Wiebke Höhn

Dissertation

Complex Single Machine Scheduling

Theoretical and Practical Aspects of Sequencing

Feb 2014

TU Berlin

Complex Single Machine Scheduling

Theoretical and Practical Aspects of Sequencing

vorgelegt von
Dipl. Math. Wiebke Höhn
Berlin

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

Doktor der Naturwissenschaften
– Dr. rer. nat. –

genehmigte Dissertation

Promotionsausschuss

Vorsitzender:	Prof. Dr. Fredi Tröltzsch
Berichter:	Prof. Dr. Rolf H. Möhring
	Prof. Dr. Marco E. Lübbecke

Tag der wissenschaftlichen Aussprache: 2. September 2013

Berlin 2014
D83

Acknowledgements

This thesis would have not been possible without the direct and indirect help of many people whom I wish to thank in the following. In particular, I am indebted to my advisor Rolf H. Möhring for his great support, his trust and for giving me the freedom of independently choosing research topics and collaboration partners. The excellent research conditions he created in the COGA group are exceptional and all a doctoral student can hope for.

Moreover, I wish to thank Marco E. Lübbecke for taking the second assessment of this thesis. Financially, this work has been supported by the German Research Foundation (DFG) as part of the Priority Programme *Algorithm Engineering* (SPP 1307).

I am deeply indebted to my coauthors Torsten Gellert, Tobias Jacobs, Felix König, Marco E. Lübbecke and Rolf H. Möhring with whom I enjoyed several vibrant discussions, experienced all ups and downs of research, and, in the end, with whom I simply had a great time. In particular, I am grateful to Felix G. König and Marco E. Lübbecke whose enthusiasm and euphoria was contagious and from whom I learned a lot about academic research, industrial projects and the combination of the two. Moreover, I greatly enjoyed working with Tobias Jacobs whom I appreciate as a pleasant and very humorous person. I am thankful for two very inspiring and productive months in Tokyo, including countless intensive discussions at NII and great after-work Isakaya evenings.

Thanks go also to Michael Bastubbe, Torsten Gellert and Olaf Maurer who, at that time as student assistants, supported the implementation work in our coil coating project.

Not contributing to the content of this thesis, but still being part of my time in the COGA group, is the project DFG Science TV. I am grateful to Rolf H. Möhring for providing us with the time necessary for doing it, and I am especially indebted to Marco E. Lübbecke for being an excellent co-perfectionist in this crazy project.

I would particularly like to thank my current and former colleagues in the COGA group for the wonderful work atmosphere, for countless nice conversations on math and non-math, and for memorable conferences and conference evenings. I was happy to have Christina Büsing as my longtime room mate, and I am thankful to her for being such a good listener. I very much enjoyed all the short chats during the day. A special thanks goes also to everyone who is involved in making the coffee of COGA South so much superior to the COGA North variant. Furthermore, I wish to express my gratitude to Ralf Hoffmann, Dorothea Kiefer and Gabriele Klink, the good souls of COGA, who keep things running so smoothly.

Moreover, I am grateful to Yann Disser, Max Klimm and Axel Werner for carefully proofreading different parts of this thesis and for lively discussions on thesis writing in general. In particular, I am indebted to Jannik Matuschke whom I want to thank not only for extensive proofreading and various critical and helpful comments, but even more for everything else.

Finally, I wish to thank my parents for their unconditional and unlimitted support and for the freedom they gave me throughout the years.

Berlin, June 2013 *Wiebke Höhn*

Contents

Introduction

"I don't think writers are sacred, but words are. They deserve respect. If you get the right ones in the right order, you can nudge the world a little or make a poem which children will speak for you when you're dead."

Tom Stoppard

Questions of finding good orders appear virtually everywhere in daily life. Their nature is highly manifold, and so are the aims and side constraints. At a rather personal level, we may seek after an order in which to add our favorite songs to a playlist, or we may need to decide in which order to go to shops for our weekend shopping. From a more professional point of view, doctors have to determine the order in which they serve their patients, factories have to decide in which order to manufacture their products, and engineers have to devise rules for the order in which to process tasks on a computer. What is meant by a *good* order, though, is often not precisely clear or not even measurable.

While good playlists are obviously a matter of taste, at a first glance one might think that criteria for good (processing) sequences in industry are objectively determined and easy to formulate. However, precise mathematical models, providing solutions which come close to what is named *good* by practitioners, are usually the result of long discussions and hard work which is inevitable for the success of a project.

This thesis starts at the point where adequate models and problem definitions have already been devised, partially in close cooperation with practitioners from steel and dairy industry. On that note we consider two general classes of problems from the broad field of sequencing problems, a class of classic single machine scheduling problems with generalized min-sum objective, aiming for schedules which are fair in a certain sense, and a class of integrated sequencing and allocation problems as they commonly arise in practice, where the major goal is a high throughput.

Our overall aim is the design and the analysis of efficient algorithms for these two problem classes. While the former class allows for the rigorous mathematical analysis of algorithms in terms of provable performance guarantees, in the latter case, such general guarantees are unlikely to be achieved. This is mainly due to complex problem formulations and algorithms which make it nearly impossible to analyze the performance jointly for all problem instances. In order to still provide meaningful performance guarantees, we resort to the computation of instance-dependent lower bounds which we then use to bound the optimality gap.

For the class of single machine scheduling problems, we provide a thorough mathematical analysis for the performance of *Smith's rule*—one of the classic algorithms in

scheduling theory—including tight approximation guarantees for a broad subclass of objective functions. Moreover, we prove the existence of a certain order relation on the jobs, observed by any optimal schedule, which was conjectured by Mondal and Sen [MS00] more than a decade ago. Finally, we give the first strong NP-hardness result for this problem, which already holds for a very restricted special case, while even for the most general problem variant only weak NP-hardness was known.

Concerning integrated sequencing and allocation problems, we propose a generic optimization framework which aims at separating complex problem-specific side constraints from general sequencing aspects. Applying this approach to a scheduling problem from coil coating, we are faced with a concurrent setup allocation subproblem, for which we observe a close relation to the maximum weight independent set problem on a special class of multiple-interval graphs. Thorough investigations of this problem were finally the basis for achieving a makespan reduction of over 13 % on average for the coil coating line of our industrial partner Salzgitter Flachstahl. Moreover, we concretize the generic approach for a problem from dairy industry, obtaining solutions on par with the manual production plans in use at the production site of our project partner Sachsenmilch, proving that both our and the experts' solutions are within 2 % of optimality.

Outline of the thesis

The thesis is divided into two parts, where the first part is dedicated to single machine scheduling with generalized min-sum objective and the second part to integrated sequencing and allocation problems.

Part I

As pointed out, in the first part, we address a classic machine scheduling problem. The aim in this problem is to schedule jobs of different priorities on a single machine such that the total weighted completion time cost is minimized, where this cost is specified by some non-decreasing cost function. Our major focus is on convex and concave cost functions, and in particularly, on monomials.

Chapter 1. In Section 1.1 of this introductory chapter, we first formally define the considered single machine scheduling problem, additionally pointing out interesting relations to scheduling with non-uniform speed and to a so called *vehicle refueling problem*. Thereafter, in Section 1.2, we discuss related work and introduce the algorithm *Smith's rule*, which will be analyzed in Chapter 2. This simple sorting rule schedules the jobs in non-increasing order of their density, i.e., their ratio of weight and processing time.

Chapter 2. In the second chapter, we investigate the performance of Smith's rule on instances of the described single machine scheduling problem with generalized min-sum objective. Starting with a simple combinatorial analysis for polynomial cost functions with non-negative coefficients in Section 2.1, we show that the approximation guarantee in this case is at least the degree of the polynomial. In Section 2.2.1 thereafter, by explicitly constructing worst-case instances, we provide a tight analysis of the approximation guarantee of Smith's rule under any particular convex or concave cost function. More specifically, for this wide class of cost functions we reduce the task of determining a worst case problem instance to a continuous optimization problem, which can be

solved by standard algebraic or numerical methods. As we show in Section 2.2.2, the analysis can be further simplified for polynomial cost functions with positive coefficients. In this case, the tight approximation ratio can be calculated as the root of a univariate polynomial. For quadratic and cubic cost, this yields approximation guarantees of 1.31 and 1.76, respectively.

In contrast to the above implicit description of the approximation factor, in Section 2.2.3, we provide explicit sandwich bounds for polynomial cost functions. In particular, we show that the approximation ratio is asymptotically equal to $k^{(k-1)/(k+1)}$, denoting by k the degree of the polynomial. In Section 2.3, to overcome unrealistic worst case instances containing only either huge or tiny jobs, we also give tight bounds for the case of integral processing times that are parameterized by the maximum and total processing time.

Chapter 3. After having addressed approximation results as mentioned above, this chapter focuses on the complexity of the problem as well as on exact algorithms and related structural insights. In Section 3.1, we give the first strong NP-hardness result for the considered single machine scheduling problem, while so far only weak NP-hardness in case of convex cost was known. We show that the problem is already strongly NP-hard for a very restricted class of piece-wise linear cost functions, which is related two alternating processor speeds in the model with non-uniform speed.

In Section 3.2, we turn towards so called *order rules* which imply relative orders of certain job pairs in an optimal schedule. Such rules help to speed up or to allow exact computations in the first place. Throughout the past decades, a considerable number of different kinds of order rules have been proposed for the case of quadratic cost functions. Following this line of research, we also address the case of quadratic cost, for which we enhance the map of known order rules by proving an extended version of a rule that has been conjectured by Mondal and Sen [MS00] more than a decade ago. Finally, in Section 3.3, we discuss exact algorithms that allow for embedding order rules; different variants of a known best-first graph search algorithms and a quadratic integer programming formulation that admits to add order rules as additional linear constraints. Since the former approach makes use of lower bounds, we also propose a new such bound, which is based on the tight parameterized analysis of Smith's rule.

Chapter 4. In the fourth chapter, we aim at empirically evaluating the results and insights of the previous chapters. In an extensive computational study for quadratic cost functions, we systematically analyze the influence of different global order rules on the performance of the exact algorithms described above. The results are presented in Section 4.2.1. In Section 4.2.2, we additionally compare the performance of different lower bounds with the bound that is based on the parameterized analysis of Smith's rule. Finally, in Section 4.2.3, we also evaluate the optimality gap of Smith's rule for different monomial cost functions. As will be described in Section 4.1, our experiments are based on sets of problem instances that have been randomly generated using a new method which allows us to adjust a certain degree of difficulty of the instances.

Chapter 5. This final chapter aims at pointing out interesting problems which remain open after the discussions in the previous chapters.

Part II

In the second part of the thesis we address a general class of complex scheduling problems as they can be found in almost every branch of industry. We propose a general optimization framework for this problem class, which we then apply to problems from steel and from dairy industry.

Chapter 6. In this first chapter of the second part, we introduce the abstract class of so called *integrated sequencing and allocation problem*s, covering all problems addressed in Part II. Very briefly explained, such problems contain a common sequencing aspect—the order of the jobs on the considered machine—and very problem-specific decisions which have to be taken for a fixed processing sequence, the latter being referred to as *allocations*. We give a formal problem definition in Section 6.1 before afterwards discussing related work in Section 6.2. The general optimization framework is then described in Section 6.3. It utilizes a genetic algorithm for the sequencing while requiring an adapted subroutine to handle the allocation decisions.

Chapter 7. In this and the following two chapters, we consider a particular integrated sequencing and allocation problem from steel production. A sequence of coils of sheet metal needs to be color coated in consecutive stages. Different coil geometries and changes of colors necessitate time-consuming setup work. In most coating stages one can choose between two parallel color tanks. This can either reduce the number of setups needed or enable setups concurrent with production. Overall, the aim is to compute a minimum makespan schedule comprising the sequencing of coils and the allocation of color tanks and setup work. A detailed description of this complex scheduling problem is given in Section 7.1. Moreover, in Section 7.2 an interval model for concurrent setup allocations is proposed. In this model, every feasible tank assignment and corresponding setup schedule can be represented by a set of non-conflicting 2-dimensional intervals.

Chapter 8. This chapter addresses the graph theoretical model for the allocation subproblem of the coil coating problem. Utilizing the interval model mentioned in the previous chapter, the allocation problem reduces to the maximum weight independent set problem in 2-union graphs, a special class of multiple-interval graphs. In fact, the graphs, that stem from instances of the allocation problem, observe an additional property which we call m-*composite*, where the parameter m corresponds to the number of coating stages. In Section 8.1, we give a formal definition of this graph class and relate it to other classes of multiple-interval graphs. Moreover, we report on related work for the maximum weight independent set problem in these graph classes.

In Section 8.2, we focus on the maximum weight independent set problem in m-composite 2-union graphs while in Section 8.3 we address the allocation problem. Even though being closely related, neither positive nor negative results directly translate from one of these problems to the other. For constant m, we show that the graph problem allows for a polynomial-time dynamic program which also applies to the allocation problem, however, only under certain restrictions of the setup durations. On the other hand, when m is part of the problem input, we prove separately that both problems are strongly NP-hard.

