrm{st}} T_X^v(s_0)$, we can prove that

$$T_{Y(n)}^v((s_0, 0)) \leq_{\mathrm{st}} T_{X(n)}^v((s_0, 0)) \quad \forall n \in \mathbb{N}.$$

To simplify notation, we will write $T_{X(n)}^v(s)$ for $T_{X(n)}^v((s,0))$ from now on. We have the following simple result.

Lemma 5.3.1 *For any valued Markov chain (X, χ) the stochastic dominance relation*

$$T_{X(n+1)}^v(s) \leq_{\mathrm{st}} T_{X(n)}^v(s)$$

holds for all states s, $n \in \mathbb{N}_0$, and $v \in V$.

Proof. Consider a sample path $\omega = (X_0, X_1, \ldots, X_n)$ of $X(n)$ with $X_0 = s$. Obviously, ω can be extended to a sample path $\omega' = (X_0, X_1, \ldots, X_n, X_{n+1})$ of $X(n+1)$ and all sample paths of $X(n+1)$ starting in s are obtained this way. There are two cases:

1. $\chi(X_n) \geq v$: For all ω' that are extensions of ω we have $T_{X(n+1)}^v(s) = T_{X(n)}^v(s)$.
2. $\chi(X_n) < v$: For those ω, $T_{X(n)}^v(s)$ is infinite, whereas for any extension ω' $T_{X(n+1)}^v(s)$ is either $n+1$ or infinite, too. Thus $T_{X(n+1)}^v(s) \leq T_{X(n)}^v(s)$. □

To analyze the stopping times, we will employ the concept of a mixture of random variables.

Definition 5.3.2 Let $(X_m)_{m\in M}$ be a family of random variables and Θ be an M-valued random variable. The random variable Y defined by $Y := X_\Theta$, i. e., the X-variable to use is given by the realization of Θ, is called a *mixture* and denoted by $[(X_m)_{m\in M} \mid \Theta]$.

An important property of $\leq_{\mathrm{st}}$ is that it is closed under mixture, as stated in the Mixture Theorem.

Theorem 5.3.3 ([MS02, p. 6]) *Suppose* $[(X_m)_{m\in M} \mid \Theta]$ *and* $[(Y_m)_{m\in M} \mid \Theta]$ *are two mixtures controlled by the same random variable* Θ *satisfying* $X_m \leq_{\mathrm{st}} Y_m$ *for all* $m \in M$*. Then we have*

$$[(X_m)_{m\in M} \mid \Theta] \leq_{\mathrm{st}} [(Y_m)_{m\in M} \mid \Theta]\,.$$

For two random variables X and Y, we will frequently write $X = Y$ to mean that they have the same distribution function.

5.3.2. GreedyFit is better than OneBin: max-BC

We will now apply the strategy described above to the comparison of the GreedyFit and the OneBin bin coloring algorithms. The main technique is to analyze a kind of stochastic recursion for $T^v_{X(n)}$ based on a mixture of random variables.

Let $\mathsf{OB} = \mathsf{OB}(m, B, C, \gamma)$ for fixed parameters m, B, C, γ. In a state $s \in \mathcal{S}_{\text{max-BC}}$ OneBin does the transitions to states

$$\begin{cases} s^{\mathrm{known}(1)} & \text{with probability } \gamma(C_1(s)),\\ s^{\mathbf{new}(1)} & \text{with probability } 1-\gamma(C_1(s)).\end{cases}$$

Using the random variable $\Theta\colon \mathcal{S}_{\text{max-BC}} \to \mathbb{N}$ defined by

$$\Theta(s) := \begin{cases} 1 & \text{the next color is known in bin 1,}\\ 2 & \text{the next color is new in bin 1,}\end{cases}$$

we can come up with a recursive expression for $T^v_{\mathsf{OB}(n)}(s)$, namely

$$T^v_{\mathsf{OB}(n)}(s) = \begin{cases} 0 & \chi(s) \geq v,\\ 1 + \Big[T^v_{\mathsf{OB}(n-1)}\big(s^{\mathrm{known}(1)}\big), T^v_{\mathsf{OB}(n-1)}\big(s^{\mathbf{new}(1)}\big) \,\Big|\, \Theta(s)\Big] & \chi(s) < v.\end{cases} \tag{5.7}$$

This recursion and the Mixture Theorem 5.3.3 are the most important ingredients for the proofs to come.

We call two states $s, s' \in \mathcal{S}_{\text{max-BC}}$ OB-*equivalent*, if the valuation, the number of items and the set of colors in bin 1 are the same in s and s', i. e., if $\chi(s) = \chi(s')$, $f_1(s) = f_1(s')$, and $C_1(s) = C_1(s')$. Note that OneBin behaves exactly the same in two OB-equivalent states and therefore the stopping times from two OB-equivalent states coincide. The following lemma gives some useful comparisons of stopping times from certain states in the $\mathsf{OB}(n)$ chains.

Lemma 5.3.4 *Consider the* OneBin *Markov chain* $\mathsf{OB} = \mathsf{OB}(m, B, C, \gamma)$ *for parameters* $m, B \geq 2$, C, *and color distribution* γ. *We have for all states* $s \in \mathcal{S}_{\text{max-BC}}$, $n \in \mathbb{N}$, *and* $v \in V$*:*

1. $T^v_{\mathsf{OB}(n)}(s^{\mathbf{new}(1)}) \leq_{\text{st}} T^v_{\mathsf{OB}(n)}(s^{\text{known}(1)})$, *and*
2. $T^v_{\mathsf{OB}(n)}(s^{\mathbf{new}(1)}) \leq_{\text{st}} T^v_{\mathsf{OB}(n)}(s')$ *for every state* s' *that is OB-equivalent to* s.

Proof. Let $s \in \mathcal{S}_{\text{max-BC}}$ be such that $\chi(s^{\mathbf{new}(1)}) < v$ (the case $\chi(s^{\mathbf{new}(1)}) \geq v$ is trivial).

1. Observe that both $s^{\mathbf{new}(1)}$ and $s^{\text{known}(1)}$ have the same number of items in bin 1, say f. In the case $f = 0$ both states are OB-equivalent, since bin 1 is empty then. Therefore, $T^v_{\mathsf{OB}(n)}(s^{\mathbf{new}(1)}) = T^v_{\mathsf{OB}(n)}(s^{\text{known}(1)})$.

   For $f > 0$ the evolution from both states will be identical after $B - f$ steps, since bin 1 is then empty again. It is therefore sufficient to show
   $$T^v_{\mathsf{OB}(B-f)}(s^{\mathbf{new}(1)}) \leq_{\text{st}} T^v_{\mathsf{OB}(B-f)}(s^{\text{known}(1)})$$
   for all $1 \leq f < B$. We will show this by induction on $j := B - f$.

   To start the induction, consider $j = 1$. There are two cases:

   - $c_1(s) = v - 2$: We have $\text{Prob}\left[T^v_{\mathsf{OB}(j)}(s^{\mathbf{new}(1)}) \leq 1\right] \leq 1$ and $\text{Prob}\left[T^v_{\mathsf{OB}(j)}(s^{\text{known}(1)}) = \infty\right] = 1$.
   - $c_1(s) < v - 2$: Then $T^v_{\mathsf{OB}(j)}(s^{\mathbf{new}(1)}) = T^v_{\mathsf{OB}(j)}(s^{\text{known}(1)}) \equiv \infty$.

   In both cases, the stochastic dominance is immediate.

   Let us now consider the induction step, i. e., $j > 1$. The key observation is that (since we need at least two items to close bin 1)
   $$f_1(s^{\mathbf{new}(1),\text{known}(1)}) = f_1(s^{\text{known}(1),\mathbf{new}(1)}) \quad \text{and}$$
   $$C_1(s^{\mathbf{new}(1),\text{known}(1)}) = C_1(s^{\text{known}(1),\mathbf{new}(1)}),$$
   which means that both states are OB-equivalent. Using the Mixture Theorem 5.3.3, we can then estimate
   $$\begin{aligned}
   &T^v_{\mathsf{OB}(j)}(s^{\mathbf{new}(1)}) \\
   &= 1 + \left[T^v_{\mathsf{OB}(j-1)}(s^{\mathbf{new}(1),\text{known}(1)}), T^v_{\mathsf{OB}(j-1)}(s^{\mathbf{new}(1),\mathbf{new}(1)}) \,\middle|\, \Theta(s^{\mathbf{new}(1)})\right] \\
   &\leq_{\text{st}} 1 + T^v_{\mathsf{OB}(j-1)}(s^{\mathbf{new}(1),\text{known}(1)}) \\
   &\hspace{20em} \text{(by induction)} \\
   &= 1 + T^v_{\mathsf{OB}(j-1)}(s^{\text{known}(1),\mathbf{new}(1)}) \\
   &\hspace{20em} (\mathsf{OB}\text{-equivalence}) \\
   &\leq_{\text{st}} 1 + \left[T^v_{\mathsf{OB}(j-1)}(s^{\text{known}(1),\text{known}(1)}), T^v_{\mathsf{OB}(j-1)}(s^{\text{known}(1),\mathbf{new}(1)}) \,\middle|\, \Theta(s^{\text{known}(1)})\right] \\
   &\hspace{20em} \text{(by induction)} \\
   &= T^v_{\mathsf{OB}(j)}(s^{\text{known}(1)}).
   \end{aligned}$$

2. Since s and s' are OB-equivalent, we have

$$\begin{aligned}
&T^v_{\mathsf{OB}(n)}(s') = T^v_{\mathsf{OB}(n)}(s) && (\mathsf{OB}\text{-equivalence})\\
&= 1 + \left[T^v_{\mathsf{OB}(n-1)}(s^{\mathbf{new}(1)}), T^v_{\mathsf{OB}(n-1)}(s^{\mathrm{known}(1)}) \,\middle|\, \Theta(s)\right] &&\\
&\geq_{\mathrm{st}} 1 + T^v_{\mathsf{OB}(n-1)}(s^{\mathbf{new}(1)}) && (\text{by 1.})\\
&\geq_{\mathrm{st}} 1 + T^v_{\mathsf{OB}(n)}(s^{\mathbf{new}(1)}) && (\text{by Lemma 5.3.1})\\
&\geq_{\mathrm{st}} T^v_{\mathsf{OB}(n)}(s^{\mathbf{new}(1)}). && \square
\end{aligned}$$

Theorem 5.3.5 *Let* OB *and* GF *be the* OneBin *and* GreedyFit max-BC-*Markov chains for fixed parameters* m, B, C *with* $B, m \geq 2$ *for some color distribution* γ*. We have for all states* $s \in \mathcal{S}_{\text{max-BC}}$*,* $n \in \mathbb{N}$*, and* $v \in V$*:*

$$T^v_{\mathsf{OB}(n)}(s) \leq_{\mathrm{st}} T^v_{\mathsf{GF}(n)}(s).$$

Proof. The proof is by induction on n. Since GreedyFit is not worse than OneBin for a single step in each state s, we have $T^v_{\mathsf{OB}(1)}(s) \leq_{\mathrm{st}} T^v_{\mathsf{GF}(1)}(s)$.

The proof idea for the induction step is depicted in Figure 5.3. Suppose we know that $T^v_{\mathsf{OB}(n)}(s) \leq_{\mathrm{st}} T^v_{\mathsf{GF}(n)}(s)$ for all $s \in \mathcal{S}_{\text{max-BC}}$. Consider a state $s \in \mathcal{S}_{\text{max-BC}}$. Define the random variable $\Theta\colon \mathcal{S}_{\text{max-BC}} \to \{1, \dots, m+1\}$ by

$$\mathrm{Prob}\left[\Theta(s) = i\right] = \begin{cases} \gamma(C_i(s)) & 1 \leq i \leq m,\\ 1 - \gamma\big(\bigcup_i C_i(s)\big) & i = m+1, \end{cases}$$

i. e., Θ in a sense "selects" the GreedyFit successor of state s. Using Θ, we can write the recursion for the stopping time of OB as

$$\begin{aligned}
&T^v_{\mathsf{OB}(n+1)}(s)\\
&\quad = 1 + \left[T^v_{\mathsf{OB}(n)}(s^{\mathrm{known}(1)}), T^v_{\mathsf{OB}(n)}(s^{\mathbf{new}(1)}), \dots, T^v_{\mathsf{OB}(n)}(s^{\mathbf{new}(1)}) \,\middle|\, \Theta(s)\right].
\end{aligned}$$

Observe that $s^{\mathrm{known}(i)}$, $2 \leq i \leq m$, are OB-equivalent to s, $s^{\mathbf{new}(\mathsf{GF}(s))}$ is either OB-equivalent to s or equal to $s^{\mathbf{new}(1)}$. We use Lemma 5.3.4 to bound this by

$$\leq_{\mathrm{st}} 1 + \left[T^v_{\mathsf{OB}(n)}(s^{\mathrm{known}(1)}), \dots, T^v_{\mathsf{OB}(n)}(s^{\mathrm{known}(m)}), T^v_{\mathsf{OB}(n)}(s^{\mathbf{new}(\mathsf{GF}(s))}) \,\middle|\, \Theta(s)\right],$$

which by the induction hypothesis is bounded by

$$\begin{aligned}
&\leq_{\mathrm{st}} 1 + \left[T^v_{\mathsf{GF}(n)}(s^{\mathrm{known}(1)}), \dots, T^v_{\mathsf{GF}(n)}(s^{\mathrm{known}(m)}), T^v_{\mathsf{GF}(n)}(s^{\mathbf{new}(\mathsf{GF}(s))}) \,\middle|\, \Theta(s)\right]\\
&= T^v_{\mathsf{GF}(n+1)}(s). \qquad \square
\end{aligned}$$

This concludes the induction step and the proof.