Chapter 9. Aiming for an efficient algorithm for the scheduling of coil coating lines, in Section 9.1, we adapt the generic algorithmic approach introduced in Section 6.3. This

necessitates to design efficient heuristics for the allocation problem as well as to generate a diverse set of coil sequences that form the initial population for the genetic sequencing algorithm, where the former is based on the graph theoretical interpretation of the allocation problem. The quality of our solutions is evaluated via instance-dependent lower bounds, which we introduce in Section 9.2. They stem from an integer program which is based on a combinatorial relaxation of the problem, showing that our solutions are within 10 % of the optimum. Our algorithm is implemented at Salzgitter Flachstahl, a major German steel producer. This has led to an average reduction in makespan by over 13 %, exceeding the initial expectations of the practitioners.

Chapter 10. In the tenth chapter, we consider a second integrated sequencing and allocation problem which arises at filling lines in dairy industry. The underlying allocation problem involves certain packing and covering aspects which not only occur in dairy production but also in other branches of industry. A detailed problem formulation is given in Section 10.1, while in Section 10.2, we discuss structural properties of the problem, e.g., the fact that every constraint of the allocation problem allows for being formulated as so called *generalized capacity constraint*.

In Section 10.3.1, as for the coil coating problem, we adapt the generic approach for integrated sequencing and allocation problems. Based on the above insights into structural properties of the problem, we propose different allocation subroutines. In cooperation with Sachsenmilch, the algorithm was implemented for their bottleneck filling line, and evaluated in an extensive computational study. For the original data from production, our algorithm computes almost optimal solutions. As a surprising result, our simple greedy algorithms for the allocation problem outperform the more elaborate ones in many aspects.

Chapter 11. In the very last chapter, we draw conclusions concerning our overall work on integrated sequencing and allocation problems.

Part I

Generalized Min-Sum Scheduling

Chapter 1

Introduction

We consider a classic machine scheduling problem in which we aim for schedules which are in a certain sense *fair*. Speaking in terms of the time customers need to wait for their job to be finished, a first natural objective to model fairness is the average completion time, which corresponds to the weighted average in the presence of different priorities among the customers. This objective, or rather the closely related (weighted) sum of completion times, has been studied in various problem settings in the literature. For the most simple single machine case Smith [Smi56] showed already in the 1950s that a simple sorting rule computes optimal schedules—which is the reason for this algorithm to be widely referred to as *Smith's rule*.

In this part of the thesis, we address a generalization of the problem studied by Smith. In this problem the cost of a job is not given by its completion time directly but by a non-decreasing cost function on this value. The cost function allows for modeling several problems which are of interest in their own right. For instance, by utilizing L_k-norms (or closely related monomials), one can flexibly compromise between problems that focus on average and worst-case cost, or alternatively, one can use the cost function to model non-uniform processor speed. As a consequence of their universality, problems addressing generalized min-sum objectives have recently attracted much attention, see e.g., [BP10a, BP10b, CS11, ELMS+12, IMP12, MV13, SW10].

In the following chapters, our main focus is on concave and convex cost functions. Monomials fall into this class as well as functions which result from modeling increasing or decreasing processor speed, or exponential functions as they are used in air traffic scheduling to relax the common first-come first-serve rule. Working at the intersection of theory and practice, we aim at achieving theoretical results which are as close as possible to reality on the one hand, while on the other hand, utilizing theoretical structural insights when designing practical algorithms. Following this philosophy, we consider two major questions: How well does the natural and simple Smith's rule perform for particular non-linear cost functions in the worst-case and how does the performance guarantee change when considering only restricted classes of instances? Can we rule out certain structures in optimal schedules which help us to design exact algorithms?

Organization of Part I. In the remainder of this chapter, we give a formal definition of the scheduling problem, also pointing out connections to related problems, before discussing related work. Thereafter, in Chapter 2, we provide a tight analysis of the approximation factor of Smith's rule under any particular convex or concave cost function. For monomial cost functions we show that this factor is asymptotically equal to $k^{(k-1)/(k+1)}$, denoting by k the rational degree of the monomial. To overcome unrealistic worst case instances,

we also give tight bounds for the case of integral processing times that are parameterized by the maximum and total processing time.

In Chapter 3 we turn to exact algorithms and to complexity results, showing strong NP-hardness for piece-wise linear cost functions neither being convex nor concave. Concerning exact algorithms, we address so called *order rules*, which rule out certain relative orders of job pairs in optimal solutions. For quadratic cost functions we prove correctness of a very efficient rule which was conjectured by Mondal and Sen [MS00]. Finally, in Chapter 4, we report on experiments that were made in order to empirically assess the performance of the different order rules from Chapter 3 as subroutines in different exact solution methods. Moreover, we present results on the experimental performance guarantee of Smith's rule for different monomial cost functions. For a more detailed summary of the following chapters we refer to the main introduction.

1.1 Problem formulation

Formally, we consider the following single machine scheduling problem.

Single machine scheduling with generalized min-sum objective	
Given:	A set of jobs $J = \{1, \dots, n\}$ where each job $j \in J$ has an individual weight $w_j \geq 0$ and processing time $p_j \geq 0$.
Task:	Schedule the jobs J on a single machine, i.e., assign them to non-overlapping processing intervals on this machine, such that the total weighted completion time cost $\sum_{j \in J} w_j f(C_j)$ is minimized, where C_j denotes the completion time of job j in the chosen schedule, and where $f : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ is a continuous and monotonically non-decreasing cost function (given by a value-giving oracle).

Note that the question of allowing preemption does not play a role here, because the jobs do not have release times and so the possibility of preemption never leads to a cheaper optimal schedule. Hence, we can think of a schedule as a permutation of the jobs. In the following we will point out two problems that are closely related to our problem.

Relation to scheduling under non-uniform processor speed. An important alternative interpretation of the above problem is the scenario of linear cost and non-uniform processor speed. Assume that the processor speed at any time t is given by a non-negative function $s : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$, and the processing times (or workloads) p_j of the jobs are given with respect to a unit speed processor. The total workload processed until time t is $S(t) := \int_0^t s(x)dx$. Conversely, if the total workload of job j and all jobs processed before it is p, then the completion time of j in the schedule is $S^{-1}(p)$. Therefore, the problem of minimizing the weighted sum of completion times in this setting is equivalent to above problem with cost function $f := S^{-1}$. Note that S^{-1} is always monotone, and it is continuous even if s is not. Moreover if s is increasing or decreasing then S^{-1} is concave or convex, respectively. The case of cost function h and processor speed function s is equivalent to the problem with cost function $f := h \circ S^{-1}$.

Relation to the vehicle refueling problem. Furthermore, we point out that the above problem allows to model a distant maximization problem whose complexity was posed as an open question by Woeginger during a Dagstuhl Seminar [ABMP10]. Given a set of n vehicles $j = 1, \dots, n$ with non-uniform tank volumes v_j and consumption rates c_j, one aims to maximize the maximum travel distance among all vehicles. The crucial point is that during traveling, one vehicle can refuel others. W.l.o.g. one can assume that at any point in time, all vehicles consume the petrol of only one of the vehicles' tank, and when this tank is empty, the corresponding vehicle stops traveling, and thus, consuming future petrol. This allows to see the problem as a (vehicle) sequencing problem where the travel distance of the last vehicle in the sequence is to maximize. Given a sequence π, this distance computes as

$$\sum_{j=1}^{n} \frac{v_{\pi(j)}}{\sum_{i=j}^{n} c_{\pi(i)}} .$$

Inverting the sequence π, and interpreting the vehicles as jobs j with processing times $p_j := c_j$ and weights $w_j := v_j$, this translates into the maximization variant of the single machine scheduling problem with objective $\sum w_j \frac{1}{C_j}$. To turn this into a minimization problem, we can simply add the constant $\sum_j w_j / \min_j p_j$ to the negative objective, leading to cost function $f(t) = \frac{1}{\min_j p_j} - \frac{1}{t}$. Since $\frac{1}{t}$ is convex, f is concave on the relevant interval $[\min_j p_j, \sum_j p_j]$. Of course, this transformation from minimization to maximization does not preserve approximation guarantees. However, if the scheduling problem can be shown to be NP-hard for every fixed concave cost function then this also implies the hardness of the vehicle refueling problem.

1.2 Preliminaries and related work

In this section, we will first introduce notation and terminology as it is commonly used in scheduling and in approximation theory. In the subsections thereafter, we will report on work related to scheduling a single machine with respect to a generalized min-sum objective. For a general introduction to the fields of combinatorial optimization and complexity theory, we refer to the textbooks by Schrijver [Sch03] and Garey and Johnson [GJ79], respectively.

1.2.1 Preliminaries

The three-field notation for scheduling problems. Graham et al. [GLLRK79] introduced a notation for machine scheduling problems in 1979 which is standard by now. The problems are represented by a triple $\alpha \,|\, \beta \,|\, \gamma$ where α describes the machine environment, β the job characteristics, and γ the objective. All problems we consider are single machine problems which is denoted by 1 in the α-field. For our problem as described above the β-field is empty. However, when discussing related work we will also consider problems which allow preemption, indicated by pmtn, and such with non-uniform release dates, indicated by r_j. In contrast to our problem, where we assume all jobs to be available from the beginning, in case of non-uniform release dates, a job $j \in J$ is released at time $r_j \geq 0$, and it cannot be processed before this time.

α	1	single machine
β	r_j	non-uniform release dates
	pmtn	preemption allowed
γ	$\sum f(C_j)$	total completion time cost with global cost function f
	$\sum w_j\, f(C_j)$	total weighted completion time cost with global cost function f
	$\sum w_j\, f(F_j)$	total cost on flow times $F_j := C_j - r_j$ with global cost function f
	$\sum f_j(C_j)$	total completion time cost with job-specific cost functions

Table 1.1: Used parameters of the three field notation $\alpha\,|\,\beta\,|\,\gamma$ by Graham et al. [GLLRK79], where α describes the machine environment, β the job characteristics, and γ the objective.

Concerning the γ-field, we consider the objective $\sum w_j\, f(C_j)$ as in our problem definition and its unweighted counterpart $\sum f(C_j)$ where the jobs are assumed to have uniform weights. In the presence of release dates, often the flow time based objectives $\sum f(F_j)$ and $\sum w_j\, f(F_j)$ are investigated, where the *flow time* F_j of a job $j \in J$ is given by the time between its release and its completion, i.e., by $C_j - r_j$. In contrast to the problems with one global cost function applying to all jobs, also the more general problem variant with job-specific cost functions has been considered. In this case, each job $j \in J$ has a personal cost function $f_j : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ and the objective is $\sum f_j(C_j)$. Note that it is redundant to consider weights alongside with job-specific cost functions as the weights can be modeled via the cost functions. Similarly, it is redundant to consider the flow time counterpart. For brevity, the summation indices are omitted in all the objectives.

Finally, the introduced single machine scheduling problem with generalized min-sum objective will be denoted as $1\,||\,\sum w_j\, f(C_j)$. In Table 1.1, we summarize the used parameters of the three-field notation.

Performance measures of algorithms. For the sake of completeness, we also formally define classic approximation algorithms and approximation schemes. An *α-approximation algorithm*, or *α-approximation* for short, is a polynomial-time algorithm that computes solutions whose objective function value is always within a factor of α of the optimum. The factor α is referred to as *approximation factor* or *approximation guarantee*. An approximation factor α is *tight* for an algorithm A, if for any $\beta < \alpha$ there is an instance for which the solution of A has an objective function value which is more than a factor of β away from the optimum.

A *polynomial-time approximation scheme* (PTAS) is an algorithm which computes solutions of approximation guarantee $1 + \varepsilon$ in polynomial time for any fixed $\varepsilon > 0$. A stronger variant in terms of the runtime is a *fully polynomial-time approximation scheme* (FPTAS) whose runtime is polynomial in $\frac{1}{\varepsilon}$ and the input size. A third variant are *quasi-polynomial-time approximation schemes* (QPTAS) which have a runtime of $\mathcal{O}(n^{\text{polylog}(n)})$ in the input size n for any fixed $\varepsilon > 0$.

As a generalization of the classic analysis of the approximation factor, Kalyanasundaram and Pruhs [KP00] introduced the model of *resource augmentation* for the analysis of machine scheduling problems. In this model solutions of an algorithm with respect to a higher machine speed are compared to optimal solutions on a unit-speed machine. Formally, a polynomial-time algorithm is *s-speed α-approximative* if the objective function value of solutions computed with the algorithm on an s-speed machine are always

within a factor of α of the optimum on a 1-speed machine. Note that due to non-uniform release times, which do not underlie the machine speed, this is not equivalent to simply providing a performance guarantee of $s \cdot \alpha$. Moreover, even for uniform release dates, we do not necessarily obtain an $(s \cdot \alpha)$-approximation when considering non-linear cost functions. A necessary condition on a cost function to observe a similar behavior is that $f(s \cdot t) = \phi(s)\, f(t)$ for some function ϕ. In this case an s-speed α-approximation translates into a (1-speed) $(\phi(s) \cdot \alpha)$-approximation. As we will show in Observation 2.10, only monomial cost functions observe this property.

Originally, the resource augmentation model was introduced to analyze the performance of *online algorithms*. On contrast to the above *offline model*, where all information is assumed to be a priori given, in the *online model* the information appears over time. The performance of online algorithms is commonly measured by their *competitive ratio*. An online algorithm is said to be *c-competitive* if the objective function value of its solution never exceeds the value of an offline optimum by a factor of c. In terms of the resource augmentation model, the online analog are *s-speed c-competitive* algorithms.

1.2.2 Solvable cases of $1 \,||\, \sum w_j f(C_j)$, order rules, and index algorithms

In this and the following sections, we will report on related work, starting with the very few cases which allow for exact polynomial-time algorithms. Considering the general problem $1 \,||\, \sum w_j f(C_j)$ with either uniform processing times or uniform weights, optimal schedules are achieved by sorting the jobs in non-decreasing order of their weights or in non-increasing order of their processing times, respectively. This can be observed by obvious exchange arguments.

In case of general processing times and weights, there are two classes of cost functions for which exact polynomial-time algorithms are known, linear and exponential functions. Their property of being *memoryless*—i.e., shifting a whole schedule in time does not change optimality—forms the basis of the corresponding algorithms. In order to explain them in a convenient way, we introduce the notions of *comparability* and *order rules* which we will also largely use in Chapter 3.

Definition 1.1 (Comparabilty) *Let i and j be two jobs in an instance of $1 \,||\, w_j f(C_j)$ with processing times p_i, p_j and weights w_i, w_j, respectively. We say that i* locally precedes *j, if for every point in time $t \geq 0$ it holds that*

$$w_i f(t + p_i) \;+\; w_j f(t + p_i + p_j) \;\leq\; w_j f(t + p_j) \;+\; w_i f(t + p_j + p_i)\,, \tag{1.1}$$

which we denote by $i \preccurlyeq_\ell j$. If $i \preccurlyeq_\ell j$ or $j \preccurlyeq_\ell i$, then we call i and j locally comparable*, otherwise* locally incomparable*. Moreover, if $i \preccurlyeq_\ell j$ and $j \preccurlyeq_\ell i$, then we refer to i and j as* locally equal*, which we denote by $i =_\ell j$.*

More generally, i globally precedes *j if for every point in time t and every subsequence of jobs $\ell_1, \ldots, \ell_s$ it holds that*

$$\begin{aligned} w_i f(t + p_i) \;+\; \sum_{r=1}^{s} w_{\ell_r} f\left(t + p_i + \sum_{q=1}^{r} p_{\ell_q}\right) \;+\; w_j f\left(t + p_i + \sum_{r=1}^{s} p_{\ell_r} + p_j\right) \\ \leq\; w_j f(t + p_j) \;+\; \sum_{r=1}^{s} w_{\ell_r} f\left(t + p_j + \sum_{q=1}^{r} p_{\ell_q}\right) \;+\; w_i f\left(t + p_j + \sum_{r=1}^{s} p_{\ell_r} + p_i\right), \end{aligned}$$

The notations $i \preccurlyeq_g j$ and $i =_g j$ are defined analogously to the local variants, and so are the notions of global comparability, incomparability and equality.

Note that if $i \preccurlyeq_\ell j$ and job j directly precedes job i, then swapping i and j does not increase the cost of the schedule. Hence, for any two jobs i and j for which $i \preccurlyeq_\ell j$ but not $j \preccurlyeq_\ell i$, we can assume w.l.o.g. that they are scheduled in the order i, j in an optimal schedule if occurring consecutively. By introducing appropriate tie breaking rules for the case $j =_\ell i$, we can even assume that every locally comparable job pair appears in some fixed local order in optimal schedules. The analogue assumptions can be made in the global case. In the following definition of *order rule*, we assume the tie breaking decision to be already taken.

Definition 1.2 (Order rules) *A function $R_f : \mathbb{R}^4_{\geq 0} \to \{\preccurlyeq_\ell, \succcurlyeq_\ell, ?\}$ is a* local order rule *for cost function f, if for any two jobs i and j with processing times p_i, p_j and weights w_i, w_j, respectively, $R_f(p_i, w_i, p_j, w_j) = \preccurlyeq_\ell$ implies that the local order i, j never contradicts optimality, and analogously j, i in case of $\succcurlyeq_\ell$. If the function value is ?, the rule does not imply a local ordering for the particular job pair.* Global order rules *are defined analogously.*

Turning back to linear and exponential cost functions, we can now observe that any pair of jobs is locally comparable: When changing the parameter t in (1.1), both sides of the inequality change by the same additive term in case of linear functions, while changing by the same factor in case of exponential functions. Due to the local comparability being satisfied for *every* job pair, the same order relations hold true on a global level, and thus, they define an optimal schedule.

For linear cost functions $t \mapsto c \cdot t$ with $c > 0$ and for exponential cost functions $t \mapsto a^t$ with $a > 1$, the local comparability inequality (1.1) translates into

$$\frac{w_i}{p_i} > \frac{w_j}{p_j} \qquad \text{and} \qquad \frac{w_i\, a^{p_i}}{a^{p_i} - 1} > \frac{w_j\, a^{p_j}}{a^{p_j} - 1},$$

respectively, which implies natural optimal algorithms for the two problems. One simply sorts the jobs in non-increasing order of the values w_j/p_j and $w_j\, a^{p_j}/(a^{p_j} - 1)$, respectively. These results were published by Smith [Smi56] and Rothkopf [Rot66], respectively.

The two algorithms lead to the general class of *index algorithms*. Even though our main problem $1 \,||\, \sum w_j f(C_j)$ assumes uniform release dates, we formulate this algorithm class in the more general form for non-uniform release dates as this variant has been investigated in related work.

Alg. 1.1: Index algorithm

Input: Jobs J with processing times $p_j \geq 0$, weights $w_j \geq 0$ and release dates $r_j \geq 0$ for every $j \in J$;
an index function $\delta : \mathbb{R}^2_{\geq 0} \to \mathbb{R}_{\geq 0}$, $(p_j, w_j) \mapsto \delta(p_j, w_j) =: \delta_j$.

Output: Preemptive single machine schedule.

Whenever the machine becomes idle or a new job is released, schedule an available, unscheduled job j with maximum index δ_j w.r.t. its remaining processing time.

Note that the index of a job solely depends on its processing time and weight, and it is completely independent of a (partially) chosen schedule. Moreover, note that, in case of uniform release dates, index algorithms will always produce non-preemptive schedules which are the result of sorting the jobs according to their indices.

The two algorithms mentioned above can now be formally described as index algorithms with index functions $\delta(p_j, w_j) = w_j/p_j$ and $\delta(p_j, w_j) = w_j\, a^{p_j}/(a^{p_j} - 1)$, respectively, where the former is—for the obvious reason of being analyzed by him—well known as *Smith's rule*. As the ratio w_j/p_j is commonly referred to as *density*, this algorithm is also called *Highest Density First* in the literature.

Alg. 1.2: Smith's rule / Highest Density First (HDF)

Input:	Jobs J with processing times $p_j \geq 0$, weights $w_j \geq 0$ and release dates $r_j \geq 0$ for every $j \in J$; index function $\delta : (p_j, w_j) \mapsto w_j/p_j$.
Output:	Preemptive single machine schedule computed with index algorithm w.r.t. to index function δ.

After having seen these results for linear and exponential cost functions, a natural question is whether also other cost functions allow for optimal index algorithms. However, Rothkopf and Smith[1] [RS84] showed that these two classes of functions are in fact the only ones which observe optimal index algorithms. The basis of the proof is the observation that such algorithms can only exist if the local order of any pair of jobs is independent of the time they are scheduled. Otherwise, one could easily construct an instance where the respective jobs have to scheduled in the reverse order of the index algorithm. Finally, this observation leads to a differential equation which has exactly two solutions, linear and exponential functions.

1.2.3 Linear cost functions and non-uniform release dates

Knowing from the above that the linear cost case $1 \,||\, \sum w_j\, C_j$ is in P for uniform release dates, we will now briefly discuss results for the more general problem variant with non-uniform release dates. This problem is well-understood in both the non-preemptive and preemptive case. Both cases are strongly NP-hard as shown by Lenstra, Rinnooy Kan and Brucker [LRKB77] and Labetoulle et al. [LLLRK84], respectively. While in the former case, the hardness is proven for uniform weights, Baker [Bak74] showed that the latter case is in P under this assumption. Afrati et al. [ABC+99] proposed PTASs for many NP-hard problem variants with and without preemption.

Regarding the related flow time objective $\sum w_j\, F_j$, NP-hardness results from the corresponding completion time problems carry over directly since optimal solutions with respect to these two objectives coincide. In contrast, approximability results cannot be transferred. Kellerer, Tautenhahn and Woeginger [KTW99] gave an $O(\sqrt{n})$-approximation algorithm for the non-preemptive unweighted case, and they showed that this factor is basically best possible. Utilizing resource augmentation, Bansal et al. [BCK+07] were able to design a 12-speed 4-approximation algorithm for non-uniform weights. The preemptive unweighted case is well-known to be solved by the *Shortest Remaining Processing Time* rule. For non-uniform weights, Chekuri and Khanna [CK02] proposed a QPTAS for $P := \max_{ij} p_i/p_j$ and $W := \max_{ij} w_i/w_j$ being polynomially bounded in n. Moreover, Becchettia et al. [BLMSP06] show that Smith's rule is a $(1+\varepsilon)$-speed $(1+\frac{1}{\varepsilon})$-competitive online algorithm.

[1] This is Stephen A. Smith in contrast to Wayne E. Smith, the author of Smith's rule.

1.2.4 Non-linear cost functions on completion times

After the linear cost case, we now turn to the setting $1 \,||\, \sum w_j f(C_j)$ in which all jobs share the same non-linear cost function. Apart from those polynomial-time algorithms described above, the only insight on the complexity we are aware of is the weak NP-hardness for general convex cost functions. It follows from a result of Yuan [Yua92] who proved the problem to be weakly NP-hard for tardiness cost $f : t \mapsto \max\{0, t - d\}$ with common due date d. This is shown by a reduction from the weakly NP-hard Partition problem. Megow and Verschae [MV13] showed that the problem remains NP-hard when changing the cost before the deadline from zero to a linear function of time with a sufficiently small slope $\varepsilon > 0$, which follows by a reduction from the problem variant considered by Yuan. However, this leaves open several questions concerning the complexity of the problem, see Chapter 5.

For arbitrary concave f, Stiller and Wiese [SW10] show that Smith's rule guarantees an approximation factor of $(\sqrt{3}+1)/2 \approx 1.366$. The result is tight in the sense that for a certain cost function f there is a problem instance where this factor is attained by Smith's rule. As the first steps of our analysis in Section 2.2.1 are analogously to those of Stiller in Wiese, they are discussed in that section in more detail. For general monotone cost functions, Epstein et al. [ELMS+12] provide a $(4 + \varepsilon)$-approximation algorithm, which generates *one* schedule that observes the respective performance guarantee for *any* cost function. The analysis is based on an elaborate lower bound involving a machine capacity function which measures the aggregated amount of processing time being available up to a certain point in time. Both algorithms, the one by Epstein et al. and Smith's rule, yield schedules that are *universal* in the sense of being generated without knowledge of the cost function.

Regarding non-universal algorithms, Megow and Verschae [MV13] propose a PTAS for general cost functions, and an FPTAS for piece-wise linear cost functions with a constant number of linear segments. The results are based on a novel approach of applying rounding techniques to the weight dimension instead of to the time dimension, where such techniques are commonly used.

For the more general setting with non-uniform release dates and preemption, Stiller and Wiese [SW10] extend the PTAS of Afrati et al. [ABC+99] from linear to concave cost. Furthermore, Fisher and Krieger [FK84] show that the maximization variant of the problem $1 \,||\, \sum w_j C_j^2$ with quadratic cost, admits an approximation factor of 3/2.

1.2.5 Non-linear cost functions on flow times

Focusing on the flow time based problem $1 \,|\, r_j, \text{pmtn} \,|\, \sum w_j f(F_j)$ with non-uniform release dates and preemption, Moseley et al. [MPS13] showed very recently strong NP-hardness. They proved that even the restricted problem $1 \,|\, r_j, \text{pmtn} \,|\, \sum F_j^k$ with uniform weights and monomial cost functions with degree $k \in (0, 1)$ and $k \in \mathbb{N}_{\geq 2}$ is hard which follows by an elaborate reduction from 3-Partition.