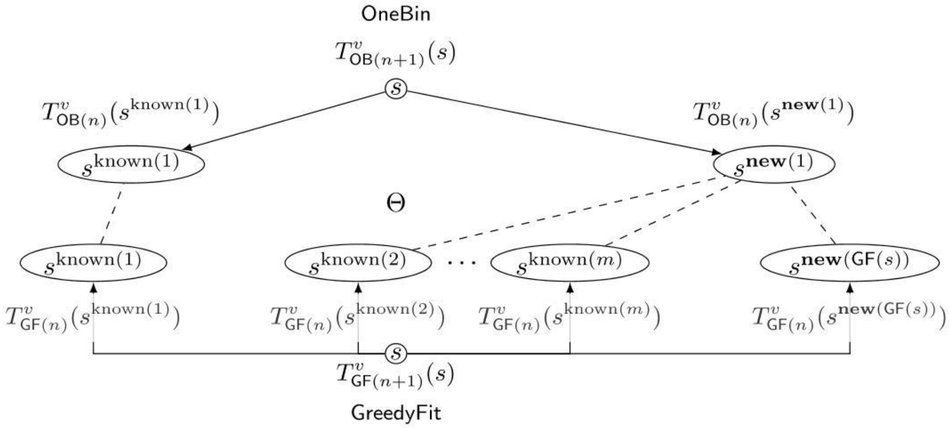

OneBin
$T^v_{\mathsf{OB}(n+1)}(s)$
s
$T^v_{\mathsf{OB}(n)}(s^{\text{known}(1)})$
$T^v_{\mathsf{OB}(n)}(s^{\textbf{new}(1)})$
$s^{\text{known}(1)}$
$s^{\textbf{new}(1)}$
Θ
$s^{\text{known}(1)}$
$s^{\text{known}(2)}$
$\cdots$
$s^{\text{known}(m)}$
$s^{\textbf{new}(\mathsf{GF}(s))}$
$T^v_{\mathsf{GF}(n)}(s^{\text{known}(1)})$
$T^v_{\mathsf{GF}(n)}(s^{\text{known}(2)})$
$T^v_{\mathsf{GF}(n)}(s^{\text{known}(m)})$
$T^v_{\mathsf{GF}(n)}(s^{\textbf{new}(\mathsf{GF}(s))})$
s
$T^v_{\mathsf{GF}(n+1)}(s)$
GreedyFit

Figure 5.3.: *Proof idea for induction step in proof of Theorem 5.3.5. The upper part shows the transitions of the* OB *Markov chain, the lower part those of the* GF *chain, both with corresponding stopping times. Using Lemma 5.3.4 and the induction hypothesis, one can map (dashed lines indicate mapping* Θ*) the successor states of the* OB *chain to those of the* GF *chain such that we have stochastic dominance for each pair of stopping times.*

Corollary 5.3.6 *Let* OB *and* GF *be the* OneBin *and* GreedyFit max-BC-*Markov chains for fixed parameters* m, B, C *and color distribution* γ*. We have for all states* $s \in \mathcal{S}_{\text{max-BC}}$*, in particular the initial empty state, and for all* $n \in \mathbb{N}_0$ *that*

$$\chi\big(\mathsf{GF}(s)_n\big) \leq_{\text{st}} \chi\big(\mathsf{OB}(s)_n\big).$$

Proof. The cases $m = 1$ and $B = 1$ are trivial. For the remaining cases, combine Theorems 4.3.5 and 5.3.5 and the relation of $\mathsf{OB}(n)$ and OB as well as $\mathsf{GF}(n)$ and GF according to Equation (5.6). □

5.3.3. GreedyFit is better than OneBin: sum-BC

The analysis of the sum-BC problem is very similar to the one of max-BC in the preceding section. Recall that the state space of the sum-BC only differs from the one of the max-BC in its interpretation of the χ-component: it now counts the sum of the colorfulnesses of all used bins instead of the maximum. Therefore, the χ-component increases with every transition due to a new color. Nevertheless, recursion (5.7) for the stopping times is also valid for the analysis of the sum-BC.

Note that the proof of Theorem 5.3.5 is based only on Lemma 5.3.4. The proof of item 2 of Lemma 5.3.4 only needs item 1 and OB-equivalence. The notion of OB-equivalence introduced for the max-BC is also appropriate for the sum-BC. In particular, stopping times for two OB-equivalent states coincide also for the sum-BC-Markov chain of OneBin. Due to these observations, it is sufficient to prove an analogue of item 1 of Lemma 5.3.4 to establish stochastic dominance between GreedyFit and OneBin for the sum-BC. The proof uses the concept of a coupling Markov chain.

Definition 5.3.7 Let $X = (X_n)_{n\in\mathbb{N}_0}$ and $Y = (Y_n)_{n\in\mathbb{N}_0}$ be Markov chains on state spaces $\mathcal{S}_X$ and $\mathcal{S}_Y$, respectively. A Markov chain $Z = (\tilde{X}, \tilde{Y})$ on state space $\mathcal{S}_X \times \mathcal{S}_Y$ is a *coupling Markov chain* if $\tilde{X}$ and $\tilde{Y}$ are distributed as X and Y, respectively. However, $\tilde{X}$ and $\tilde{Y}$ need not be independent.

Lemma 5.3.8 *Consider the* OneBin *Markov chain* $\mathsf{OB} = \mathsf{OB}(m, B, C, \gamma)$ *for parameters* $m, B \geq 2$*,* C*, and color distribution* γ *for the* sum-BC*. We then have*

$$T^v_{\mathsf{OB}(n)}(s^{\mathbf{new}(1)}) \leq_{\mathrm{st}} T^v_{\mathsf{OB}(n)}(s^{\mathrm{known}(1)})$$

for all states $s \in \mathcal{S}_{\text{sum-BC}}$*,* $n \in \mathbb{N}$*, and* $v \in V$*.*

Proof. We will show the stronger $T^v_{\mathsf{OB}}(s^{\mathrm{new}(1,c)}) \leq_{\mathrm{st}} T^v_{\mathsf{OB}}(s^{\mathrm{known}(1)})$ for all $c \notin C_1(s)$ by constructing a coupling Markov chain $Z = (X, Y)$ on a state space that is a subset of $\mathcal{S}_{\text{sum-BC}} \times \mathcal{S}_{\text{sum-BC}}$. The first component of Z behaves exactly as OB started in state $s^{\mathrm{new}(1,c)}$ and the second component as OB started in $s^{\mathrm{known}(1)}$.

A state $(s^{\mathrm{n}}, s^{\mathrm{k}})$ of Z that can be reached from the initial state $(s^{\mathrm{new}(1,c)}, s^{\mathrm{known}(1)})$ will always satisfy the invariant

- either $\chi(s^{\mathrm{n}}) \geq \chi(s^{\mathrm{k}})$, $f_1(s^{\mathrm{n}}) = f_1(s^{\mathrm{k}})$, and $C_1(s^{\mathrm{n}}) = C_1(s^{\mathrm{k}})$ or
- $\chi(s^{\mathrm{n}}) = \chi(s^{\mathrm{k}}) + 1$, $f_1(s^{\mathrm{n}}) = f_1(s^{\mathrm{k}})$, and $C_1(s^{\mathrm{n}}) = C_1(s^{\mathrm{k}}) \cup \{c\}$.

Since in both cases $\chi(s^{\mathrm{n}}) \geq \chi(s^{\mathrm{k}})$, the invariant implies

$$\mathrm{Prob}\left[T^v_X(s^{\mathrm{new}(1,c)}) \leq T^v_Y(s^{\mathrm{known}(1)})\right] = 1,$$

so by Strassen's Theorem the stochastic dominance is established.

It remains to describe Z. The initial state is $(s^{\mathrm{new}(1,c)}, s^{\mathrm{known}(1)})$, which obviously satisfies the invariant. Consider any state $(s^{\mathrm{n}}, s^{\mathrm{k}})$ satisfying the invariant. If s^{n} and s^{k} differ at most in the χ-component, then the transitions of Z are such that the same happens in both components, leading to further states satisfying the invariant.

Suppose s^{n} and s^{k} differ also in the C_1-component. The transitions are then determined by the next color c' drawn according to γ as follows:

$$\begin{cases} \left(s^{\mathrm{n,new}(1,c')}, s^{\mathrm{k,new}(1,c')}\right) & c' \notin C_1(s^{\mathrm{n}}) = C_1(s^{\mathrm{k}}) \cup \{c\}, \\ \left(s^{\mathrm{n,known}(1)}, s^{\mathrm{k,new}(1,c')}\right) & c' = c, \\ \left(s^{\mathrm{n,known}(1)}, s^{\mathrm{k,known}(1)}\right) & c' \in C_1(s^{\mathrm{k}}). \end{cases}$$

Note that all the states satisfy the invariant and that the second kind of transition leads to states which differ at most in the χ-component (the other way of reaching such a state is when bin 1 is empty again). Finally, we can verify that these transitions mirror the behavior of the OB chain in each component:

$$\mathrm{Prob}\left[X_{n+1} = s^{\mathrm{n,new}(1,c')} \middle| X_n = s^{\mathrm{n}}\right] = 1 - \gamma(C_1(s^{\mathrm{n}})),$$

$$\mathrm{Prob}\left[X_{n+1} = s^{\mathrm{n,known}(1)} \middle| X_n = s^{\mathrm{n}}\right] = \gamma(C_1(s^{\mathrm{n}})),$$

$$\begin{aligned}\text{Prob}\left[Y_{n+1} = s^{\mathrm{k,new}(1,c')} \middle| Y_n = s^{\mathrm{k}}\right] &= 1 - \gamma(C_1(s^{\mathrm{k}})),\\ \text{Prob}\left[Y_{n+1} = s^{\mathrm{k,known}(1)} \middle| Y_n = s^{\mathrm{k}}\right] &= \gamma(C_1(s^{\mathrm{k}})).\end{aligned} \qquad \square$$

Theorem 5.3.9 *Let* OB *and* GF *be the* OneBin *and* GreedyFit sum-BC-*Markov chains for fixed parameters m, B, C and color distribution γ. We have for all states $s \in \mathcal{S}_{\text{sum-BC}}$, in particular the initial empty state, and for all $n \in \mathbb{N}_0$ that*

$$\chi\big(\mathsf{GF}(s)_n\big) \leq_{\text{st}} \chi\big(\mathsf{OB}(s)_n\big).$$

Appendix A.

Tables of computational results

A.1. Simulation results for elevator control algorithms

The following tables provide the detailed results of our extensive simulation studies described in Section 3.3. For each of the two buildings and every population percentage between 6% and 20% we report the waiting time median $\alpha_{0.5}$, the 0.9-quantile $\alpha_{0.9}$ of the waiting time, and the number of reissues as a fraction of the total number of passengers. Each of these values is determined from ten independent simulation runs, with different algorithms running on exactly the same instances.

From this data, we can determine the handling capacity according to our definition of "reasonable" service quality, meaning that $\alpha_{0.5}$ is at most 25 seconds, $\alpha_{0.9}$ is at most 55 seconds, and at most 10% of the passengers have to reissue their call. Since these thresholds are sometimes missed by a small margin only and the next precentage level gives much worse results, we were less strict and actually used 58 seconds and less than 11% as the critical thresholds. The row that determines the handling capacity is marked grey, with the limiting value indicated in bold.

scenario	IA, linear cost				DA, linear cost				IA, quadratic cost				DA, quadratic cost			
	$\alpha_{0.5}$	$\alpha_{0.9}$	$\emptyset$	reissues	$\alpha_{0.5}$	$\alpha_{0.9}$	$\emptyset$	reissues	$\alpha_{0.5}$	$\alpha_{0.9}$	$\emptyset$	reissues	$\alpha_{0.5}$	$\alpha_{0.9}$	$\emptyset$	reissues
Up 6%	2	18	6	0.1%	0	13	4	0.2%	8	24	10	0.0%	6	24	9	0.0%
Up 7%	2	23	7	1.8%	1	16	5	1.2%	8	28	11	0.2%	5	23	8	0.0%
Up 8%	3	24	8	2.8%	0	17	5	2.5%	8	29	11	0.2%	5	24	9	0.1%
Up 9%	4	28	9	6.3%	0	18	6	6.2%	8	29	11	0.3%	4	25	9	0.6%
Up 10%	5	33	11	**9.5%**	1	27	8	**8.6%**	9	32	12	1.2%	4	27	10	1.4%
Up 11%	8	43	15	16.9%	4	32	10	14.7%	11	36	14	5.7%	5	29	10	3.1%
Up 12%	9	50	17	27.3%	6	46	15	24.6%	12	38	16	**10.4%**	5	33	11	**8.9%**
Up 13%	15	66	24	41.8%	8	59	19	37.0%	17	51	21	21.5%	8	41	15	12.6%
Up 14%	19	98	34	56.9%	12	81	28	56.6%	24	73	31	37.1%	16	60	23	34.5%
Up 15%	28	110	43	63.6%	19	107	38	66.1%	30	88	37	48.3%	21	76	30	46.6%
Up 16%	34	140	54	79.0%	29	129	49	79.5%	37	114	48	62.1%	30	98	41	62.0%
Up 17%	44	159	65	90.1%	37	152	60	93.8%	46	135	58	73.8%	40	116	51	76.3%
Up 18%	56	175	76	106.4%	59	173	73	114.0%	59	148	68	87.2%	50	131	59	86.4%
Up 19%	80	199	89	118.5%	70	191	84	116.1%	77	177	85	103.0%	65	151	70	101.0%
Up 20%	95	224	101	131.7%	88	216	94	127.8%	87	195	93	112.5%	76	171	81	112.1%
Real Up 6%	3	21	7	0.4%	2	19	6	0.6%	9	28	12	0.0%	8	26	10	0.0%
Real Up 7%	4	23	8	2.2%	2	22	7	1.2%	10	29	12	0.1%	7	28	10	0.2%
Real Up 8%	4	24	9	2.8%	2	21	7	1.5%	10	33	13	0.0%	6	26	10	0.1%
Real Up 9%	5	29	10	5.2%	2	24	8	**5.4%**	10	33	13	0.5%	6	29	11	1.1%
Real Up 10%	7	37	13	**10.1%**	5	33	11	11.7%	11	37	15	2.9%	7	34	12	2.4%
Real Up 11%	12	49	19	26.0%	6	37	13	19.1%	13	42	18	**9.1%**	8	37	14	**6.2%**
Real Up 12%	17	59	24	33.7%	13	58	21	34.9%	19	54	23	19.4%	12	43	17	11.4%
Real Up 13%	22	86	33	48.3%	19	87	33	52.6%	25	67	29	27.3%	19	61	25	28.8%
Real Up 14%	27	107	42	57.4%	25	107	41	63.4%	32	91	39	40.9%	25	84	34	42.0%
Real Up 15%	39	131	55	72.2%	33	135	53	77.5%	38	114	48	51.5%	31	104	43	51.7%
Real Up 16%	52	150	66	88.7%	47	157	65	92.5%	45	130	57	63.0%	39	122	52	67.8%
Real Up 17%	73	174	81	102.8%	61	179	78	102.9%	57	149	68	73.7%	57	144	66	82.1%
Real Up 18%	83	191	89	110.3%	82	198	89	118.8%	72	172	81	89.6%	63	166	76	91.9%
Real Up 19%	100	219	105	124.9%	98	218	102	129.8%	83	189	91	97.8%	77	182	86	104.8%
Real Up 20%	111	235	112	130.9%	104	238	111	136.1%	99	209	103	108.2%	89	200	96	112.5%

Table A.1.: *Results obtained by the algorithm* BestInsert *controlling an immediate assignment (IA) or delayed assignment (DA) destination call system in **building A**. The cost considered by* BestInsert *only include the service cost, i. e., waiting time and travel time cost, either linearly or quadratic.*

scenario	IA, linear cost				DA, linear cost				IA, quadratic cost				DA, quadratic cost			
	$\alpha_{0.5}$	$\alpha_{0.9}$	$\emptyset$	reissues	$\alpha_{0.5}$	$\alpha_{0.9}$	$\emptyset$	reissues	$\alpha_{0.5}$	$\alpha_{0.9}$	$\emptyset$	reissues	$\alpha_{0.5}$	$\alpha_{0.9}$	$\emptyset$	reissues
Up 6%	2	13	5	0.0%	2	12	4	0.0%	8	20	9	0.0%	8	20	9	0.0%
Up 7%	4	20	7	0.0%	3	17	6	0.0%	9	22	10	0.0%	8	22	9	0.0%
Up 8%	5	24	9	0.1%	5	23	9	0.3%	10	26	12	0.0%	10	26	11	0.0%
Up 9%	5	24	9	0.0%	4	22	8	0.2%	11	28	13	0.0%	11	29	13	0.0%
Up 10%	6	28	10	0.9%	5	27	10	0.0%	12	31	13	0.0%	11	30	13	0.0%
Up 11%	7	34	12	2.6%	6	30	11	1.4%	12	32	14	0.1%	11	31	13	0.3%
Up 12%	9	36	13	3.6%	6	33	12	3.5%	14	35	15	0.7%	12	34	14	0.8%
Up 13%	11	39	15	**5.3%**	10	42	15	7.3%	13	38	16	0.9%	11	35	14	0.8%
Up 14%	14	45	18	12.0%	11	43	16	**10.4%**	14	39	17	3.5%	13	39	16	2.1%
Up 15%	16	51	21	19.0%	15	52	21	20.8%	17	45	20	**7.5%**	15	42	18	**6.7%**
Up 16%	21	67	28	29.4%	21	69	28	35.6%	21	53	24	16.3%	18	48	21	15.9%
Up 17%	33	95	41	53.2%	29	90	38	55.0%	31	72	34	30.0%	27	66	31	32.8%
Up 18%	40	122	51	66.6%	37	124	51	72.8%	37	91	42	44.4%	33	88	39	47.1%
Up 19%	49	140	61	82.2%	45	138	58	77.8%	40	105	47	51.4%	38	96	44	54.3%
Up 20%	66	173	76	95.6%	62	166	73	100.0%	53	128	61	70.1%	55	124	59	77.4%
Real Up 6%	4	18	7	0.0%	4	18	6	0.0%	9	23	10	0.0%	9	22	10	0.0%
Real Up 7%	5	22	9	0.0%	5	24	9	0.0%	11	25	12	0.0%	11	25	12	0.0%
Real Up 8%	5	20	8	0.1%	6	22	8	0.1%	11	27	13	0.0%	11	27	12	0.0%
Real Up 9%	6	26	10	0.0%	6	24	9	0.3%	12	32	14	0.0%	11	31	13	0.0%
Real Up 10%	7	29	11	0.6%	7	28	11	0.2%	13	34	15	0.0%	12	33	14	0.0%
Real Up 11%	9	32	13	2.4%	8	33	12	2.0%	15	39	17	0.1%	14	38	16	0.1%
Real Up 12%	9	34	13	4.4%	9	33	13	4.9%	15	40	17	0.5%	14	38	16	0.9%
Real Up 13%	12	41	17	**9.5%**	11	41	16	**10.3%**	17	44	20	3.0%	15	42	18	1.6%
Real Up 14%	14	42	18	13.6%	15	48	20	15.3%	18	47	21	**4.5%**	15	44	19	2.5%
Real Up 15%	18	61	25	24.5%	16	59	23	24.2%	24	57	27	11.7%	19	**50**	22	**10.0%**
Real Up 16%	28	86	37	43.9%	30	93	40	55.8%	27	70	32	20.6%	26	72	32	27.4%
Real Up 17%	32	103	43	54.4%	31	96	41	56.5%	31	81	37	29.1%	29	77	34	32.0%
Real Up 18%	42	134	58	70.2%	42	125	54	71.1%	40	105	47	40.3%	37	99	45	42.9%
Real Up 19%	53	154	68	83.1%	53	159	69	87.4%	51	122	58	55.8%	50	124	58	62.2%
Real Up 20%	59	169	74	91.6%	63	179	79	108.1%	53	131	61	59.0%	49	126	58	66.0%

Table A.2.: *Results obtained by the algorithm* BestInsert *controlling an immediate assignment (IA) or delayed assignment (DA) destination call system in **building B**. The cost considered by* BestInsert *only include the service cost, i. e., waiting time and travel time cost, either linearly or quadratic.*

scenario	IA, linear cost				DA, linear cost				IA, quadratic cost				DA, quadratic cost			
	$\alpha_{0.5}$	$\alpha_{0.9}$	$\emptyset$	reissues	$\alpha_{0.5}$	$\alpha_{0.9}$	$\emptyset$	reissues	$\alpha_{0.5}$	$\alpha_{0.9}$	$\emptyset$	reissues	$\alpha_{0.5}$	$\alpha_{0.9}$	$\emptyset$	reissues
Up 6%	2	18	6	0.0%	0	13	4	0.2%	8	24	10	0.0%	6	24	9	0.0%
Up 7%	3	23	7	0.0%	1	16	5	0.8%	8	28	11	0.0%	5	23	8	0.0%
Up 8%	3	24	8	0.0%	0	17	5	2.1%	8	28	11	0.0%	5	24	9	0.1%
Up 9%	3	24	8	0.0%	0	19	6	4.5%	8	29	11	0.0%	4	25	9	0.6%
Up 10%	5	32	11	0.0%	2	29	9	6.0%	8	32	12	0.0%	4	27	10	1.4%
Up 11%	7	35	13	0.0%	3	31	10	7.8%	10	33	13	0.0%	5	29	10	3.0%
Up 12%	10	41	15	0.0%	6	37	13	8.5%	11	35	14	0.0%	5	32	11	5.1%
Up 13%	16	**51**	21	0.0%	9	**45**	16	8.6%	15	42	18	0.0%	7	38	14	6.4%
Up 14%	26	67	30	2.0%	15	61	23	6.6%	23	**54**	25	0.1%	15	**50**	20	7.9%
Up 15%	35	86	39	11.2%	20	74	29	7.9%	29	71	33	2.4%	19	63	25	6.9%
Up 16%	47	111	50	22.2%	28	82	35	8.4%	39	89	43	11.7%	25	78	33	7.3%
Up 17%	53	134	59	34.4%	33	91	39	10.7%	49	117	54	24.9%	31	87	38	9.7%
Up 18%	64	158	72	48.6%	42	101	47	15.3%	61	144	66	39.8%	38	95	44	13.1%
Up 19%	75	186	86	64.6%	49	111	52	18.7%	72	167	80	57.3%	47	106	50	17.3%
Up 20%	82	214	99	77.8%	55	119	57	24.8%	79	191	91	67.9%	53	117	56	22.6%
Real Up 6%	3	21	7	0.0%	2	19	6	0.6%	9	28	12	0.0%	8	26	10	0.0%
Real Up 7%	4	23	8	0.0%	2	22	7	1.2%	10	29	12	0.0%	7	28	10	0.2%
Real Up 8%	4	24	9	0.0%	2	21	7	1.4%	10	33	13	0.0%	6	26	10	0.1%
Real Up 9%	5	27	10	0.0%	2	25	8	4.6%	10	33	13	0.0%	6	29	11	0.8%
Real Up 10%	5	30	11	0.0%	4	31	11	6.6%	11	36	14	0.0%	7	34	12	1.9%
Real Up 11%	9	39	15	0.0%	6	32	11	9.7%	12	40	16	0.0%	8	36	13	4.7%
Real Up 12%	16	46	20	0.0%	12	45	18	8.1%	17	49	21	0.0%	11	42	16	5.3%
Real Up 13%	23	**58**	27	0.3%	17	**57**	23	6.8%	21	**55**	25	0.0%	17	**53**	22	7.3%
Real Up 14%	28	73	34	3.0%	23	72	30	7.0%	30	70	33	1.2%	22	65	28	7.2%
Real Up 15%	38	90	43	10.1%	28	83	35	8.2%	37	84	40	6.8%	25	79	33	6.7%
Real Up 16%	47	117	54	20.1%	32	95	41	10.8%	43	99	47	13.2%	32	89	39	9.9%
Real Up 17%	63	148	70	36.7%	41	105	48	14.8%	57	132	61	27.0%	40	103	48	13.8%
Real Up 18%	67	167	77	44.8%	46	115	54	18.1%	66	160	74	39.4%	47	115	54	21.9%
Real Up 19%	75	189	88	54.9%	55	129	62	23.3%	71	170	79	46.2%	52	123	59	24.5%
Real Up 20%	84	206	98	64.3%	60	135	65	27.0%	81	192	93	58.6%	64	145	69	37.8%