Turning to the online setting of the problem $1 \,|\, r_j, \text{pmtn} \,|\, \sum F_j^k$, Bansal and Pruhs [BP10b] showed that there exists no $n^{o(1)}$-competitive online algorithm which approximates the optimum simultaneously for all degrees $k > 1$. However, utilizing resource augmentation, they were able to prove that Smith's rule is in fact $(1 + \varepsilon)$-speed $O(\frac{1}{\varepsilon^{2k}})$-competitive even for the weighted objective $\sum w_j F_j^k$. In this analysis, the cost of an online schedule produced with Smith's rule are mapped to parts of the cost of

an offline optimum, thereby bounding the cost, and thus, the competitive ratio. Utilizing a similar technique, Im, Moseley, and Pruhs [IMP12] showed Smith's rule to be $(2+\varepsilon)$-speed $O(\frac{1}{\varepsilon})$-competitive for general cost functions f. Moreover, they argued that Smith's rule performs basically best possible in terms of universal algorithms. They showed that there exists no $(2-\varepsilon)$-speed $O(1)$-competitive universal online algorithm for any fixed $\varepsilon > 0$. Furthermore, they proved that any $O(1)$-competitive online algorithm, including non-universal algorithms, requires some speed augmentation larger than $\frac{7}{6} - \varepsilon$.

Finally note that, as observed in Section 1.2.1, for monomial cost functions and uniform release dates, an s-speed schedule of cost c translates into a 1-speed schedule of cost $s^k c$. In this way, the above results also imply classic approximation results for this setting.

1.2.6 Job-specific non-linear cost.

In contrast to the setting with one common global cost function, where the complexity is not very well understood, it is straightforward to observe that the problem is strongly NP-hard when considering individual cost functions of the jobs. In this case, we can simply model the total weighted tardiness objective $\sum w_j \max\{C_j - d_j, 0\}$ with job-specific deadlines d_j for which the problem is known to be strongly NP-hard [Law77].

Considering flow times, Bansal and Pruhs [BP10a] interpreted the problem $1 \,|\, r_j, \text{pmtn} \,|\, \sum f_j(C_j)$ as geometric set cover problem. Showing that approximation algorithms for this problem transfer to the scheduling problem with an increase of the approximation guarantee by factor 4, they focus on the set cover problem for which they derive approximation results via LP rounding. In the end, this leads to an $O(\log\log p_{\max})$-approximation for the scheduling problem, where $p_{\max}$ denotes the maximum processing time under the assumption of general integer processing times. In case of uniform release dates, the set cover problem reduces to a generalized caching problem for which a 4-approximation is known, yielding a 16-approximation for the problem $1 \,||\, \sum f_j(C_j)$. Cheung and Shmoys [CS11] claimed to achieve a $(2+\varepsilon)$-approximation via a primal-dual approach using cover inequalities. However, Mestre and Verschae [MV14] showed that there exist instances where the algorithm constructs a dual solution whose value differs from the optimal integral solution cost by a factor of 4. Moreover, Mestre and Verschae showed that the local-ratio interpretation of the algorithm by Cheung and Shmoys is in fact a pseudopolynomial time 4-approximation, yielding a $(4+\varepsilon)$-approximation in polynomial time.

1.2.7 Exact algorithms for quadratic cost functions

Finally, we report on work that has been done in the area of exact algorithms. Most of the results in the literature focus on the special case of quadratic cost functions—recall that the complexity of this special case is still open. Half a century ago, Schild and Fredman [SF62] proposed a complete enumeration scheme to solve this problem utilizing a local order rule to reduce the solution space. Nearly two decades later, Townsend [Tow78] was the first to solve problem $1 \,||\, \sum w_j C_j^2$ by a branch-and-bound algorithm. In order to prune the search space, he proposed a lower bound on the optimal cost related to a local order rule, which we discuss in more detail in Section 3.3.1. This initial article triggered a long series of papers proposing many improvements of the branch-and-bound algorithm.

Bagga and Kalra [BK80] add a further node elimination rule by giving a sufficient condition for sets of jobs appearing at the first r positions in an optimal schedule. Global order rules in this context were first used by Gupta and Sen[2] [GS84], who proved the sufficient condition $p_j > p_i \ \wedge \ w_j\, p_j < w_i\, p_i$. Order rules that additionally depend on the time interval within which the job pair is processed are used by Sen[2], Dileepan and Ruparel [SDR90] and Della Croce et al. [DCST+95]. Finally, Mondal and Sen[2] [MS00] conjectured the global order rule $w_i \geq w_j \wedge w_i/p_i \geq w_j/p_j$ and demonstrated in an experimental setup that this rule would significantly further reduce the set of potentially optimal solutions. An overview of the order rules known so far can be found in Figure 3.2 (a) in Section 3.2.

In the original branch-and-bound method by Townsend, nodes represent prefixes of schedules. Sen[2], Bagchi and Ramaswamy [SBR96] propose to reduce the number of generated nodes by using a graph searching procedure. There, nodes alternatively represent unordered sets of jobs, and a schedule is then represented by a path in that graph. Kaindl, Kainz and Radda observe in [KKR01] that it is more efficient to build the schedule from the end to the beginning than vice versa. As we adopt these ideas, a more detailed description of these algorithms can be found in Section 3.3.1.

In contrast to the above order rules, which follow Definition 1.2, Szwarc [Szw98] proposed a decomposition rule which involves information of the entire instance. It deduces global order relations from local ones or from order relations that are even restricted to specific time intervals. We will discuss the rule in more detail in Section 3.2.1.

[2]It was Tapan Sen who was involved in [GS84, SDR90] while Anup K. Sen is author of [SBR96, MS00].

Chapter 2

Analysis of Smith's rule for convex and concave cost functions

In this chapter, we investigate the performance of Smith's rule in single machine scheduling with generalized min-sum objective. By explicitly constructing worst-case instances, we provide a tight analysis of the approximation guarantee of Smith's rule under any particular convex or concave cost function. More specifically, for this wide class of cost functions we reduce the task of determining a worst case problem instance to a continuous optimization problem, which can be solved by standard algebraic or numerical methods. For polynomial cost functions with positive coefficients it turns out that the tight approximation ratio can be calculated as the root of a univariate polynomial. We show that this approximation ratio is asymptotically equal to $k^{(k-1)/(k+1)}$, denoting by k the degree of the cost function. To overcome unrealistic worst case instances, we also give tight bounds for the case of integral processing times that are parameterized by the maximum and total processing time.

Publication remark: The results in this chapter are based on joint work with Tobias Jacobs, partially published in the Proceedings of the *10th Latin American Symposium on Theoretical Informatics* (LATIN 2012) [HJ12b].

Originally published in 1956 for minimizing the weighted sum of completion times on a single machine [Smi56], Smith's rule is still one of the standard algorithms being analyzed for new (online) problem variants in scheduling, see e.g., [BP10b, IMP12, SW10]. Its intuitive idea of scheduling important short jobs before unimportant long jobs makes it to a natural approach in many problem settings.

Stiller and Wiese have shown that Smith's rule is $(\sqrt{3}+1)/2$-approximative for any concave cost function [SW10]. In contrast, when addressing general convex functions, the approximation factor can get arbitrarily bad as we will see later on. In order to still provide a meaningful analysis of the performance of Smith's rule, we focus on the setting in which the cost function is some arbitrary but fixed convex or concave function. A tight approximation factor is then computed in dependency of this function. The worst case problem instances that establish these factors turn out to be rather artificial in the sense of containing one very long job and several infinitesimal small jobs. Aiming for an additional, more realistic analysis, we provide tight approximation factors for a problem variant which is parameterized by the maximum processing time and the total processing time under the assumption of a discrete time domain.

The approximation factors of the general analysis can be obtained as the solution of a continuous optimization problem with at most two degrees of freedom. In case of cost functions that are polynomials with positive coefficients, it will turn out that the

approximation factor can be calculated simply by determining the root of a univariate polynomial. In fact, this factor solely depends on the degree of the polynomial. For instance, for cost functions of degree 2 and 3 this results in factors 1.31 and 1.76, respectively. Regarding universal scheduling methods—that compute solutions without knowledge of the cost function—Smith's rule provides the best known approximation factor for polynomials up to degree six. Moreover, our analysis implies that Smith's rule guarantees an approximation factor of less than 1.25 for the problem of minimizing the L_k-norm of the total weighted completion time cost $(\sum w_j C_j^k)^{1/k}$ for any $k > 1$. Besides the implicit description as solution of a continuous optimization problem, we also give lower and upper bounds on the approximation factor. These bounds show that for polynomial cost functions of degree k, the tight asymptotic approximation factor is $k^{(k-1)/(k+1)}$.

This chapter is organized as follows. In Section 2.1, we start with a simple combinatorial analysis of Smith's rule for polynomial cost functions, upper bounding the approximation factor by the degree of the polynomial. As we will see afterwards, this bound is not tight. In Section 2.2 we provide the tight analysis of Smith's rule for arbitrary but fixed convex and concave cost function, where in Section 2.2.1 we discuss the analysis in its general form while further simplifying it for polynomials with positive coefficients in Section 2.2.2. The lower and upper bounds on the approximation factor for polynomials are provided in Section 2.2.3. Finally, the parameterized analysis is discussed in Section 2.3.

In the following, for brevity, we will often refer to Smith's rule by its alternative name Highest Density First *(HDF)*. Recall that Smith's rule was formally described in Algorithm 1.2.

2.1 Combinatorial analysis of Smith's rule

In this section, we provide a first combinatorial analysis of Smith's rule. This analysis is not tight and it is restricted to cost functions that observe a so called *local β-gap rule*. However, it is much more intuitive than the tight analysis provided in Section 2.2, and it yields additional structural insights into the problem. For cost functions that are polynomials with non-negative coefficients, this analysis proves an approximation factor equal to the degree of the cost function.

In Section 1.2.2, we have already introduced the general concept of order rules, which we will investigate in more detail in Chapter 3. In that chapter, we will also define the *local β-gap rule*. A cost function observes this rule if, whenever two jobs i and j with $w_i/p_i \geq \beta \cdot w_j/p_j$ are scheduled consecutively in an optimal schedule, i is scheduled before j. Conversely, when they are not scheduled in this order then the cost can be decreased by swapping i and j. Recall that the term *local* refers to the fact that this order relation only holds if the jobs are scheduled consecutively, but not necessarily when there are other jobs scheduled in between.

Theorem 2.1 *Consider problem $1 \,||\, \sum w_j f(C_j)$ for a non-decreasing cost function f. If f observes a local β-gap rule, then Smith's rule is a factor β-approximation algorithm for this problem, no matter how ties are broken.*

Proof. For the sake of simplicity we assume that the jobs are labeled in the order they are scheduled by Smith's rule. We refer to that schedule simply by HDF and also consider an optimal schedule OPT. The proof technique is to analyze the increase in cost that occurs during a transformation from OPT to HDF. Slightly abusing notation, we refer to the cost of these schedules also by OPT and HDF.

The transformation proceeds as follows: Starting from OPT, repeatedly interchange job n with its right neighbor until it has become the very last job. Then perform a corresponding series of local interchanges to move job $n-1$ to its position in HDF, and so on. During this transformation process, each job i first moves to the left by being interchanged with some jobs j with $j > i$, then it moves to the right by being interchanged with some other jobs j with $j < i$. Once i is at its HDF position, it does not move anymore.

In each single local operation some pair of jobs j, i with $j > i$ is interchanged, such that afterwards it is scheduled in the order i, j. During this operation, the completion time of i decreases by p_j and the completion time of j increases by p_i, therefore the cost of i decreases by some $\Delta_{ij}^- > 0$ while the cost of j increases by some $\Delta_{ij}^+ > 0$. By the ordering of Smith's rule, we know that $w_i/p_i \geq w_j/p_j$, which implies that

$$\frac{\beta\, w_i}{p_i} \;\geq\; \beta \cdot \frac{w_j}{p_j}\,.$$

In other words, if w_i were by a factor of β larger, then the sufficient condition of the β-gap rule would hold and the interchange operation would decrease the schedule's cost. Making w_i by a factor of β larger would cause also Δ_{ij}^- to be larger exactly by a factor of β. It follows that $\beta\Delta_{ij}^- > \Delta_{ij}^+$ for each pair i, j of jobs that are interchanged at some point of the transformation process. Let M be the set of all pairs (i, j) of interchanged jobs $i < j$ in the transformation. Summing up for all elements of M, we obtain

$$\sum_{(i,j)\in M} \Delta_{ij}^+ \;<\; \beta \cdot \sum_{(i,j)\in M} \Delta_{ij}^-\,. \tag{2.1}$$

Furthermore, during the transformation process each job first moves a number of times left and then a number of times right in the schedule. Each time moving left, the job becomes responsible for some value Δ_{ij}^-. As no job can move to a position smaller than 0, the Δ_{ij}^--values corresponding to any particular job sum up to at most the cost of that job in OPT. Hence, for the sum of all Δ_{ij}^--values of all jobs it holds that

$$\sum_{(i,j)\in M} \Delta_{ij}^- \;\leq\; \mathrm{OPT}\,. \tag{2.2}$$

From equation (2.1) and (2.2) we obtain the approximation factor as follows:

$$\mathrm{HDF} \;-\; \mathrm{OPT} \;=\; \sum_{(i,j)\in M} \Delta_{ij}^+ \;-\sum_{(i,j)\in M} \Delta_{ij}^- \;<\; (\beta-1)\cdot \sum_{(i,j)\in M} \Delta_{ij}^- \;\leq\; (\beta-1)\cdot \mathrm{OPT}\,.$$

□

As we will show in Lemma 3.4, the local β-gap rule is observed by polynomial cost functions with non-negative coefficients for β being the cost function's degree. This leads directly to the following corollary.