Table A.3.: *Results obtained by the algorithm* BestInsert *controlling an immediate assignment (IA) or delayed assignment (DA) destination call system in **building A**. The cost considered by* BestInsert *includes the service cost plus a capacity violation penalty corresponding to 120 seconds waiting time.*

scenario	IA, linear cost				DA, linear cost				IA, quadratic cost				DA, quadratic cost			
	$\alpha_{0.5}$	$\alpha_{0.9}$	$\emptyset$	reissues	$\alpha_{0.5}$	$\alpha_{0.9}$	$\emptyset$	reissues	$\alpha_{0.5}$	$\alpha_{0.9}$	$\emptyset$	reissues	$\alpha_{0.5}$	$\alpha_{0.9}$	$\emptyset$	reissues
Up 6%	2	13	5	0.0%	2	12	4	0.0%	8	20	9	0.0%	8	20	9	0.0%
Up 7%	4	20	7	0.0%	3	17	6	0.0%	9	22	10	0.0%	8	22	9	0.0%
Up 8%	5	24	9	0.0%	5	23	9	0.3%	10	26	12	0.0%	10	26	11	0.0%
Up 9%	5	24	9	0.0%	4	22	8	0.2%	11	28	13	0.0%	11	29	13	0.0%
Up 10%	6	28	10	0.0%	5	27	10	0.0%	12	31	13	0.0%	11	30	13	0.0%
Up 11%	7	33	12	0.0%	6	30	11	0.7%	12	32	14	0.0%	11	31	13	0.3%
Up 12%	8	35	13	0.0%	7	34	12	1.4%	14	35	15	0.0%	12	33	14	0.4%
Up 13%	10	37	14	0.0%	9	40	14	2.5%	13	37	16	0.0%	11	35	14	0.7%
Up 14%	13	44	18	0.0%	10	41	15	1.8%	15	39	17	0.0%	12	39	16	1.1%
Up 15%	14	47	19	0.0%	13	47	18	2.3%	16	41	18	0.0%	14	42	17	2.8%
Up 16%	20	**57**	25	0.2%	17	**53**	22	3.4%	19	**48**	22	0.0%	17	**46**	20	2.7%
Up 17%	32	71	34	1.4%	23	64	28	2.2%	28	61	30	0.0%	23	59	27	2.0%
Up 18%	40	89	43	8.4%	27	77	33	2.6%	33	71	35	0.4%	26	68	30	1.7%
Up 19%	42	101	48	15.2%	33	81	37	3.0%	38	82	41	5.0%	29	73	34	2.5%
Up 20%	56	133	62	29.0%	37	88	41	3.5%	50	107	53	17.2%	35	83	40	2.8%
Real Up 6%	4	18	7	0.0%	4	18	6	0.0%	9	23	10	0.0%	9	22	10	0.0%
Real Up 7%	5	22	9	0.0%	5	24	9	0.0%	11	25	12	0.0%	11	25	12	0.0%
Real Up 8%	5	20	8	0.0%	6	22	8	0.1%	11	27	13	0.0%	11	27	12	0.0%
Real Up 9%	6	26	10	0.0%	6	24	9	0.0%	12	32	14	0.0%	11	31	13	0.0%
Real Up 10%	7	29	11	0.0%	7	28	11	0.2%	13	34	15	0.0%	12	33	14	0.0%
Real Up 11%	9	32	13	0.0%	8	33	12	0.5%	15	38	17	0.0%	14	38	16	0.1%
Real Up 12%	8	33	13	0.0%	8	35	13	1.7%	15	40	17	0.0%	14	38	16	0.7%
Real Up 13%	13	41	17	0.0%	11	39	16	2.8%	17	43	19	0.0%	15	42	18	1.0%
Real Up 14%	13	41	17	0.0%	12	43	17	2.6%	18	47	20	0.0%	15	44	19	1.2%
Real Up 15%	17	**48**	22	0.0%	15	**51**	20	2.8%	22	**53**	25	0.0%	18	**46**	21	2.2%
Real Up 16%	26	67	32	1.0%	23	67	29	2.5%	26	61	29	0.0%	23	61	27	1.9%
Real Up 17%	27	73	34	1.6%	24	68	30	2.8%	29	69	32	0.6%	25	63	29	2.6%
Real Up 18%	35	90	43	5.7%	30	78	36	2.5%	36	81	40	2.4%	30	79	36	3.1%
Real Up 19%	46	114	55	17.8%	38	93	43	4.4%	47	104	51	12.0%	38	92	43	3.9%
Real Up 20%	48	124	58	20.5%	36	92	43	3.5%	46	103	51	11.9%	36	90	42	4.5%

Table A.4.: *Results obtained by the algorithm* BestInsert *controlling an immediate assignment (IA) or delayed assignment (DA) destination call system in* ***building B****. The cost considered by* BestInsert *includes the service cost plus a capacity violation penalty corresponding to 120 seconds waiting time.*

scenario	IA, linear cost				DA, linear cost				IA, quadratic cost				DA, quadratic cost			
	$\alpha_{0.5}$	$\alpha_{0.9}$	$\emptyset$	reissues	$\alpha_{0.5}$	$\alpha_{0.9}$	$\emptyset$	reissues	$\alpha_{0.5}$	$\alpha_{0.9}$	$\emptyset$	reissues	$\alpha_{0.5}$	$\alpha_{0.9}$	$\emptyset$	reissues
Up 6%	0	11	3	0.0%	0	7	2	0.1%	0	11	3	0.0%	0	7	2	0.2%
Up 7%	0	14	3	0.0%	0	9	2	1.0%	0	14	4	0.0%	0	8	2	0.5%
Up 8%	0	19	5	0.0%	0	11	3	2.0%	0	16	4	0.0%	0	15	4	2.2%
Up 9%	0	19	5	0.0%	0	11	3	3.5%	0	24	7	0.0%	0	14	4	3.6%
Up 10%	0	35	11	0.0%	0	16	4	7.4%	0	34	11	0.0%	0	19	5	6.4%
Up 11%	1	**51**	17	0.6%	0	19	6	**10.7%**	4	43	16	0.1%	0	27	8	**8.7%**
Up 12%	0	66	22	1.0%	2	31	10	12.6%	10	**58**	21	0.5%	2	34	11	12.2%
Up 13%	18	92	34	5.5%	8	41	15	13.0%	26	83	32	3.3%	8	44	16	12.0%
Up 14%	28	101	40	14.5%	15	56	22	9.8%	35	95	39	8.2%	14	58	22	9.6%
Up 15%	41	115	49	23.1%	21	68	28	11.0%	47	103	48	14.4%	20	68	27	10.5%
Up 16%	51	141	58	33.5%	28	83	35	13.4%	52	118	55	21.2%	28	84	35	12.6%
Up 17%	60	156	66	42.2%	35	91	40	18.0%	60	137	64	29.1%	35	91	40	14.8%
Up 18%	72	180	78	54.7%	42	100	46	22.1%	67	159	72	36.5%	43	99	46	18.3%
Up 19%	83	201	89	65.5%	53	113	54	29.3%	83	197	87	49.5%	53	115	55	27.8%
Up 20%	97	224	100	73.1%	60	121	59	32.8%	94	218	100	56.9%	61	129	62	35.1%
Real Up 6%	0	20	5	0.0%	0	16	5	0.2%	0	22	6	0.0%	0	20	5	0.0%
Real Up 7%	0	23	7	0.0%	0	17	5	0.5%	0	25	8	0.0%	0	20	6	0.5%
Real Up 8%	0	24	8	0.0%	0	20	6	1.3%	0	26	9	0.0%	0	22	6	1.0%
Real Up 9%	2	37	13	0.0%	0	23	8	3.1%	3	37	13	0.0%	0	28	8	2.6%
Real Up 10%	2	**41**	14	0.0%	0	28	9	3.8%	7	**39**	15	0.0%	0	30	10	3.1%
Real Up 11%	14	69	27	0.3%	3	35	13	6.8%	17	60	24	0.1%	5	39	14	5.6%
Real Up 12%	18	80	30	1.2%	6	44	16	8.8%	26	79	32	0.4%	8	48	17	8.4%
Real Up 13%	24	91	39	2.5%	10	**49**	19	10.3%	30	90	37	1.0%	10	**53**	20	9.2%
Real Up 14%	35	106	47	9.6%	16	67	26	8.9%	42	102	48	6.2%	18	70	27	8.6%
Real Up 15%	42	119	53	15.3%	21	81	32	11.6%	43	107	51	8.5%	26	83	34	9.3%
Real Up 16%	51	141	62	24.1%	29	93	38	13.8%	56	122	64	14.6%	28	94	39	11.7%
Real Up 17%	57	150	68	30.2%	36	102	44	16.0%	61	141	68	20.2%	40	105	46	15.3%
Real Up 18%	68	175	79	42.5%	43	115	51	19.6%	73	169	81	28.7%	48	123	55	22.9%
Real Up 19%	75	187	86	46.0%	48	121	55	23.4%	76	189	87	34.0%	55	133	61	26.9%
Real Up 20%	85	206	94	54.9%	57	142	65	27.9%	90	214	98	41.5%	61	149	68	30.2%

Table A.5.: *Results obtained by the algorithm* BestInsert *controlling an immediate assignment (IA) or delayed assignment (DA) destination call system in* ***building A****. The cost considered by* BestInsert *includes the service cost, the capacity violation penalty as before plus the RTT penalty.*

scenario	IA, linear cost				DA, linear cost				IA, quadratic cost				DA, quadratic cost			
	$\alpha_{0.5}$	$\alpha_{0.9}$	$\emptyset$	reissues	$\alpha_{0.5}$	$\alpha_{0.9}$	$\emptyset$	reissues	$\alpha_{0.5}$	$\alpha_{0.9}$	$\emptyset$	reissues	$\alpha_{0.5}$	$\alpha_{0.9}$	$\emptyset$	reissues
Up 6%	0	10	3	0.0%	0	9	2	0.0%	0	12	3	0.0%	0	11	3	0.0%
Up 7%	0	10	3	0.0%	0	10	3	0.0%	0	12	3	0.0%	0	12	3	0.0%
Up 8%	0	14	4	0.0%	0	16	4	0.0%	0	14	4	0.0%	0	15	4	0.0%
Up 9%	1	19	6	0.0%	1	18	6	0.0%	1	19	6	0.0%	0	19	6	0.0%
Up 10%	3	24	8	0.0%	2	20	7	0.1%	2	24	8	0.0%	2	26	8	0.1%
Up 11%	5	32	11	0.0%	4	24	8	0.2%	6	30	11	0.0%	4	26	9	0.2%
Up 12%	10	36	14	0.0%	5	27	10	0.8%	11	34	14	0.0%	7	30	11	0.5%
Up 13%	12	39	16	0.0%	6	32	12	1.8%	13	40	16	0.0%	8	33	12	0.6%
Up 14%	16	**48**	20	0.0%	8	37	13	1.5%	17	45	20	0.0%	11	36	14	2.1%
Up 15%	26	62	27	0.1%	13	42	17	3.3%	24	**54**	24	0.0%	13	38	15	2.7%
Up 16%	31	72	32	1.2%	16	48	20	3.8%	28	62	29	1.4%	16	48	20	2.8%
Up 17%	39	84	40	4.5%	22	**55**	24	3.6%	36	74	35	1.5%	23	**56**	25	3.6%
Up 18%	45	95	46	9.2%	26	65	29	3.5%	44	91	44	5.3%	27	66	30	3.5%
Up 19%	42	97	47	10.6%	29	72	33	4.1%	45	95	47	9.4%	30	71	32	2.8%
Up 20%	57	116	57	18.8%	35	83	39	6.7%	56	109	56	15.4%	37	81	39	3.9%
Real Up 6%	0	15	4	0.0%	0	16	4	0.0%	0	16	4	0.0%	0	15	4	0.0%
Real Up 7%	0	19	6	0.0%	0	19	6	0.0%	0	19	6	0.0%	0	20	6	0.0%
Real Up 8%	2	20	6	0.0%	2	20	6	0.0%	1	21	7	0.0%	1	21	7	0.0%
Real Up 9%	4	24	9	0.0%	3	23	8	0.0%	4	27	10	0.0%	4	24	9	0.0%
Real Up 10%	7	32	12	0.0%	4	26	10	0.0%	7	32	12	0.0%	5	30	11	0.2%
Real Up 11%	11	35	15	0.0%	8	33	13	0.3%	11	37	15	0.0%	8	33	13	0.3%
Real Up 12%	16	44	21	0.0%	11	41	16	0.2%	17	43	19	0.0%	12	38	16	0.3%
Real Up 13%	20	**56**	25	0.0%	11	44	18	0.8%	19	55	24	0.0%	13	43	18	0.8%
Real Up 14%	25	63	30	0.0%	14	43	19	1.2%	23	**55**	26	0.0%	16	46	20	1.2%
Real Up 15%	30	76	36	0.2%	17	**58**	25	1.7%	28	70	32	0.0%	19	**55**	24	2.1%
Real Up 16%	36	86	44	2.3%	22	64	28	2.8%	35	82	40	0.4%	23	66	29	2.5%
Real Up 17%	39	90	44	1.6%	24	67	30	3.2%	43	89	45	1.0%	27	67	31	3.5%
Real Up 18%	43	104	52	5.2%	26	85	36	3.1%	46	95	52	3.0%	30	85	37	3.5%
Real Up 19%	51	105	57	8.7%	33	90	41	5.3%	50	108	58	7.4%	36	92	42	5.4%
Real Up 20%	50	115	58	11.0%	34	93	42	5.6%	56	120	63	10.6%	38	90	42	4.9%