Corollary 2.2 *For problems $1 \,||\, \sum w_j\, f(C_j)$ with cost function f being a polynomial with non-negative coefficients and degree $k \in \mathbb{Q}$, Smith's rule is a k-approximation algorithm.*

2.2 Tight analysis of Smith's rule

In this section we present a tight analysis of the worst case approximation factor obtained by Smith's rule for any arbitrary but fixed convex or concave cost function. We start with a simple observation that will be used throughout this chapter.

Observation 2.3 *Problem* $1\,||\,\sum w_j\,f(C_j)$ *is invariant to weight scaling, i.e., if* I *is a problem instance and* I' *is obtained from* I *by multiplying all job weights with a constant* c*, then the cost of any schedule for* I' *is* c *times the cost of the same schedule for* I*.*

We denote by $\mathrm{HDF}(I)$ the schedule computed for instance I by Smith's rule and by $\mathrm{OPT}(I)$ an optimal schedule for I. Again, slightly abusing notation, the cost of these schedules will also be denoted by $\mathrm{HDF}(I)$ and $\mathrm{OPT}(I)$.

2.2.1 General analysis

We start with an analysis for general convex or concave cost functions which we will then simplify for certain polynomials in the subsequent subsection. The main result of this section is given in the following theorem.

Theorem 2.4 *Considering a fixed convex cost function* f*, the tight approximation ratio* α_f *of Smith's rule for problem* $1\,||\,\sum w_j\,f(C_j)$ *is given by*

$$\alpha_f \;=\; \sup\left\{\frac{\mathrm{HDF}(I)}{\mathrm{OPT}(I)}\right\} \;=\; \sup\left\{r_f(p,q)\mid p\geq 0, q\geq 0\right\} \tag{2.3}$$

where

$$r_f(p,q) \;:=\; \frac{\int_0^q f(t)dt + p\cdot f(q+p)}{p\cdot f(p) + \int_p^{p+q} f(t)dt}\;. \tag{2.4}$$

When f *is concave, the tight ratio is* $\sup\{1/r_f(p,q)\mid p\geq 0, q\geq 0\}$*. The approximation factors hold regardless of the tie breaking strategy used by Smith's rule.*

As a first consequence of this theorem, we observe that Smith's rule can perform arbitrarily bad for exponential cost functions $f: t\mapsto e^t$. Choosing $q := \ln p$, it follows

$$r_f(p,q) \;=\; \frac{e^q - 1 + p\cdot e^q\cdot e^p}{p\cdot e^p + e^q\cdot e^p - e^p} \;=\; \frac{p^2\cdot e^p + p - 1}{(2\,p-1)\cdot e^p} \;\xrightarrow[p\to\infty]{}\; \infty\,.$$

This implies immediately the corollary below. However, recall that an alternative index rule solves the problem $1\,||\,\sum w_j\,f(C_j)$ with exponential cost to optimality [Rot66].

Corollary 2.5 *For exponential cost functions* f *it holds that* $\alpha_f = \infty$*.*

In what follows, we prove a series of lemmas which successively narrow the space of instances we need to consider when searching for a worst case problem instance for Smith's rule. Determining the worst case solution in the final instance space will then be shown to be equivalent to the continuous optimization problem described in the above theorem.

Very similar to the analysis of Stiller and Wiese [SW10], we show that it is sufficient to consider instances with uniform density $\frac{w_j}{p_j}$ (Lemma 2.7), and that a most expensive

schedule is obtained by inverting the optimal job order (Lemma 2.8). Thereafter, again as Stiller and Wiese, we restrict to instances with several small jobs and one large job (Lemma 2.9). However, their proof of this property is based on a modification of the cost function which makes it invalid for our problem setting. Our proof follows a completely different line in this main part of argumentation.

As a first step, we show that the approximation factor of Smith's rule is independent of the tie breaking policy. The proof is based on continuity considerations which follow the general principle described in [Jac12].

Lemma 2.6 *If considering $1 \,||\, \sum w_j\, f(C_j)$ with continuous cost function f, then the approximation factor α_f is independent of the tie breaking policy employed by Smith's rule.*

Proof. For any $n \geq 1$, let $\alpha_{f,n}$ be the largest approximation factor achieved for any instance with n jobs, i.e., $\alpha_{f,n} := \sup\{\mathrm{HDF}(I)/\mathrm{OPT}(I) \mid I \text{ has } n \text{ jobs}\}$. It suffices to show that $\alpha_{f,n}$ is independent of the tie breaking policy for any n.

We interpret problem instances I with n jobs as vectors $(w_1, p_1, w_2, p_2, \ldots, w_n, p_n)$ in $\mathbb{R}^{2n}$. As f is continuous, the cost of a fixed schedule S for instance I is continuous in I. Again abusing notation, we also denote this cost by $S(I)$. The cost $\mathrm{OPT}(I) = \min\{R(I) \mid R \text{ is a schedule for } I\}$ is continuous in I as well, and therefore also $S(I)/\mathrm{OPT}(I)$ for any fixed S.

Let HDF_a and HDF_b be two variants of Smith's rule, differing in their tie breaking rule. We will show that for any instance I there is a series of instances $(I_k)_{k\in\mathbb{N}}$ with

$$\lim_{k\to\infty} \frac{\mathrm{HDF}_a(I_k)}{\mathrm{OPT}(I_k)} \;=\; \frac{\mathrm{HDF}_b(I)}{\mathrm{OPT}(I)}\,.$$

As HDF_a and HDF_b have been chosen arbitrarily, this claim then also holds with HDF_a and HDF_b interchanged, which proves the lemma. The series $(I_k)_{k\in\mathbb{N}}$ is chosen such that:

(i) no element I_k contains two jobs with equal density,
(ii) the series converges against I, and
(iii) HDF_b outputs the very same schedule for I as it does for each I_k.

These requirements are easy to achieve by adding some suitable multiples of ε to the job weights, and then letting ε converge to zero as k goes to infinity. From the continuity considerations above and the fact that HDF_a and HDF_b produce the same schedule for each I_k due to (i), it follows that $\lim_{k\to\infty} \mathrm{HDF}_a(I_k) = \lim_{k\to\infty} \mathrm{HDF}_b(I_k) = \mathrm{HDF}_b(I)$. □

As a consequence of Lemma 2.6, we can assume that Smith's rule always breaks ties in the worst possible way. In terms of an adversary model, we can assume that the adversary not only chooses the problem instance, but also is allowed to choose the way Smith's rule breaks ties.

The next lemma shows that we can restrict our attention to problem instances where the density is the same for all jobs. By the argument given above, the adversary can freely choose the schedule HDF in such instances.

Lemma 2.7 *For the problem $1 \,||\, \sum w_j\, f(C_j)$ with cost function f, there exists an instance of unit density that achieves the worst case performance ratio α_f of Smith's rule for this cost function. More formally,*

$$\alpha_f \;=\; \sup\left\{ \frac{\mathrm{HDF}(I)}{\mathrm{OPT}(I)} \,\middle|\, w_j = p_j \text{ for each job } j \in I \right\}.$$

Proof. We show that any instance I of $1\,||\sum w_j f(C_j)$ can be transformed into an instance I' having density $\frac{w_j}{p_j} = 1$ for each job j and satisfying $\mathrm{HDF}(I')/\mathrm{OPT}(I') \geq \mathrm{HDF}(I)/\mathrm{OPT}(I)$. It suffices to show the existence of a suitable instance I' with the density being the same for each job, because then, density 1 can be obtained by weight scaling as argued in Observation 2.3.

We can assume w.l.o.g. that $p_j > 0$ for all jobs j. Jobs j with $p_j = 0$ are scheduled by both HDF and OPT at the very beginning of the schedule. Since they neither account for cost nor they delay other jobs, they can be simply removed from the instance. Similarly, we can argue that jobs j with $w_j = 0$ can be removed (or added) without changing the total cost, because both algorithms schedule them after all other jobs at zero cost. However, for the simplicity of the proof, we assume for now that there exists at least one job j with $w_j = 0$, and at the end of our transformation, we remove all zero weight jobs.

We partition the jobs of I into classes according to their density. For $x \geq 0$, the density class d_x is defined to contain all jobs j of I with $\frac{w_j}{p_j} = x$. By this definition, all jobs of weight zero are contained in the the density class d_0, which is non-empty by assumption. If there exists only one ratio class besides d_0, then the lemma follows by simply deleting all jobs from d_0.

If there are more than two non-empty ratio classes, we show how to reduce their number without decreasing the approximation factor. More specifically, we will show that there is an instance I' having one non-empty density class less than I, and $\mathrm{HDF}(I')/\mathrm{OPT}(I') \geq \mathrm{HDF}(I)/\mathrm{OPT}(I)$. After at most n applications of that argument we arrive at the desired instance that has only a single non-empty density class in addition to d_0.

Let C_j be the completion time of job j in $\mathrm{HDF}(I)$, and let C_j^* be the corresponding completion time in $\mathrm{OPT}(I)$. Let further $X := \{x \mid d_x \neq \emptyset\}$ be the family of densities that occur in I. For $x \in X$ we define the contribution of density class d_x to the cost of $\mathrm{HDF}(I)$ and $\mathrm{OPT}(I)$ as

$$\mathrm{HDF}_x(I) := \sum_{j \in d_x} w_j\, f(C_j) \qquad \text{and} \qquad \mathrm{OPT}_x(I) := \sum_{j \in d_x} w_j\, f(C_j^*)\,,$$

respectively. The approximation factor of instance I can be recalculated as

$$\frac{\mathrm{HDF}(I)}{\mathrm{OPT}(I)} \;=\; \frac{\sum_{x \in X} \mathrm{HDF}_x(I)}{\sum_{x \in X} \mathrm{OPT}_x(I)}\,.$$

We now choose one density class d_x with $x \neq 0$ satisfying

$$\frac{\mathrm{HDF}_x(I)}{\mathrm{OPT}_x(I)} \leq \frac{\mathrm{HDF}(I)}{\mathrm{OPT}(I)} \tag{2.5}$$

—which exists by the pigeonhole principle. We define $y := \max\{z \mid z \in X, z < x\}$ as the density next smaller than x. Such a y always exists as $x > 0$ and d_0 is not empty.

The instance I' is now constructed as follows. The processing time p'_j of each job $j \in I'$ is the same as the processing time p_j of the job $j \in I$. For each $j \notin d_x$, we choose the original weight $w'_j := w_j$. As for the jobs $j \in d_x$, we define $w'_j := p'_j \cdot y$, so that in I' these jobs become part of the density class d_y.

Aiming to bound the ratio $\mathrm{HDF}(I')/\mathrm{OPT}(I')$, we denote by $\mathrm{HDF}_I(I')$ and $\mathrm{OPT}_I(I')$ the schedules for instance I' which run the jobs in the same order as their counterparts

are scheduled in $\mathrm{HDF}(I)$ and $\mathrm{OPT}(I)$, respectively. Firstly, we observe that the schedule $\mathrm{HDF}_I(I')$ is feasible according to HDF since $\frac{w_i}{p_i} > \frac{w_j}{p_j}$ implies $\frac{w_{i'}}{p_{i'}} \geq \frac{w_{j'}}{p_{j'}}$ for any two jobs i and j. Moreover, by Lemma 2.6, we can assume $\mathrm{HDF}(I')$ to be the most costly among all HDF schedules for I', and thus,

$$\mathrm{HDF}(I') \;\geq\; \mathrm{HDF}_I(I') \;=\; \mathrm{HDF}(I) - (1 - \tfrac{y}{x}) \cdot \mathrm{HDF}_x(I) \;.$$

Considering the schedule $\mathrm{OPT}(I')$, clearly no other schedule can underprice its cost, and thus,

$$\mathrm{OPT}(I') \;\leq\; \mathrm{OPT}_I(I') \;=\; \mathrm{OPT}(I) - (1 - \tfrac{y}{x}) \cdot \mathrm{OPT}_x(I) \;.$$

Note that $\mathrm{OPT}(I')$ is strictly positive, as d_x was not the only non-empty density class besides d_0. Hence, equation (2.5) implies that

$$\frac{\mathrm{HDF}(I')}{\mathrm{OPT}(I')} \;\geq\; \frac{\mathrm{HDF}(I) - \left(1 - \frac{y}{x}\right) \cdot \mathrm{HDF}_x(I)}{\mathrm{OPT}(I) - \left(1 - \frac{y}{x}\right) \cdot \mathrm{OPT}_x(I)} \;\geq\; \frac{\mathrm{HDF}(I)}{\mathrm{OPT}(I)} \;. \qquad \square$$

We now know that for analyzing the worst case approximation factor of Smith's rule we can restrict our attention to problem instances with unit density, and due to Lemma 2.6 we can further assume that Smith's rule schedules the jobs in the worst possible order while OPT schedules them in the best possible order. When the cost function is concave or convex, there is a very simple characterization of these special orders, as shown in the following lemma.