Table A.6.: *Results obtained by the algorithm* BestInsert *controlling an immediate assignment (IA) or delayed assignment (DA) destination call system in* ***building B****. The cost considered by* BestInsert *includes the service cost, the capacity violation penalty as before plus the RTT penalty.*

scenario	IA, linear cost				DA, linear cost				IA, quadratic cost				DA, quadratic cost			
	$\alpha_{0.5}$	$\alpha_{0.9}$	$\emptyset$	reissues	$\alpha_{0.5}$	$\alpha_{0.9}$	$\emptyset$	reissues	$\alpha_{0.5}$	$\alpha_{0.9}$	$\emptyset$	reissues	$\alpha_{0.5}$	$\alpha_{0.9}$	$\emptyset$	reissues
Up 6%	6	27	10	0.0%	5	23	8	0.0%	9	25	11	0.0%	7	25	10	0.0%
Up 7%	7	31	11	0.0%	5	26	9	0.0%	9	28	11	0.0%	7	25	10	0.0%
Up 8%	7	33	12	0.0%	5	28	10	0.2%	11	33	14	0.0%	7	27	10	0.0%
Up 9%	7	35	12	0.0%	4	27	9	0.5%	10	34	14	0.0%	6	27	10	0.1%
Up 10%	8	40	14	0.0%	6	35	12	0.6%	11	36	14	0.0%	7	30	11	0.5%
Up 11%	9	36	14	0.0%	5	33	11	2.1%	10	35	14	0.0%	6	31	11	1.2%
Up 12%	8	39	14	0.0%	6	34	12	4.3%	9	34	13	0.0%	6	32	12	3.2%
Up 13%	9	38	14	0.0%	7	38	13	5.3%	14	44	18	0.0%	7	35	13	4.3%
Up 14%	14	**45**	19	0.1%	13	47	19	6.4%	18	**51**	22	0.0%	13	43	17	6.3%
Up 15%	23	63	28	2.5%	17	**58**	23	6.2%	25	61	28	1.5%	17	**58**	23	7.0%
Up 16%	34	77	37	6.4%	23	73	30	6.5%	33	76	36	4.8%	23	73	30	6.6%
Up 17%	40	90	43	13.6%	29	84	36	8.0%	40	89	43	12.3%	28	82	35	8.0%
Up 18%	49	112	53	23.9%	37	97	44	11.6%	51	116	55	25.9%	37	93	43	12.7%
Up 19%	61	137	64	36.5%	42	103	48	14.7%	61	139	66	37.8%	45	105	49	16.0%
Up 20%	67	156	74	49.3%	47	111	52	17.7%	67	154	74	48.3%	51	112	54	21.1%
Real Up 6%	8	30	12	0.0%	7	26	10	0.0%	12	31	13	0.0%	10	28	12	0.0%
Real Up 7%	9	29	12	0.0%	8	28	11	0.0%	11	31	13	0.0%	10	30	12	0.0%
Real Up 8%	9	35	13	0.0%	7	29	11	0.2%	12	34	14	0.0%	9	32	12	0.0%
Real Up 9%	10	34	13	0.0%	8	32	12	0.3%	14	37	16	0.0%	9	33	13	0.1%
Real Up 10%	10	35	14	0.0%	8	33	12	0.9%	13	40	17	0.0%	9	35	13	0.5%
Real Up 11%	11	39	15	0.0%	9	34	13	3.1%	14	44	18	0.0%	10	38	15	1.5%
Real Up 12%	13	42	17	0.0%	12	40	16	4.5%	16	47	20	0.0%	12	42	16	3.2%
Real Up 13%	18	**49**	22	0.0%	14	46	19	5.8%	19	**53**	23	0.0%	15	**46**	19	5.3%
Real Up 14%	24	60	28	0.2%	20	**58**	25	5.3%	26	65	30	0.5%	20	60	25	6.2%
Real Up 15%	29	71	34	2.2%	24	72	31	6.0%	34	76	37	3.4%	25	74	31	6.5%
Real Up 16%	35	83	40	6.7%	28	84	36	8.0%	37	87	42	7.7%	29	85	37	7.9%
Real Up 17%	45	102	51	15.0%	36	95	43	10.8%	49	109	53	17.3%	38	95	44	10.8%
Real Up 18%	56	127	61	27.0%	44	109	51	15.2%	57	135	64	29.5%	44	109	51	19.1%
Real Up 19%	62	144	70	34.6%	47	117	55	17.8%	63	148	69	33.7%	53	121	58	23.5%
Real Up 20%	70	165	79	43.6%	54	128	61	22.2%	73	168	82	45.7%	60	138	65	32.9%

Table A.7.: *Results obtained by the algorithm* BestInsert *controlling an immediate assignment (IA) or delayed assignment (DA) destination call system in* ***building A****. The cost considered by* BestInsert *includes the service cost and the capacity violation penalty as before. In addition, the number of destination floors served from the main floor is limited to 3.*

scenario	IA, linear cost				DA, linear cost				IA, quadratic cost				DA, quadratic cost			
	$\alpha_{0.5}$	$\alpha_{0.9}$	$\emptyset$	reissues	$\alpha_{0.5}$	$\alpha_{0.9}$	$\emptyset$	reissues	$\alpha_{0.5}$	$\alpha_{0.9}$	$\emptyset$	reissues	$\alpha_{0.5}$	$\alpha_{0.9}$	$\emptyset$	reissues
Up 6%	2	13	5	0.0%	2	12	4	0.0%	8	20	9	0.0%	8	20	9	0.0%
Up 7%	5	21	8	0.0%	4	20	7	0.0%	9	22	10	0.0%	8	22	9	0.0%
Up 8%	7	27	10	0.0%	6	25	10	0.0%	10	26	12	0.0%	10	26	11	0.0%
Up 9%	7	26	10	0.0%	6	26	9	0.0%	11	29	13	0.0%	11	30	13	0.0%
Up 10%	8	31	12	0.0%	8	31	12	0.1%	12	32	14	0.0%	11	31	13	0.0%
Up 11%	11	36	14	0.0%	10	36	14	0.3%	13	34	15	0.0%	12	32	14	0.1%
Up 12%	11	42	16	0.0%	10	40	15	0.1%	15	37	16	0.0%	13	35	15	0.2%
Up 13%	14	42	18	0.0%	12	42	16	0.5%	15	38	17	0.0%	13	36	15	0.7%
Up 14%	15	46	19	0.0%	13	43	18	0.8%	18	43	19	0.0%	15	41	18	0.6%
Up 15%	17	50	21	0.0%	15	47	20	0.9%	19	48	21	0.0%	17	45	20	1.2%
Up 16%	22	**55**	25	0.0%	18	49	22	1.7%	21	**50**	23	0.0%	17	46	20	1.9%
Up 17%	27	62	30	0.5%	24	**58**	27	1.4%	26	59	28	0.3%	22	**55**	25	2.1%
Up 18%	30	67	32	1.2%	27	64	29	1.6%	29	64	31	0.6%	25	62	28	1.9%
Up 19%	34	73	36	3.2%	30	72	33	1.3%	33	72	35	2.6%	28	70	32	1.7%
Up 20%	47	92	48	12.6%	36	82	39	1.6%	42	86	44	7.7%	36	80	39	2.5%
Real Up 6%	4	19	7	0.0%	4	19	7	0.0%	9	23	10	0.0%	9	22	10	0.0%
Real Up 7%	6	23	9	0.0%	6	24	9	0.0%	11	25	12	0.0%	11	25	12	0.0%
Real Up 8%	7	25	10	0.0%	7	24	9	0.0%	12	28	13	0.0%	11	27	12	0.0%
Real Up 9%	8	30	12	0.0%	7	29	11	0.0%	12	32	14	0.0%	12	31	14	0.0%
Real Up 10%	9	32	13	0.0%	9	32	13	0.1%	14	34	16	0.0%	13	34	15	0.0%
Real Up 11%	13	36	16	0.0%	12	37	15	0.1%	17	40	18	0.0%	15	39	17	0.0%
Real Up 12%	14	39	17	0.0%	13	39	16	0.4%	17	42	19	0.0%	16	42	18	0.3%
Real Up 13%	17	44	20	0.0%	16	47	20	0.7%	21	49	23	0.0%	18	45	21	0.5%
Real Up 14%	17	44	20	0.0%	17	45	20	0.6%	20	47	22	0.0%	17	47	20	0.6%
Real Up 15%	20	**49**	23	0.0%	18	49	21	1.3%	23	**54**	26	0.0%	19	**51**	22	1.7%
Real Up 16%	25	59	29	0.1%	24	**58**	27	1.6%	26	61	29	0.1%	24	60	27	1.5%
Real Up 17%	26	63	30	0.2%	26	61	29	1.5%	26	64	29	0.0%	26	62	29	2.3%
Real Up 18%	31	71	35	1.3%	28	71	33	1.9%	34	75	37	1.4%	30	73	34	1.6%
Real Up 19%	41	88	45	5.7%	35	84	40	2.5%	44	91	46	6.1%	36	84	40	3.4%
Real Up 20%	39	86	45	5.7%	35	85	40	4.0%	43	89	46	6.6%	37	83	40	4.0%

Table A.8.: *Results obtained by the algorithm* BestInsert *controlling an immediate assignment (IA) or delayed assignment (DA) destination call system in **building B**. The cost considered by* BestInsert *includes the service cost and the capacity violation penalty as before. In addition, the number of destination floors served from the main floor is limited to 5.*

scenario	IA, linear cost				DA, linear cost				IA, quadratic cost				DA, quadratic cost			
	$\alpha_{0.5}$	$\alpha_{0.9}$	$\emptyset$	reissues	$\alpha_{0.5}$	$\alpha_{0.9}$	$\emptyset$	reissues	$\alpha_{0.5}$	$\alpha_{0.9}$	$\emptyset$	reissues	$\alpha_{0.5}$	$\alpha_{0.9}$	$\emptyset$	reissues
Up 12%	10	41	15	0.0%	5	33	11	10.8%	11	35	14	0.0%	5	29	10	5.6%
Up 13%	16	**50**	20	0.0%	8	**42**	15	8.5%	15	42	18	0.0%	8	34	13	8.1%
Up 14%	26	67	30	2.0%	13	59	22	6.2%	23	**55**	25	0.1%	14	**45**	19	7.4%
Up 15%	35	86	39	11.2%	17	74	28	4.9%	29	71	33	2.4%	19	62	25	6.1%
Up 16%	47	111	50	22.2%	25	86	35	6.9%	39	89	43	11.7%	26	76	33	6.1%
Real Up 12%	16	43	20	0.0%	9	37	14	9.5%	17	49	21	0.0%	9	35	13	7.6%
Real Up 13%	22	**58**	27	0.2%	12	**50**	20	6.8%	21	**54**	24	0.0%	12	42	17	8.6%
Real Up 14%	27	71	33	2.6%	18	67	27	6.8%	30	69	33	1.1%	18	**57**	24	6.0%
Real Up 15%	36	88	42	9.7%	21	86	34	10.2%	36	84	40	6.6%	22	67	28	7.3%
Real Up 16%	47	114	54	20.3%	26	94	39	10.9%	43	99	47	13.0%	29	83	36	10.1%

Table A.9.: *Results obtained by the algorithm* ExactReplan *controlling an immediate assignment (IA) or delayed assignment (DA) destination call system in* ***building A****. The cost considered by* BestInsert *includes the service cost plus a capacity violation penalty corresponding to 120 seconds waiting time.*

scenario	IA, linear cost				DA, linear cost				IA, quadratic cost				DA, quadratic cost			
	$\alpha_{0.5}$	$\alpha_{0.9}$	$\emptyset$	reissues	$\alpha_{0.5}$	$\alpha_{0.9}$	$\emptyset$	reissues	$\alpha_{0.5}$	$\alpha_{0.9}$	$\emptyset$	reissues	$\alpha_{0.5}$	$\alpha_{0.9}$	$\emptyset$	reissues
Up 14%	13	44	18	0.0%	7	34	13	2.1%	15	39	17	0.0%	12	34	14	1.0%
Up 15%	14	47	19	0.0%	10	43	16	3.3%	16	41	18	0.0%	12	36	16	2.5%
Up 16%	20	**57**	25	0.2%	15	**53**	21	3.4%	19	**48**	22	0.0%	15	42	18	2.8%
Up 17%	32	71	34	1.4%	21	67	28	3.5%	28	61	30	0.0%	21	**53**	24	2.8%
Up 18%	40	89	43	8.4%	22	80	33	5.1%	33	71	35	0.4%	24	66	29	4.8%
Real Up 14%	13	40	17	0.0%	8	34	14	3.1%	17	46	20	0.0%	13	35	16	1.6%
Real Up 15%	17	**48**	22	0.0%	12	**43**	19	3.8%	22	**52**	24	0.0%	14	39	17	2.9%
Real Up 16%	25	67	32	1.2%	18	64	27	4.9%	25	61	28	0.0%	19	53	24	4.4%
Real Up 17%	27	70	33	1.3%	18	69	29	7.9%	28	69	32	0.6%	21	**58**	26	5.1%
Real Up 18%	35	87	42	6.3%	25	92	38	12.9%	34	82	39	2.3%	25	76	32	6.5%

Table A.10.: *Results obtained by the algorithm* ExactReplan *controlling an immediate assignment (IA) or delayed assignment (DA) destination call system in* ***building B****. The cost considered by* BestInsert *includes the service cost plus a capacity violation penalty corresponding to 120 seconds waiting time.*

(a) Building A.

scenario	conventional system (CGC)			
	$\alpha_{0.5}$	$\alpha_{0.9}$	$\emptyset$	reissues
Up 6%	0	3	1	**9.0%**
Up 7%	0	11	3	29.1%
Up 8%	3	22	7	63.8%
Up 9%	20	43	21	132.9%
Up 10%	39	80	40	222.1%
Up 11%	53	111	56	291.3%
Up 12%	75	146	76	378.2%
Up 13%	97	181	97	474.5%
Up 14%	116	215	115	551.8%
Up 15%	129	251	133	632.7%
Up 16%	155	288	156	731.6%
Up 17%	174	323	175	814.3%
Up 18%	192	356	193	897.6%
Up 19%	214	392	212	980.4%
Up 20%	233	427	232	1066.2%
Real Up 6%	0	15	4	**14.5%**
Real Up 7%	0	16	5	28.3%
Real Up 8%	4	26	9	62.6%
Real Up 9%	15	50	20	111.8%
Real Up 10%	18	71	27	140.9%
Real Up 11%	44	108	48	260.7%
Real Up 12%	67	148	71	357.5%
Real Up 13%	85	178	87	442.5%
Real Up 14%	95	208	102	499.8%
Real Up 15%	122	247	126	632.3%
Real Up 16%	138	271	138	657.6%
Real Up 17%	144	298	151	688.8%
Real Up 18%	171	337	173	804.9%
Real Up 19%	191	362	185	854.2%
Real Up 20%	212	400	205	969.4%

(b) Building B.

scenario	conventional system (CGC)			
	$\alpha_{0.5}$	$\alpha_{0.9}$	$\emptyset$	reissues
Up 6%	0	7	2	1.1%
Up 7%	0	9	2	2.5%
Up 8%	0	12	3	6.0%
Up 9%	0	14	4	**10.3%**
Up 10%	0	21	6	22.9%
Up 11%	2	23	8	34.4%
Up 12%	6	31	11	47.5%
Up 13%	9	41	15	66.9%
Up 14%	19	57	24	104.0%
Up 15%	30	76	34	141.5%
Up 16%	42	101	44	181.5%
Up 17%	55	116	56	236.2%
Up 18%	62	139	67	274.1%
Up 19%	70	163	77	311.5%
Up 20%	83	182	90	370.4%
Real Up 6%	0	15	4	0.8%
Real Up 7%	0	16	5	1.9%
Real Up 8%	0	18	6	6.3%
Real Up 9%	1	20	6	**9.8%**
Real Up 10%	3	25	8	19.2%
Real Up 11%	6	30	11	26.3%
Real Up 12%	13	41	16	53.4%
Real Up 13%	18	60	24	76.2%
Real Up 14%	23	60	27	98.5%
Real Up 15%	36	92	42	159.0%
Real Up 16%	55	120	58	220.0%
Real Up 17%	60	134	63	235.9%
Real Up 18%	61	150	70	234.9%
Real Up 19%	77	178	87	314.9%
Real Up 20%	93	194	96	372.0%

Table A.11.: *Results obtained by the algorithm* CGC *controlling a conventional system.*

A.2. Exact Markov chain simulation results for bin packing

Table A.12 shows the results of the exact simulation of the Markov chains of the BBF_k and NF_k algorithms for various bin packing parameters referenced by Observation 4.3.6 on page 108. The simulation covers the first 200 items, starting from all empty bins. For all parameters, the number of bins used by BBF_k is stochastically dominated by that needed by NF_k.

The Markov chain simulations were implemented using the GNU Multiple Precision Library [GMP06]. The simulations ran on a system with an Intel Core 2 X9650 CPU with 3.0 GHz and 8 GB of RAM with 32 bit code, except for the parameters marked with *, which ran on the same system but used 64 bit code due to the high memory requirements.

Table A.12.: *Results of the exact simulation of the Markov chains of the* BBF_k *and* NF_k *algorithms for various bin packing parameters.*

m	B	u	$\mathbb{E}[\mathrm{BBF}_k]$	$\mathbb{E}[\mathrm{NF}_k]$	time [s]
2	5	4	112.79	116.25	4.08
2	6	4	91.22	94.03	5.11
2	6	5	112.62	117.49	26.50
2	7	4	76.50	77.41	5.84
2	7	5	96.56	99.34	32.55
2	7	6	114.08	118.60	38.24
2	8	4	64.80	65.78	3.21
2	8	5	81.70	83.75	35.28
2	8	6	99.17	102.96	46.34
2	8	7	114.12	119.28	83.82
2	9	4	57.86	58.38	3.83
2	9	5	71.08	72.08	36.76
2	9	6	86.91	89.14	53.55
2	9	7	102.08	105.82	98.54
2	9	8	114.91	119.89	31.05
2	10	4	51.64	52.06	4.44
2	10	5	62.70	63.63	22.74
2	10	6	75.93	77.57	57.05
2	10	7	90.29	93.30	111.85
2	10	8	103.66	107.94	36.64
2	10	9	114.99	120.31	197.80
2	11	4	46.73	47.03	4.79
2	11	5	57.09	57.72	25.47
2	11	6	67.82	68.82	59.16
2	11	7	80.64	82.50	122.52
2	11	8	93.61	96.74	41.84

Table A.12.: *Results of the exact simulation of the Markov chains of the* BBF_k *and* NF_k *algorithms (continued).*

m	B	u	$\mathbb{E}[\mathrm{BBF}_k]$	$\mathbb{E}[\mathrm{NF}_k]$	time [s]
2	11	9	105.41	109.68	225.86
2	11	10	115.47	120.70	232.29
2	12	4	42.70	42.94	3.40
2	12	5	52.09	52.53	28.30
2	12	6	61.28	62.20	34.75
2	12	7	72.12	73.54	124.72
2	12	8	84.13	86.59	46.58
2	12	9	95.87	99.48	254.00
2	12	10	106.46	111.05	264.71
2	12	11	115.55	120.98	387.90
2	13	4	39.42	39.57	3.80
2	13	5	47.77	48.13	29.60
2	13	6	56.63	57.29	38.94
2	13	7	65.58	66.60	125.54
2	13	8	76.29	77.92	51.22
2	13	9	87.44	90.11	280.91
2	13	10	98.08	101.78	297.00
2	13	11	107.61	112.22	435.56
2	13	12	115.86	121.24	308.55
2	14	4	36.53	36.66	4.18
2	14	5	44.20	44.49	29.93
2	14	6	52.37	52.86	42.62
2	14	7	60.28	61.19	79.12
2	14	8	69.41	70.71	54.70
2	14	9	79.66	81.74	299.38
2	14	10	89.97	93.05	328.35
2	14	11	99.67	103.68	482.00
2	14	12	108.37	113.17	344.69
2	14	13	115.94	121.44	680.79
3	5	4	109.69	112.44	10.07
3	6	4	87.75	90.15	12.96
3	6	5	108.85	113.32	82.36
3	7	4	74.18	74.92	14.46
3	7	5	93.03	95.42	105.00
3	7	6	110.52	114.54	138.12
3	8	4	63.64	64.39	8.68
3	8	5	78.58	80.35	113.18

Table A.12.: *Results of the exact simulation of the Markov chains of the* BBF_k *and* NF_k *algorithms (continued).*

m	B	u	$\mathbb{E}[\mathrm{BBF}_k]$	$\mathbb{E}[\mathrm{NF}_k]$	time [s]
3	8	6	95.28	98.82	176.50
3	8	7	110.24	115.11	347.47
3	9	4	56.71	57.15	11.02
3	9	5	68.93	69.85	115.88
3	9	6	83.46	85.43	206.44
3	9	7	98.27	101.72	432.74
3	9	8	111.16	115.77	149.86
3	10	4	50.76	51.09	12.80
3	10	5	61.42	62.19	76.10
3	10	6	73.09	74.58	217.44
3	10	7	86.47	89.31	505.02
3	10	8	99.62	103.77	187.07
3	10	9	111.02	116.14	1023.46
3	11	4	46.13	46.33	13.45
3	11	5	55.84	56.41	87.96
3	11	6	65.86	66.79	226.59
3	11	7	77.39	79.08	553.13
3	11	8	89.80	92.72	222.68
3	11	9	101.52	105.56	1234.14
3	11	10	111.63	116.56	1320.20
3	12	4	42.26	42.42	9.84
3	12	5	51.01	51.42	99.95
3	12	6	59.92	60.74	141.29
3	12	7	69.57	70.88	552.39
3	12	8	80.50	82.84	253.50
3	12	9	91.86	95.37	1439.81
3	12	10	102.42	106.90	1588.69
3	12	11	111.57	116.82	2392.22
3	13	4	39.03	39.14	11.21
3	13	5	46.96	47.27	102.05
3	13	6	55.32	55.94	162.60
3	13	7	63.72	64.69	560.34
3	13	8	73.29	74.79	275.41
3	13	9	83.71	86.23	1628.79
3	13	10	94.12	97.68	1849.33
3	13	11	103.66	108.09	2823.37
3	13	12	111.96	117.09	2071.51

Table A.12.: *Results of the exact simulation of the Markov chains of the* BBF_k *and* NF_k *algorithms (continued).*

m	B	u	$\mathbb{E}[\mathrm{BBF}_k]$	$\mathbb{E}[\mathrm{NF}_k]$	time [s]
3	14	4	36.24	36.32	12.31
3	14	5	43.58	43.81	101.98
3	14	6	51.19	51.67	182.02
3	14	7	58.88	59.71	372.47
3	14	8	67.05	68.28	289.10
3	14	9	76.28	78.25	1731.99
3	14	10	86.05	89.05	2110.15
3	14	11	95.61	99.55	3260.20
3	14	12	104.32	109.03	2434.24
3	14	13	111.95	117.29	4876.30
4	5	4	108.00	110.35	24.72
4	6	4	86.14	88.05	31.91
4	6	5	106.96	110.90	256.07
4	7	4	73.18	73.75	35.18
4	7	5	91.17	93.22	329.24
4	7	6	108.68	112.23	491.76
4	8	4	63.27	63.78	24.21
4	8	5	77.24	78.64	352.73
4	8	6	93.25	96.43	653.11
4	8	7	108.26	112.69	1433.43
4	9	4	56.33	56.64	32.29
4	9	5	68.09	68.84	361.48
4	9	6	81.76	83.42	754.80
4	9	7	96.26	99.36	1866.02
4	9	8	109.23	113.39	705.25
4	10	4	50.57	50.75	37.39
4	10	5	60.95	61.54	253.25
4	10	6	71.91	73.15	786.89
4	10	7	84.53	87.06	2201.15
4	10	8	97.49	101.34	929.99
4	10	9	108.97	113.72	5234.23
4	11	4	46.00	46.11	39.21
4	11	5	55.41	55.84	304.10
4	11	6	65.09	65.88	860.44
4	11	7	75.86	77.30	2376.93
4	11	8	87.80	90.41	1126.86
4	11	9	99.46	103.16	6656.04