In the remainder of this section, the proofs are given for convex cost functions only, even though holding for concave functions as well. However, by syntactically replacing all terms marked with * with their opposite, one obtains the proof for the concave case. Examples of such opposite pairs are convex/concave, best/worst, non-positive/non-negative, $\geq$ / $\leq$ or $>$ / $<$.

Lemma 2.8 *Considering problem* $1 \,||\, \sum w_j f(C_j)$ *with convex cost function* f*, the tight approximation factor is*

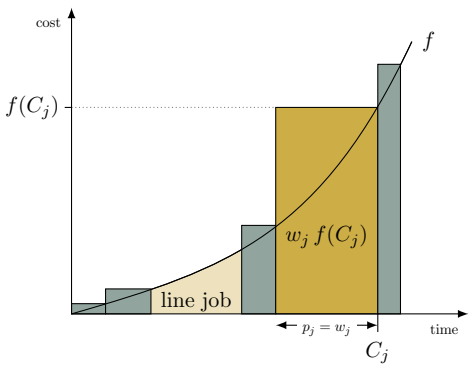
$$\alpha_f \;=\; \sup \left\{ \frac{\mathrm{INC}(I)}{\mathrm{DEC}(I)} \,\middle|\, w_j = p_j \text{ for each job } j \in I \right\}$$

where $\mathrm{INC}(I)$ *and* $\mathrm{DEC}(I)$ *denotes the cost of the schedule where the jobs in* I *are processed in order of their non-decreasing and non-increasing processing time, respectively. If* f *is concave, then* α_f *is given by* $\sup \{\mathrm{DEC}(I)/\mathrm{INC}(I) \mid w_j = p_j \text{ for each job } j \in I\}$.

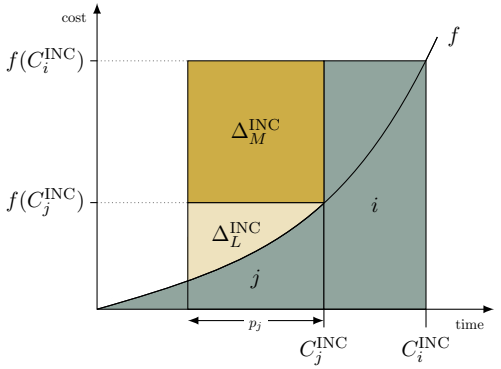
Proof. Due to Lemma 2.7 and Lemma 2.6, it suffices to show that for convex* cost functions and instances with $w_j = p_j$ for each job j, $\mathrm{DEC}(I)$ is a best* possible schedule and $\mathrm{INC}(I)$ is a worst* possible schedule.

Let S be an arbitrary schedule for an instance I with $p_j = w_j$ for each j. Consider some pair of adjacent jobs i, j, let t be the point in time when S begins to process i, and let $S^{i \leftrightarrow j}$ denote the schedule obtained by interchanging i and j. As the interchange operation only affects the cost of i and j it holds that

$$\begin{aligned} & S(I) - S^{i \leftrightarrow j}(I) \\ &\quad = \; \Big(p_i \, f(t + p_i) + p_j \, f(t + p_i + p_j) \Big) \;-\; \Big(p_j \, f(t + p_j) + p_i \, f(t + p_j + p_i) \Big) \\ &\quad = \; \tfrac{p_j}{p_i} \cdot p_i \Big(f(t + p_i + p_j) - f(t + p_j) \Big) \;-\; p_i \Big(f(t + p_i + p_j) - f(t + p_i) \Big) \,. \end{aligned}$$



(a) A job j's cost is represented by a rectangle. Line jobs are a collection of infinitesimal small jobs. Their total cost is given by the area under the graph of f.



(b) Jobs i and j in schedule INC(I) from Lemma 2.9. The marked areas represent the change in cost when merging the jobs or when making job j a line job.

Figure 2.1: Geometric interpretation of a schedule for jobs j with unit density $\frac{w_j}{p_j}$.

Define the function

$$h : x \mapsto p_i \Big(f(t + p_i + p_j) - f(t + p_j + p_i - x) \Big).$$

As f is convex*, function h is concave*, and we can rewrite the above as

$$S(I) - S^{i \leftrightarrow j}(I) \;=\; \tfrac{p_j}{p_i} \cdot h(p_i) - h\left(\tfrac{p_j}{p_i} \cdot p_i\right) \;. \tag{2.6}$$

The schedule INC(I) can be obtained from S by interchanging pairs of adjacent jobs i, j with $p_i > p_j$. We can analyze the cost of each such interchange operation using equation (2.6). Although h depends on the jobs i, j, its concavity* is always guaranteed. As $p_i > p_j$ implies $\frac{p_j}{p_i} < 1$, the right hand side of equation (2.6) is always non-positive* because h is concave* and $h(0) = 0$. Therefore, no such interchange operation will decrease* the schedule's cost. As the initial schedule S has been arbitrary, it follows that INC(I) is the worst* possible schedule. Using the same argumentation with $\frac{p_j}{p_i} > 1$ one can also show that DEC(I) is the best* possible schedule. □

At this point we introduce a geometric interpretation of our scheduling problem. In this interpretation each job j is represented by a rectangle having width w_j and height $f(C_j)$. As we can restrict our attention to unit ratio jobs, the width equals p_j. Hence, by arranging the rectangles along the x-axis in the order in which the corresponding jobs appear in some schedule S, each rectangle ends at the x-axis at its completion time in S. When drawing the graph of the cost function f into the same graphic, all upper right corners of the rectangles lie on this graph. The total cost of S is equal to the area of all rectangles. Note that the area below the graph of f, i.e., $\int_0^{\sum p_j} f(x)dx$ is a lower bound on the cost any schedule. An example is depicted in Figure 2.1(a).

We refer to jobs j with $w_j = p_j$ as *regular jobs*, and moreover, we introduce the notion of so called *line jobs*. A line job represents an infinite set of infinitesimal small regular jobs having finite total processing time. More formally, for some fixed p and $\varepsilon > 0$ consider the multiset consisting of p/ε identical jobs, each having processing time and weight ε. Then the line job of length p represents the job multiset obtained for $\varepsilon \to 0$.

Instead of being represented by a rectangle, a line job corresponds to a strip of width p whose upper boundary is given by the graph of f; see again Figure 2.1(a). In the presence of line jobs we calculate the cost of a schedule by summing up the area of all rectangles and the stripes of the line jobs.

The correctness of this interpretation follows from the continuity of the cost function f. A difference between regular jobs and line jobs is that line jobs can be preempted—because they actually consist of many small jobs. However, for problem instances I that consist of both regular jobs and line jobs, observe that the schedule $\mathrm{INC}(I)$ first processes the line jobs and then continues with the regular jobs while, symmetrically, $\mathrm{DEC}(I)$ processes the line jobs at the very end of the schedule. As a result of Lemma 2.8 the possibility of preemption does not play a role in our analysis.

Lemma 2.9 *For determining the worst case performance ratio of Smith's rule for the problem $1\,||\,w_j\,f(C_j)$ with concave or convex cost function f, we can restrict our attention to instances that consist of one regular job and one line job. More formally,*

$$\alpha_f \;=\; \sup\left\{ \frac{\mathrm{INC}(I)}{\mathrm{DEC}(I)} \;\middle|\; \begin{array}{l} I \text{ consists of one regular} \\ \text{job and one line job} \end{array} \right\}$$

in the case of convex cost functions. For concave cost functions it holds

$$\alpha_f \;=\; \sup\left\{ \frac{\mathrm{DEC}(I)}{\mathrm{INC}(I)} \;\middle|\; \begin{array}{l} I \text{ consists of one regular} \\ \text{job and one line job} \end{array} \right\}.$$

Proof. Based on the insight we have from Lemma 2.8 we start with some instance I consisting of regular jobs and show how to transform it into an instance I' having the property stated in the lemma, and satisfying $\mathrm{INC}(I')/\mathrm{DEC}(I') \geq^* \mathrm{INC}(I)/\mathrm{DEC}(I)$ in case of convex* cost functions.

The transformation of I proceeds as follows. While the instance contains more than one regular job, consider the first regular job j which is processed by schedule $\mathrm{INC}(I)$. Either transform j into a line job with processing time p_j, or replace j and its successor i in $\mathrm{INC}(I)$ by a new job k having processing time and weight $p_j + p_i$. We will show below that at least one of these two possibilities can always be realized without decreasing the optimality gap. We finally obtain a problem instance with one regular job and multiple line jobs. The line jobs can then be merged into a single line job without changing the cost of INC and DEC, which holds because this final operation does not change the total area of stripes in the geometric interpretation.

Let the jobs j and i be defined as in the preceding paragraph. Let I be the instance before the corresponding transformation step, let I_L be the instance obtained by replacing j with a line job, and let I_M be the instance obtained by merging j and i. We will show that

$$\frac{\mathrm{INC}(I_L)}{\mathrm{DEC}(I_L)} \geq^* \frac{\mathrm{INC}(I)}{\mathrm{DEC}(I)} \quad\vee\quad \frac{\mathrm{INC}(I_M)}{\mathrm{DEC}(I_M)} \geq^* \frac{\mathrm{INC}(I)}{\mathrm{DEC}(I)} \tag{2.7}$$

in case of convex* cost functions. Throughout the proof we assume the schedules INC and DEC to be fixed in the sense that $\mathrm{INC}(I_M)$ and $\mathrm{DEC}(I_M)$ are obtained from $\mathrm{INC}(I)$ and $\mathrm{DEC}(I)$, respectively, by simply inserting the merged job k at the former position of the jobs i and j. From Lemma 2.8 we know that the true $\mathrm{INC}(I_M)$ and $\mathrm{DEC}(I_M)$ will make the factor even larger. Let

$$\Delta_L^{\mathrm{INC}} := \mathrm{INC}(I) - \mathrm{INC}(I_L)\,, \qquad \Delta_M^{\mathrm{INC}} := \mathrm{INC}(I_M) - \mathrm{INC}(I)\,,$$

$$\Delta_L^{\mathrm{DEC}} := \mathrm{DEC}(I) - \mathrm{DEC}(I_L)\,, \qquad \Delta_M^{\mathrm{DEC}} := \mathrm{DEC}(I_M) - \mathrm{DEC}(I)\,.$$

Note that the definitions are chosen in such a way that for non-decreasing cost functions all four Δ-values will be non-negative—although the assumption of non-decreasingness is not necessary for the result to hold. See also Figure 2.1(b) for a geometric interpretation of the Δ-values. We are going to show that

$$\frac{\Delta_L^{\mathrm{INC}}}{\Delta_L^{\mathrm{DEC}}} \;\leq^* \frac{\Delta_M^{\mathrm{INC}}}{\Delta_M^{\mathrm{DEC}}} \,. \tag{2.8}$$

Using (2.8) and the fact that the proposition $A \vee B$ is logically equivalent to $\neg A \Rightarrow B$, one can show (2.7) for convex* cost functions via the following implication chain:

$$\begin{aligned} \frac{\mathrm{INC}(I_L)}{\mathrm{DEC}(I_L)} \;&=\; \frac{\mathrm{INC}(I) - \Delta_L^{\mathrm{INC}}}{\mathrm{DEC}(I) - \Delta_L^{\mathrm{DEC}}} \;<^* \frac{\mathrm{INC}(I)}{\mathrm{DEC}(I)} \\ \Longrightarrow \quad \frac{\Delta_L^{\mathrm{INC}}}{\Delta_L^{\mathrm{DEC}}} \;&>^* \frac{\mathrm{INC}(I)}{\mathrm{DEC}(I)} \quad \Longrightarrow \quad \frac{\Delta_M^{\mathrm{INC}}}{\Delta_M^{\mathrm{DEC}}} \;>^* \frac{\mathrm{INC}(I)}{\mathrm{DEC}(I)} \,, \end{aligned}$$

where equation (2.8) is used for the last implication. This finally implies

$$\frac{\mathrm{INC}(I_M)}{\mathrm{DEC}(I_M)} \;=\; \frac{\mathrm{INC}(I) + \Delta_M^{\mathrm{INC}}}{\mathrm{DEC}(I) + \Delta_M^{\mathrm{DEC}}} \;>^* \frac{\mathrm{INC}(I)}{\mathrm{DEC}(I)} \,.$$

For proving equation (2.8) we need explicit formulae for the four Δ-values. Let C_j^{INC} be the completion time of job j in $\mathrm{INC}(I)$, and let C_j^{DEC}, C_i^{INC} and C_i^{DEC} be defined analogously. The value Δ_L^{INC} is the difference of the contribution of regular job j to the cost of $\mathrm{INC}(I)$ and the contribution of line job j to the cost of $\mathrm{INC}(I_L)$, so

$$\Delta_L^{\mathrm{INC}} \;=\; p_j \cdot f\left(C_j^{\mathrm{INC}}\right) - \int_{C_j^{\mathrm{INC}}-p_j}^{C_j^{\mathrm{INC}}} f(t)\,dt \;=\; \int_0^{p_j} \Big(f\left(C_j^{\mathrm{INC}}\right) - f\left(C_j^{\mathrm{INC}} - p_j + x\right)\Big) dx \,.$$

The formula for Δ_L^{DEC} follows analogously with respect to the completion times in DEC.