Table A.12.: *Results of the exact simulation of the Markov chains of the* BBF_k *and* NF_k *algorithms (continued).*

m	B	u	$\mathbb{E}[\mathrm{BBF}_k]$	$\mathbb{E}[\mathrm{NF}_k]$	time [s]
4	11	10	109.65	114.16	7390.48
4	12	4	42.18	42.25	28.73
4	12	5	50.70	50.98	346.29
4	12	6	59.41	60.07	572.85
4	12	7	68.54	69.64	2358.11
4	12	8	78.73	80.79	1278.68
4	12	9	89.73	92.98	7984.21
4	12	10	100.27	104.47	9426.20
4	12	11	109.50	114.39	14622.80
4	13	4	38.97	39.01	32.86
4	13	5	46.77	46.96	346.00
4	13	6	54.84	55.34	680.93
4	13	7	63.00	63.83	2447.53
4	13	8	71.95	73.24	1363.04
4	13	9	81.81	84.05	9081.97
4	13	10	92.02	95.29	11334.40
4	13	11	101.58	105.68	18138.90
4	13	12	109.95	114.69	13770.70
4	14	4	36.21	36.24	36.12
4	14	5	43.43	43.58	344.79
4	14	6	50.81	51.18	774.04
4	14	7	58.35	59.03	1744.76
4	14	8	66.12	67.17	1460.87
4	14	9	74.70	76.42	9627.75
4	14	10	84.00	86.76	13072.00
4	14	11	93.42	97.13	21776.00
4	14	12	102.17	106.61	17045.20
4	14	13	109.87	114.87	34718.70
5	5	4	106.94	109.02	58.73
5	6	4	85.32	86.80	74.96
5	6	5	105.86	109.32	782.57
5	7	4	72.70	73.14	83.67
5	7	5	90.03	91.81	1002.00
5	7	6	107.55	110.72	1727.42
5	8	4	63.13	63.48	69.65
5	8	5	76.60	77.68	1063.31
5	8	6	92.04	94.86	2335.00

Table A.12.: *Results of the exact simulation of the Markov chains of the* BBF_k *and* NF_k *algorithms (continued).*

m	B	u	$\mathbb{E}[\mathrm{BBF}_k]$	$\mathbb{E}[\mathrm{NF}_k]$	time [s]
5	8	7	107.07	111.09	5861.31
5	9	4	56.20	56.40	95.39
5	9	5	67.72	68.31	1104.30
5	9	6	80.80	82.19	2644.17
5	9	7	95.01	97.82	7860.54
5	9	8	108.04	111.83	3282.17
5	10	4	50.52	50.62	108.33
5	10	5	60.77	61.20	834.69
5	10	6	71.38	72.37	2741.75
5	10	7	83.42	85.63	9262.64
5	10	8	96.18	99.73	4571.52
5	10	9	107.72	112.10	26725.40
5	11	4	45.97	46.02	114.82
5	11	5	55.25	55.56	1046.26
5	11	6	64.75	65.39	3231.74
5	11	7	75.05	76.27	9863.31
5	11	8	86.59	88.92	5493.74
5	11	9	98.17	101.58	35517.10
5	11	10	108.44	112.57	41170.90
5	12	4	42.16	42.20	85.47
5	12	5	50.60	50.78	1164.64
5	12	6	59.19	59.70	2294.35
5	12	7	68.08	68.99	9790.54
5	12	8	77.77	79.54	6122.56
5	12	9	88.43	91.41	43257.80
5	12	10	98.93	102.85	55275.40
5	12	11	108.23	112.78	88846.20
5	13	4	38.95	38.97	97.92
5	13	5	46.71	46.83	1135.21
5	13	6	54.65	55.03	2833.83
5	13	7	62.68	63.36	10579.30
5	13	8	71.28	72.37	6487.19
5	13	9	80.71	82.68	49022.10
5	13	10	90.72	93.72	67716.40
5	13	11	100.28	104.08	115473.00
5	13	12	108.71	113.09	50884.10*
5	14	4	36.20	36.21	105.35

Table A.12.: *Results of the exact simulation of the Markov chains of the* BBF_k *and* NF_k *algorithms (continued).*

m	B	u	$\mathbb{E}[\mathrm{BBF}_k]$	$\mathbb{E}[\mathrm{NF}_k]$	time [s]
5	14	5	43.39	43.48	1144.85
5	14	6	50.68	50.94	3253.52
5	14	7	58.10	58.66	8145.65
5	14	8	65.69	66.58	7224.89
5	14	9	73.87	75.35	51474.60
5	14	10	82.79	85.28	78354.90
5	14	11	92.06	95.52	77932.90*
5	14	12	100.83	104.99	66313.80*
5	14	13	108.59	113.26	134709.00*

Bibliography

[ABK$^+$] Tobias Achterberg, Timo Berthold, Thorsten Koch, Alexander Martin, and Kati Wolter. SCIP – solving constraint integer programs. Available at `http://scip.zib.de`.

[ACDE07] David L. Applegate, William Cook, Sanjeeb Dash, and Daniel G. Espinoza. Exact solutions to linear programming problems. *Oper. Res. Lett.*, 35(6):693–699, 2007.

[Ach07] Tobias Achterberg. *Constraint Integer Programming.* PhD thesis, Technische Universität Berlin, 2007.

[ADLO07] Spyros Angelopoulos, Reza Dorrigiv, and Alejandro López-Ortiz. On the separation and equivalence of paging strategies. In *SODA 2007*, pages 229–237, 2007.

[ADU71] Alfred V. Aho, Peter J. Denning, and Jeffrey D. Ullman. Principles of optimal page replacement. *J. ACM*, 18(1):80–93, 1971.

[AFG05] Susanne Albers, Lene M. Favrholdt, and Oliver Giel. On paging with locality of reference. *J. Comput. System Sci.*, 70(2):145–175, 2005.

[AKR00] Norbert Ascheuer, Sven O. Krumke, and Jörg Rambau. Online Dial-a-Ride problems: Minimizing the completion time. In *Proceedings of the 17st Symposium on Theoretical Aspects of Computer Science*, volume 1770 of *Lecture Notes in Computer Science*, pages 639–650. Springer, 2000.

[Alb03] Susanne Albers. Online algorithms: A survey. *Math. Programming*, 97:3–26, 2003.

[AM00] Susanne Albers and Michael Mitzenmacher. Average-case analyses of first fit and random fit bin packing. *Random Structures Algorithms*, 16(3):240–259, 2000.

[AS09] S. Angelopoulos and P. Schweitzer. Paging and list update under bijective analysis. In *Proceedings of the 20th ACM-SIAM Symposium on Discrete Algorithms*, pages 1136–1145, 2009.

[Bar02] Gina Carol Barney. *Elevator Traffic Handbook: Theory and Practice.* Taylor and Francis, 2002.

[Bec04] Luca Becchetti. Modeling locality: A probabilistic analysis of LRU and FWF. In *Proceedings of the 12th European Symp. on Algorithms (ESA)*, pages 98–109, 2004.

[BEY98] Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

[BF07] Joan Boyar and Lene M. Favrholdt. The relative worst order ratio for online algorithms. *ACM Trans. Algorithms*, 3(2):Article 22, 2007.

[BIRS95] A. Borodin, S. Irani, P. Raghavan, and B. Schieber. Competitive paging with locality of reference. *J. Comput. System Sci.*, 50(2):244–258, 1995.

[BJL$^+$84] J. L. Bentley, D. S. Johnson, F. T. Leighton, C. C. McGeoch, and L. A. McGeoch. Some unexpected expected behavior results for bin packing. In *Proceedings of the 16th Annual ACM Symposium on Theory of Computing*, pages 279–288, 1984.

[BLMS$^+$06] Luca Becchetti, Stefano Leonardi, Alberto Marchetti-Spaccamela, Guido Schäfer, and Tjark Vredeveld. Average case and smoothed competitive analysis for the multi-level feedback algorithm. *Math. Oper. Res.*, 31(1):85–108, 2006.

[BLS92] Allan Borodin, Nathan Linial, and Michael E. Saks. An optimal on-line algorithm for metrical task systems. *J. ACM*, 39(4):745–763, 1992.

[BM01] Justin Boyan and Michael Mitzenmacher. Improved results for route planning in stochastic transportation networks. In *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 895–902, 2001.

[bMBFP06] Mouad ben Mamoun, Ana Bušić, Jean-Michel Fourneau, and Nihal Pekergin. Increasing convex monotone Markov chains: Theory, algorithms, and applications. In Amy N. Langville and William J. Stewart, editors, *MAM 2006: Markov Anniversary Meeting*, pages 189–210. Boson Books, 2006.

[BSL96] Dimitris J. Bertsimas and David Simchi-Levi. A new generation of vehicle routing research: Robust algorithms, addressing uncertainty. *Operations Res.*, 44(2):286–304, 1996.

[CB98] Robert H. Crites and Andrew G. Barto. Elevator group control using multiple reinforcement learning agents. *Machine Learning*, pages 235–262, 1998.

[CCG$^+$02] E. G. Coffman, Jr., C. Courcoubetis, M. R. Garey, D. S. Johnson, P. W. Shor, R. R. Weber, and M. Yannakakis. Perfect packing theorems and the average-case behavior of optimal and online bin packing. *SIAM Rev.*, 44(1):95–108, 2002.

[CD73] Edward G. Coffman, Jr. and Peter J. Denning. *Operating Systems Theory*. Series in automatic computation. Prentice-Hall, 1973.

[CGJ97] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson. Approximation algorithms for bin packing: A survey. In Dorit S. Hochbaum, editor, *Approximation Algorithms for NP-hard problems*, chapter 2. PWS Publishing, 1997.

[Cha92] Barun Chandra. Does randomization help in on-line bin packing? *Inform. Process. Lett.*, 43(1):15–19, 1992.

[CI89] János Csirik and Balázs Imreh. On the worst-case performance of the nkf bin-packing heuristic. *Acta Cybernet.*, 9(2):89–105, 1989.

[CJ01] J. Csirik and D. S. Johnson. Bounded space on-line bin packing: Best is better than first. *Algorithmica*, 11:115–138, 2001.

[CJSW93] E. G. Coffman, Jr., D. S. Johnson, P. W. Shor, and R. R. Weber. Markov Chains, computer proofs, and average-case analysis of best fit bin packing. In *Proc. 25th Ann. ACM Symp. on Theory of Computing*, pages 412–421, 1993.

[CMT81] Nicos Christofides, A. Mingozzi, and P. Toth. State-space relaxation procedures for the computation of bounds to routing problems. *Networks*, 11:145–164, 1981.

[CPL] ILOG CPLEX. `http://www.ilog.com/products/cplex/`.

[CSHY80] E. G. Coffman, Jr., K. So, M. Hofri, and A. C. Yao. A stochastic model of bin-packing. *Information and Control*, 44:105–115, 1980.

[Dal68] Daryl J. Daley. Stochastically monotone Markov chains. *Z. Wahrsch. Verw. Gebiete*, 10:305–317, 1968.

[DDS05] Guy Desaulniers, Jacques Desrosiers, and Marius M. Solomon, editors. *Column generation.* GERAD 25^{th} anniversary series. Springer, 2005.

[Den68] P. J. Denning. The working set model of program behavior. *Communications of the ACM*, 11:323–333, 1968.

[Den80] P. J. Denning. Working sets past and present. *IEEE Transactions on Software Engineering*, 6:64–84, 1980.

[DLO05] Reza Dorrigiv and Alejandro López-Ortiz. A survey of performance measures for on-line algorithms. *SIGACT News*, 36(3):67–81, 2005.

[DLO08] Reza Dorrigiv and Alejandro López-Ortiz. On certain new models for paging with locality of reference. In *WALCOM 2008*, volume 4921 of *Lecture Notes in Computer Science*, pages 200–209, 2008.