The value Δ_M^{INC} is the difference between the cost contribution of the merged job k to $\mathrm{INC}(I_M)$ and the contributions of the separate jobs i and j to the cost of $\mathrm{INC}(I)$. The completion time of k in $\mathrm{INC}(I_M)$ equals the former completion time of i in $\mathrm{INC}(I)$.

$$\begin{aligned} \Delta_M^{\mathrm{INC}} \;&=\; (p_j + p_i)\, f\left(C_i^{\mathrm{INC}}\right) \;-\; p_j\, f\left(C_j^{\mathrm{INC}}\right) \;-\; p_i\, f\left(C_i^{\mathrm{INC}}\right) \\ &=\; p_j \left(f\left(C_j^{\mathrm{INC}} + p_i\right) - f\left(C_j^{\mathrm{INC}}\right)\right) . \end{aligned}$$

For computing the value Δ_L^{DEC}, observe that the merged job k completes at time C_j^{DEC} in $\mathrm{DEC}(I_M)$, thus,

$$\begin{aligned} \Delta_M^{\mathrm{DEC}} \;&=\; (p_i + p_j)\, f\left(C_j^{\mathrm{DEC}}\right) \;-\; p_i\, f\left(C_i^{\mathrm{DEC}}\right) \;-\; p_j\, f\left(C_j^{\mathrm{DEC}}\right) \\ &=\; p_i \left(f\left(C_j^{\mathrm{DEC}}\right) \;-\; f\left(C_j^{\mathrm{DEC}} - p_i\right)\right) . \end{aligned}$$

For convex* cost functions, the values Δ_L^{DEC} and Δ_M^{DEC} are related in the following way

$$\begin{aligned} \Delta_L^{\mathrm{DEC}} \;&=\; \int_0^{p_j} \Big(f\left(C_j^{\mathrm{DEC}}\right) \;-\; f\left(C_j^{\mathrm{DEC}} - p_j + x\right)\Big) dx \\ &\geq^* \; \frac{p_j}{2} \left(f\left(C_j^{\mathrm{DEC}}\right) \;-\; f\left(C_j^{\mathrm{DEC}} - p_j\right)\right) \;=\; \frac{p_j\, \Delta_M^{\mathrm{DEC}}}{2p_i} \,. \end{aligned} \tag{2.9}$$

The inequality holds because f being convex* implies that the expression in the integral is concave*.

For obtaining a similar relation between Δ_L^{INC} and Δ_M^{INC}, observe that for convex* cost functions f it holds that

$$\frac{f\left(C_j^{\mathrm{INC}}\right) - f\left(C_j^{\mathrm{INC}} - p_j + x\right)}{p_j - x} \leq^* \frac{f\left(C_j^{\mathrm{INC}} + p_i\right) - f\left(C_j^{\mathrm{INC}}\right)}{p_i} \quad \forall x \in [0, p_j)\,. \tag{2.10}$$

As the right hand side of (2.10) is independent of x, it follows that

$$\begin{aligned} \frac{\Delta_L^{\mathrm{INC}}}{\frac{1}{2}\,p_j^2} &= \frac{\int_0^{p_j}\Big(f\big(C_j^{\mathrm{INC}}\big) - f\big(C_j^{\mathrm{INC}} - p_j + x\big)\Big)dx}{\int_0^{p_j}(p_j - x)\;dx} \\ &\leq^* \quad \frac{f\big(C_j^{\mathrm{INC}} + p_i\big) - f\big(C_j^{\mathrm{INC}}\big)}{p_i} \quad = \quad \frac{\Delta_M^{\mathrm{INC}}}{p_i\,p_j}\,. \end{aligned} \tag{2.11}$$

Equation (2.8) now follows directly from (2.9) and (2.11). □

Finally, Theorem 2.4 is a direct consequence of Lemma 2.9. On the right hand side of equation (2.3), the parameters p and q respectively correspond to the length of the regular job and the line job in the problem instance I. The expressions in the numerator and denominator of $r_f(p, q)$ in equation (2.4) are exactly the cost of $\mathrm{INC}(I)$ and $\mathrm{DEC}(I)$, respectively. The correctness of the concave case can be verified analogously.

2.2.2 Simplified analysis for polynomials

In this section we show that for an important class of cost functions, Theorem 2.4 can be further simplified. We have already exploited the fact that problem $1\,||\,\sum w_j\,f(C_j)$ is invariant to weight scaling. Similarly, we say that a cost function f is *invariant to time scaling* if there is a function $\phi : \mathbb{R} \to \mathbb{R}$ such that when scaling every time value $t \geq 0$ by some factor $c \geq 0$, then $f(c\,t) = \phi(c)\,f(t)$. This implies that when scaling all processing times of an instance by c, the cost of every schedule changes by factor $\phi(c)$. Note that while invariance to weight scaling holds regardless of the cost function, not every cost function is invariant to time scaling, consider e.g., $f : t \mapsto t^2 + t$. In fact, time scalability is a characterization of monomials.

Observation 2.10 *A continuous non-negative function is time scalable if and only if it is a monomial of the form $t \mapsto c\,t^k$ for some $c, k > 0$.*

Proof. By definition, a cost function f is invariant to time scaling if there is a function $\phi : \mathbb{R} \to \mathbb{R}$ observing $f(c\,t) = \phi(c)\,f(t)$ for any $c, t > 0$. It is straightforward to see that monomials have this property. In the following we show that also the inverse implication holds. We first analyze the function ϕ using multiple times that $f(t) = f(1)\phi(t)$. For any x and y it holds that

$$f(1)\,\phi(x\,y) \;=\; f(x \cdot y) \;=\; f(1) \cdot \phi(x) \cdot \phi(y)\;,$$

and hence, $\phi(x \cdot y) = \phi(x) \cdot \phi(y)$. This implies $\phi(t^k) = \phi(t)^k$ for any positive integer k. By standard continuity arguments the latter equality also holds for any positive real number k. We conclude

$$\phi(t) \;=\; \phi\left(2^{\log_2 t}\right) \;=\; \phi(2)^{\log_2 t} \;=\; t^{\log_2 \phi(2)}\,,$$

and hence,

$$f(t) \;=\; f(1)\,\phi(t) \;=\; f(1)\,t^{\log_2 \phi(2)}\,.$$

□

Assuming time scalability we can normalize the total processing time to 1, and hence, Theorem 2.4 yields the following corollary.

Corollary 2.11 *For monomial cost functions $t \mapsto t^k$, $k > 1$, the tight approximation ratio of Smith's rule can be determined as*

$$\alpha_k := \alpha_{t^k} = \max_{0 \le p \le 1} r_k(p)$$

with

$$r_k(p) \;:=\; r_{t^k}(p, 1-p) \;=\; \frac{(k+1)\,p + (1-p)^{k+1}}{1 + k\,p^{k+1}}\,. \tag{2.12}$$

For $0 < k < 1$ the ratio α_k is obtained analogously when maximizing over $1/r_k(p)$.

As the denominator of $r_k(p)$ is strictly positive on the interval $p \in [0, 1]$, the same holds for the denominator of the first derivative $r'_k(p)$ of $r_k(p)$. Therefore, for monomial cost functions, the determination of Smith's rule's approximation factor reduces to the calculation of the root of a univariate polynomial—the denominator of $r'_k(p)$. Although polynomial cost functions are not invariant to time scaling in general, an important subclass of polynomials can be analyzed as monomials as we show in the following.

Theorem 2.12 *Consider a polynomial cost function $f : t \mapsto c_1 t^{\ell_1} + \cdots + c_m t^{\ell_m}$ with positive coefficients c_i, rational exponents $\ell_i \ge 1$, and $\max\{\ell_i\} = k$. Then the tight approximation factor α_f of Smith's Rule for problem $1 \,||\, \sum w_j\, f(C_j)$ is equal to α_k.*

Proof. Let $f = c_1\, f_1 + \ldots + c_m\, f_m$ be the polynomial cost function, where $f_1, \ldots, f_m$ are monomials, and let f_1 be the monomial with the highest degree k. For any schedule S for problem instance I, let $S_i(I)$ denote the cost of S with respect to cost function f_i. Denoting by $S_i^*(I)$ an optimal schedule for I under cost function f_i, it holds

$$\frac{\mathrm{HDF}(I)}{\mathrm{OPT}(I)} \;=\; \frac{\sum_{i=1}^m c_i \cdot \mathrm{HDF}_i(I)}{\sum_{i=1}^m c_i \cdot \mathrm{OPT}_i(I)} \;\le\; \frac{\sum_{i=1}^m c_i \cdot \mathrm{HDF}_i(I)}{\sum_{i=1}^m c_i \cdot S_i^*(I)} \;\le \max_{i=1\ldots m} \frac{\mathrm{HDF}_i(I)}{S_i^*(I)} \;\le\; \alpha_k\,.$$

The last inequality is a consequence of the tight approximation factor for monomial cost function $t \mapsto t^k$ being monotone in k for $k \ge 1$ which we prove in Lemma 2.15 (i) below.

In order to show that the above inequality is tight, fix I as a problem instance where the worst case approximation factor of Smith's rule with respect to f_1 is reached. As f_1 is invariant to time scaling, the same approximation factor is reached for each instance $c \cdot I$, which is obtained from I by multiplying all processing times by constant c. As f_1 is the monomial with the largest degree, for $c \to \infty$ the optimal solution $\mathrm{OPT}(c \cdot I)$ with respect to f converges against the optimal solution $\mathrm{OPT}_1(c \cdot I)$ with respect to f_1. Analogously, $\mathrm{HDF}(c \cdot I)$ converges against $\mathrm{HDF}_1(c \cdot I)$, and hence, the overall ratio $\mathrm{HDF}(c \cdot I)/\mathrm{OPT}(c \cdot I)$ converges against α_k. □

k	α_k	$\sqrt[k]{\alpha_k}$	$\frac{\alpha_k}{k}$	p_k
1/100	1.0036	1.4355	100.362	0.631
1/10	1.0313	1.3611	10.313	0.621
1/9	1.0341	1.3528	9.307	0.620
1/8	1.0375	1.3426	8.300	0.618
1/7	1.0416	1.3299	7.291	0.617
1/6	1.0465	1.3136	6.279	0.614
1/5	1.0526	1.2920	5.263	0.611
1/4	1.0598	1.2617	4.239	0.607
1/3	1.0676	1.2167	3.203	0.600
1/2	1.0689	1.1425	2.138	0.589
1	1.0000	1.0000	1.000	–
2	1.3064	1.1430	0.653	0.554
3	1.7587	1.2071	0.586	0.557
4	2.3090	1.2327	0.577	0.569
5	2.9283	1.2397	0.586	0.583
6	3.5974	1.2378	0.600	0.599
7	4.3039	1.2318	0.615	0.615
8	5.0398	1.2241	0.630	0.630
9	5.7996	1.2157	0.644	0.644
10	6.5793	1.2073	0.658	0.658
11	7.3761	1.1992	0.671	0.671
12	8.1875	1.1915	0.682	0.682
13	9.0117	1.1843	0.693	0.693
14	9.8472	1.1775	0.703	0.703
15	10.6925	1.1711	0.713	0.713
16	11.5467	1.1652	0.722	0.722
17	12.4089	1.1597	0.730	0.730
18	13.2782	1.1545	0.738	0.738
19	14.1540	1.1497	0.745	0.745
20	15.0357	1.1451	0.752	0.752
30	24.0892	1.1119	0.803	0.803
40	33.4126	1.0917	0.835	0.835
50	42.8888	1.0781	0.858	0.858
100	91.2843	1.0462	0.913	0.913
200	189.7292	1.0266	0.949	0.949
500	487.7482	1.0125	0.975	0.975
1000	986.2931	1.0069	0.986	0.986
2000	1984.8634	1.0038	0.992	0.992
5000	4982.9980	1.0017	0.997	0.997
10000	9981.5981	1.0009	0.998	0.998

Table 2.1: Tight approximation factors α_k of Smith's rule for monomial cost functions t^k and the related L_k-norm according to Corollary 2.11; see also Figure 2.2. By p_k we denote the corresponding p-value for which the factor α_k is attained.

In Table 2.1 and Figure 2.2 we show tight approximation factors of Smith's rule for monomials of different degrees as they compute according to Corollary 2.11. Note that Theorem 2.12 implies that these factors also apply to polynomials with non-negative coefficients of the respective degree.