[Doi00] M. Doisy. A coupling technique for stochastic comparison of functions of Markov processes. *Journal of Applied Mathematics & Decision Sciences*, 4(1):39–64, 2000. Available at `http://www.hindawi.com/GetRegularIssueArticles.aspx?journal=JAMDS&volume=4`.

[FKL+91] Amos Fiat, Richard Karp, Mike Luby, Lyle McGeoch, Daniel Sleator, and Neal E. Young. Competitive paging algorithms. *J. Algorithms*, pages 685–699, 1991.

[FM97] A. Fiat and M. Mendel. Truly online paging with locality of reference. In *FOCS '97: Proceedings of the 38th Annual Symposium on Foundations of Computer Science*, pages 326–335, 1997.

[FR06] Philipp Friese and Jörg Rambau. Online-optimization of a multi-elevator transport system with reoptimization algorithms based on set-partitioning models. *Discrete Appl. Math.*, 154(13):1908–1931, 2006. also available as ZIB Report 05-03.

[FW74] P. A. Franaszek and T. J. Wagner. Some distribution-free aspects of paging algorithm performance. *J. ACM*, 21(1):31–39, 1974.

[FW98] Amos Fiat and Gerhard J. Woeginger, editors. *Online Algorithms: The State of the Art*, volume 1442 of *Lecture Notes in Computer Science*. Springer, 1998.

[GCC] GCC, the GNU compiler collection. `http://gcc.gnu.org/`.

[GKRT02] Martin Grötschel, Sven Oliver Krumke, Jörg Rambau, and Luis M. Torres. Online-dispatching of automobile service units. In U. Leopold-Wildburger, F. Rendl, and G. Wäscher, editors, *Operations Research Proceedings*, pages 168–173. Springer, 2002.

[Glo70] Gordon David Gloss. *The computer control of passenger traffic in large lift systems*. PhD thesis, Victoria University of Manchester, 1970.

[GMP06] GNU GMP – the GNU multiple precision arithmetic library. available at `http://gmplib.org/gmp-man-4.2.1.pdf`, 2006.

[GW95] Gábor Galambos and Gerhard J. Woeginger. On-line bin packing – a restricted survey. *ZOR—Math. Methods Oper. Res.*, 42:25–45, 1995.

[HKR00] Dietrich Hauptmeier, Sven O. Krumke, and Jörg Rambau. The online Dial-a-Ride problem under reasonable load. In *CIAC 2000*, volume 1767 of *Lecture Notes in Computer Science*, pages 125–136. Springer, 2000.

[HKR06] Benjamin Hiller, Sven Oliver Krumke, and Jörg Rambau. Reoptimization gaps versus model errors in online-dispatching of service units for ADAC. *Discrete Appl. Math.*, 154(13):1897–1907, 2006. Traces of the Latin American

Conference on Combinatorics, Graphs and Applications – A selection of papers from LACGA 2004, Santiago, Chile.

[HKRW01] Dietrich Hauptmeier, Sven O. Krumke, Jörg Rambau, and Hans-Christoph Wirth. Euler is standing in line – Dial-a-Ride problems with precedence-constraints. *Discrete Appl. Math.*, 113(1):87–107, 2001.

[HT08] Benjamin Hiller and Andreas Tuchscherer. Real-time destination-call elevator group control on embedded microcontrollers. In *Operations Research Proceedings 2007*, pages 357–362. Springer, 2008.

[HV08] Benjamin Hiller and Tjark Vredeveld. Probabilistic analysis of online bin coloring algorithms via stochastic comparison. In *Proceedings of the 16th Annual European Symposium on Algorithms*, volume 5193 of *Lecture Notes in Computer Science*, pages 528–539. Springer, 2008.

[IKP96] Sandy Irani, Anna R. Karlin, and Steven Phillips. Strongly competitive algorithms for paging with locality of reference. *SIAM J. Comput.*, 25(3):477–497, 1996.

[JDU$^+$74] D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey, and R. L. Graham. Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM J. Comput.*, 3(4):299–325, 1974.

[Joh74] D. S. Johnson. Fast algorithms for bin packing. *J. Comput. System Sci.*, 8(8):272–314, 1974.

[Kar82] Narendra Karmarkar. Probabilistic analysis of some bin-packing problems. In *FOCS 1982*, pages 107–111, 1982.

[KdPSR01] Sven Oliver Krumke, Willem E. de Paepe, Leen Stougie, and Jörg Rambau. Online bin coloring. In Friedhelm Meyer auf der Heide, editor, *Proceedings of the 9th Annual European Symposium on Algorithms*, volume 2161 of *Lecture Notes in Computer Science*, pages 74–84, 2001.

[KO02] Jana Koehler and Daniel Ottiger. An AI-based approach to destination control in elevators. *AI Magazine*, 23(3):59–78, 2002.

[KP94] Elias Koutsoupias and Christos Papadimitriou. Beyond competitive analysis. In *Proceedings of the 35th Annual IEEE Symposium on Foundations of Computer Science*, pages 394–400, 1994.

[KP00] Bala Kalyanasundaram and Kirk Pruhs. Speed is as powerful as clairvoyance. *J. ACM*, 47(4):617–643, 2000.

[KPR00] Anna R. Karlin, Steven J. Phillips, and Prabhakar Raghavan. Markov paging. *SIAM J. Comput.*, 30(2):906–922, 2000.

[KRS98] Claire Kenyon, Yuval Rabani, and Alistair Sinclair. Biased random walks, Lyapunov functions, and stochastic analysis of best fit bin packing. *J. Algorithms*, 27(2):218–235, 1998.

[KRT02] Sven Oliver Krumke, Jörg Rambau, and Luis M. Torres. Realtime-dispatching of guided and unguided automobile service units with soft time windows. In Rolf H. Möhring and Rajeev Raman, editors, *Proceedings of the 10th Annual European Symposium on Algorithms*, volume 2461 of *Lecture Notes in Computer Science*, pages 637–648. Springer, 2002.

[KSLKK98] Chang Bum Kim, Kyoung A. Seong, Hyung Lee-Kwang, and Jeong O. Kim. Design and implementation of a fuzzy elevator group control system. *IEEE Transactions on Systems, Man and Cybernetics, Part A*, 28(3):277–287, 1998.

[Las70] Leon S. Lasdon. *Optimization Theory for Large Systems.* Macmillan, 1970.

[Lin92] Torgny Lindvall. *Lectures on the coupling method.* Wiley Series in Probability and Mathematical Statistics. John Wiley & Sons, 1992.

[LL85] C. C. Lee and D. T. Lee. A simple online bin-packing algorithm. *J. ACM*, 32(3):562–572, 1985.

[Mao93] Weizhen Mao. Tight worst-case performance bounds for next-k-fit bin packing. *SIAM J. Comput.*, 22(1):46–56, 1993.

[Mit96] Michael Mitzenmacher. Bounds on the greedy routing algorithm for array networks. *J. Comput. System Sci.*, 53:317–327, 1996.

[MS91] Lyle A. McGeoch and D. D. Sleator. A strongly competitive randomized paging algorithm. *Algorithmica*, 6:816–825, 1991.

[MS02] Alfred Müller and Dietrich Stoyan. *Comparison Models for Stochastic Models and Risks.* John Wiley & Sons, 2002.

[NB03] Daniel Nikovski and Matthew Brand. Decision-theoretic group elevator scheduling. In *Proceedings of the 13th International Conference on Automated Planning and Scheduling*, 2003.

[NR08] Nir Naaman and Raphael Rom. Average case analysis of bounded space bin packing algorithms. *Algorithmica*, 50:72–97, 2008.

[PC97] David L. Pepyne and Christos G. Cassandros. Optimal dispatching control for elevator systems during uppeak traffic. *IEEE Transactions on Control Systems Technology*, 5(6):629–642, 1997.

[PC98] David L. Pepyne and Christos G. Cassandros. Design and implementation of an adaptive dispatching controller for elevator systems during uppeak traffic. *IEEE Transactions on Control Systems Technology*, 6(5):635–650, 1998.

[PS06] Konstantinos Panagiotou and Alexander Souza. On adequate performance measures for paging. In *STOC '06: Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, pages 487–496, 2006.

[Put05] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* John Wiley and Sons, 2nd edition, 2005.

[Ram89] Prakash Ramanan. Average-case analysis of the Smart Next Fit algorithm. *Inform. Process. Lett.*, 31(5):221–225, 1989.

[RS08] Giovanni Righini and Matteo Salani. New dynamic programming algorithms for the resource constrained elementary shortest path problem. *Networks*, 51(3):156–170, 2008.

[Sch90] Jordis Schröder. Advanced dispatching: Destination hall calls + instant car-to-call assignments: M10. *Elevator World*, pages 40–46, March 1990.

[Sec99] Bernhard Seckinger. Synthese von Aufzugssteuerungen mit Hilfe von constraintbasierten Suchverfahren. Diplomarbeit, Universität Freiburg, 1999.

[Sho86] P. W. Shor. The average-case analysis of some on-line algorithms for bin packing. *Combinatorica*, 6(2):179–200, 1986.

[Sin93] Alistair Sinclair. *Algorithms for Random Generation and Counting: A Markov Chain Approach.* Progress in Theoretical Computer Science. Birkhäuser, 1993.

[SK99] Bernhard Seckinger and Jana Koehler. Online-Synthese von Aufzugssteuerungen als Planungsproblem. In *13th Workshop on Planning and Configuration*, pages 127–134, 1999.

[Smi56] Wayne E. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3(1-2):59–66, 1956.

[SP02] Rory Smith and Richard Peters. ETD algorithm with destination dispatch and booster options. *Elevator World*, pages 136–145, July 2002.

[SS94] Moshe Shaked and J. George Shanthikumar. *Stochastic orders and their applications.* Probability and mathematical statistics. Academic Press, 1994.

[SS06] A. Souza and A. Steger. The expected competitive ratio for weighted completion time scheduling. *Theory of Computing Systems*, 39:121–136, 2006.

[SSE03] Janne Sorsa, Marja-Liisa Siikonen, and H. Ehtamo. Optimal control of double-deck elevator group using genetic algorithm. *Intl. Trans. in Op. Res.*, 10(2):103–114, 2003.

[SSS06] Mark Scharbrodt, Thomas Schickinger, and Angelika Steger. A new average case analysis for completion time scheduling. *J. ACM*, pages 121–146, 2006.

[ST85] Daniel Dominic Sleator and Robert Endre Tarjan. Amortized efficiency of list update and paging rules. *Comm. ACM*, 28(2):202–208, 1985.

[ST04] D. A. Spielman and S. H. Teng. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *J. ACM*, 51:385–463, 2004.

[Tor98] Eric Torng. A unified analysis of paging and caching. *Algorithmica*, 20(2):175–200, 1998.

[TUA05a] Shunji Tanaka, Yukihiro Uraguchi, and Mituhiko Araki. Dynamic optimization of the operation of single-car elevator systems with destination hall call registration: Part I. Formulation and simulations. *European J. Oper. Res.*, 167(2):550–573, 2005.

[TUA05b] Shunji Tanaka, Yukihiro Uraguchi, and Mituhiko Araki. Dynamic optimization of the operation of single-car elevator systems with destination hall call registration: Part II. The solution algorithm. *European J. Oper. Res.*, 167(2):574–587, 2005.

[TY99a] Tapio Tyni and Jari Ylinen. Method and apparatus for allocating landing calls in an elevator group. US Patent 5,932,852, August 3, 1999. KONE Corporation.

[TY99b] Tapio Tyni and Jari Ylinen. Genetic procedure for allocating landing calls in an elevator group. US Patent 5,907,137, May 25, 1999. KONE Corporation.

[TY05] Tapio Tyni and Jari Ylinen. Method and apparatus for allocating passengers by a genetic algorithm. US Patent 6,913,117, July 5, 2005. KONE Corporation.

[TY06] Tapio Tyni and Jari Ylinen. Evolutionary bi-objective optimisation in the elevator car routing problem. *European J. Oper. Res.*, 169(3):960–977, 2006.

[TYMR06] Tapio Tyni, Jari Ylinen, Mika Matela, and Toni Rintala. Genetic allocation method for an elevator group. US Patent 7,140,472, November 28, 2006. KONE Corporation.

[You94] Neal E. Young. The k-server dual and loose competitiveness for paging. *Algorithmica*, 11(6):525–541, 1994.

[You00] Neal E. Young. On-line paging against adversarially biased random inputs. *J. Algorithms*, 37(1):218–235, 2000.

[ZO00] Kenny Qili Zhu and Kar-Loon Ong. A reactive method for real time dynamic vehicle routing problem. In *12th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2000)*, pages 176–, 2000.