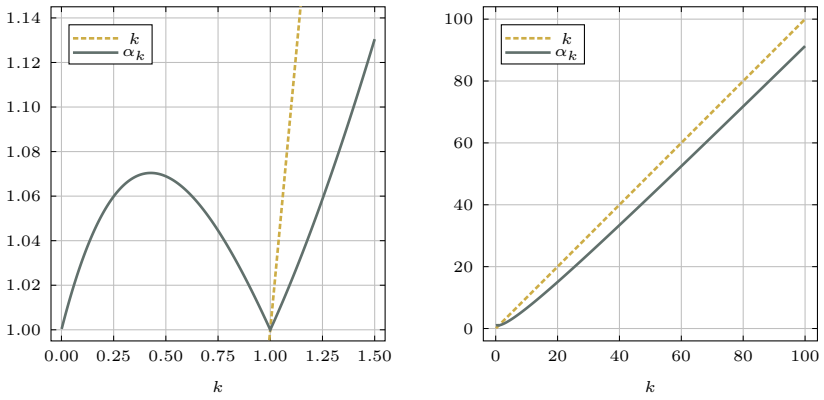
Since the objective $\sum w_j C_j^k$ differs from the L_k-norm objective $\left(\sum w_j C_j^k\right)^{1/k}$ only by the root, so does the tight approximation factor. The corresponding values are also presented in Table 2.1 and Figure 2.2. For $k > 1$, one can observe from the computed values that the approximation ratio stays below 1.25.

2.2.3 Bounding the approximation factor

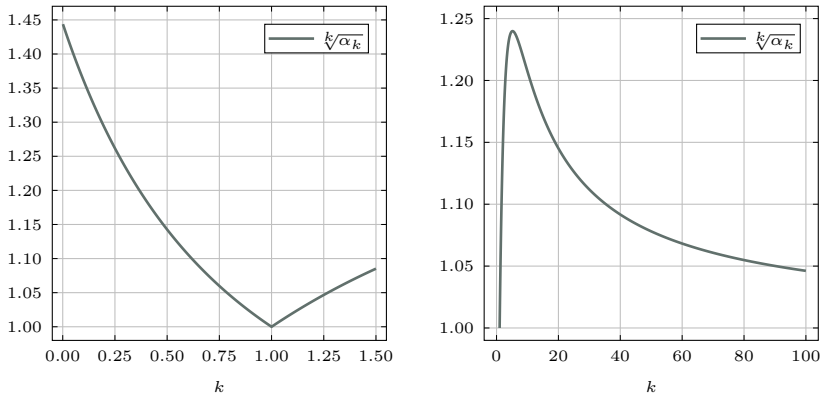
We conclude this section with additional characteristics of the tight approximation factor α_k of Smith's rule for monomials of degree k. While in Theorem 2.12 the factor was described implicitly as a result of a continuous optimization problem, here, explicit lower and upper bounds on the approximation factor are given. These bounds allow to easily analyze the asymptotic behavior of the approximation factor with respect to the degree of the monomial. The asymptotic results are summarized in the subsequent theorem, and more detailed characteristics are presented in Lemma 2.14 and 2.15. The first lemma provides properties of the extremal p-value

$$p_k := \begin{cases} \arg\min\limits_{0 \le p \le 1} r_k(p) & \text{, for } 0 < k < 1 \\ \arg\max\limits_{0 \le p \le 1} r_k(p) & \text{, for } k > 1\,, \end{cases}$$

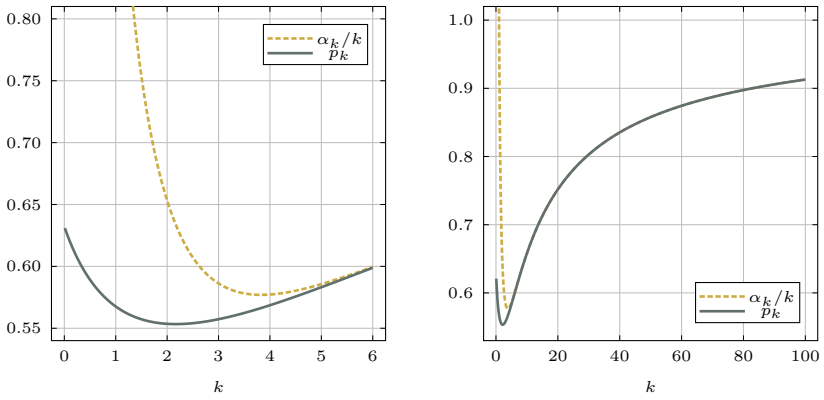
including its uniqueness, while the second lemma characterizes the approximation factor α_k.



(a) Tight approximation factor of problem $1 \,||\, \sum w_j \, C_j^k$.



(b) Tight approximation factor of problem $1 \,||\, \left(\sum w_j \, C_j^k\right)^{1/k}$.



(c) Values of p for which the tight approximation factor is attained; the respective values with respect to degree k is denoted by p_k. Surprisingly, p_k converges against the ratio of the approximation factor and the degree of the monomial. This observation was the initial motivation to further analyze the relation of p_k, α_k and k which led to the results in Section 2.2.3.

Figure 2.2: Tight analysis of the approximation factor α_k of Smith's rule for monomial cost functions t^k with $k > 0$; see also Table 2.1. The computed factors and the corresponding values of p are based on Corollary 2.11. Recall that by Theorem 2.12, the tight approximation factor of t^k equals the factor of polynomial cost functions of degree $k \geq 1$ with positive coefficients.

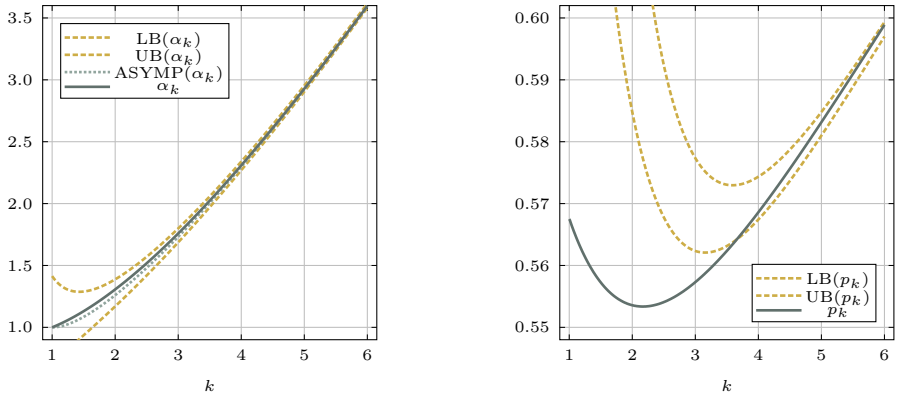


Figure 2.3: The tight approximation factor α_k and the value p_k with their lower and upper bounds according to Lemma 2.15 and 2.14, respectively, and their asymptotic bounds according to Theorem 2.13; see also Table 2.2. For the purpose of this figure, we denote $\mathrm{ASYMP}(\alpha_k) := k^{(k-1)/(k+1)}$.

Theorem 2.13 *Consider the tight approximation factor α_k of Smith's rule for problem $1\,||\sum w_j\,C_j^k$. Denoting by $p_k \in [0,1]$ a value such that $\alpha_k = r_k(p_k)$, the following holds:*

1. $\lim_{k\to\infty}\left(\alpha_k - k^{\frac{k-1}{k+1}}\right) = 0 \quad$ *and* $\quad \lim_{k\to\infty}\left(p_k - \sqrt[k+1]{\frac{1}{k^2}}\right) = 0\,,$
2. $\lim_{k\to\infty}\dfrac{\alpha_k}{k\cdot p_k} = 1\,,$
3. $\lim_{k\to\infty}(k-\alpha_k) = \infty\,.$

Proof. The theorem is a direct consequence of the below Lemmas 2.14 and 2.15. The statements (i) and (ii) follow from the lower and upper bounds on p_k and α_k as given in these two lemmas. The values $\delta_{p_k}^-, \delta_{\alpha_k}^-$ and $\delta_{\alpha_k}^+$ therein obviously tend to 1 for $k\to\infty$. Statement (iii) is implied by Lemma 2.15 (iii). □

Being obvious from this proof, a major insight of the Lemmas 2.14 and 2.15 are the bounds on α_k and p_k. Exemplary values are given in Figure 2.3 and Table 2.2.

Lemma 2.14 (Properties of p_k) *The value p_k observes the following properties:*

(i) *p_k is unique for $k \neq 1$, and it is the unique root of r_k' on $]0,1]$,*

(ii) *$p_k \geq \frac{1}{2}$ for all $k \neq 1$,*

(iii) *$\delta_{p_k}^- \cdot \sqrt[k+1]{\frac{1}{k^2}} \leq p_k \leq \sqrt[k+1]{\frac{1}{k^2}}$ with $\delta_{p_k}^- := \sqrt[k+1]{\frac{k^2}{k^2+1}}$ where the lower bound holds for $k \geq 4$.*

Proof. The analysis of p_k and the related function r_k as given in (2.12) is majorly based on a discussion of r_k's first derivative

$$r_k' : p \;\mapsto\; \frac{k+1}{\left(1+kp^{k+1}\right)^2}\cdot s_k(p) \qquad \text{with}$$
$$s_k(p) \;:=\; 1-(1-p)^k - kp^k\left(kp+(1-p)^k\right)\,. \tag{2.13}$$

k	$\mathrm{LB}(\alpha_k)$	$k^{\frac{k-1}{k+1}}$	α_k	$\mathrm{UB}(\alpha_k)$	$\mathrm{LB}(p_k)$	p_k	$\sqrt[k+1]{\frac{1}{k^2}} = \mathrm{UB}(p_k)$
4	2.26971	2.29740	2.30901	2.33949	0.56743	0.56859	0.57435
5	2.90497	2.92402	2.92834	2.95310	0.58099	0.58315	0.58480
6	3.58197	3.59602	3.59745	3.61736	0.59700	0.59888	0.59934
7	4.29266	4.30352	4.30394	4.31984	0.61324	0.61467	0.61479
8	5.03101	5.03968	5.03980	5.05257	0.62888	0.62993	0.62996
9	5.79243	5.79955	5.79958	5.80998	0.64360	0.64439	0.64439
10	6.57338	6.57933	6.57934	6.58794	0.65734	0.65793	0.65793
15	10.68955	10.69251	10.69251	10.69656	0.71264	0.71283	0.71283
20	15.03387	15.03566	15.03566	15.03799	0.75169	0.75178	0.75178
25	19.51549	19.51669	19.51669	19.51821	0.78062	0.78067	0.78067
30	24.08838	24.08924	24.08924	24.09031	0.80295	0.80297	0.80297
35	28.72610	28.72675	28.72675	28.72754	0.82075	0.82076	0.82076
40	33.41213	33.41263	33.41263	33.41324	0.83530	0.83532	0.83532
45	38.13547	38.13587	38.13587	38.13636	0.84745	0.84746	0.84746
50	42.88845	42.88878	42.88878	42.88917	0.85777	0.85778	0.85778

Table 2.2: The tight approximation factor α_k and the value p_k with their lower and upper bounds according to Lemma 2.15 and 2.14, respectively, and their asymptotic bounds according to Theorem 2.13; see also Figure 2.3.

As a first step, we show that

$$s_k\left(\tfrac{1}{2}\right) \;=\; 1 - \frac{1}{2^k} - \frac{k^2}{2^{k+1}} - \frac{k}{2^{2k}} \tag{2.14}$$

is strictly negative for $0 < k < 1$ and strictly positive for $k > 1$, respectively. This observation is then used to show properties (i) and (ii).

In order to prove the observation, we first address the case $k \in]k', \infty[$ where

$$k' \;:=\; \ln\left(\tfrac{1}{\ln(2)}\right)/\ln(2) \;\; (\approx\; 0.53)\,.$$

On this interval, we show that $s_k(\frac{1}{2})$ is increasing in k. Consider

$$\tfrac{d}{dk}\; s_k\left(\tfrac{1}{2}\right) \;=\; \frac{1}{2^{2k}}\left(k\Big(2^{k-1}\big(\ln(2)\,k - 2\big) + 2\,\ln(2)\Big) + \ln(2)\,2^k - 1\right).$$

Due to $k > k'$, the last term $\ln(2)\,2^k - 1$ is positive. Moreover, by considering its derivative, we can verify that the term $2^{k-1}\big(\ln(2)\,k - 2\big) + 2\,\ln(2)$ attains its minimum in $k = 1/\ln(2)$, and this minimum value is also positive. Hence, $\frac{d}{dk}\; s_k\left(\frac{1}{2}\right)$ is positive, and so, $s_k\left(\frac{1}{2}\right)$ is increasing in k. Observing that $s_1(\frac{1}{2})$ is zero, it follows that $s_k(\frac{1}{2})$ is strictly negative for $k' < k < 1$ and strictly positive for $k > 1$, respectively.

Finally, for $0 < k \le k'$, we show that $s_k(\frac{1}{2})$ is negative by arguing that $s_0(\frac{1}{2}) = 0$, $s_{k'}(\frac{1}{2}) < 0$ and that $s_k(\frac{1}{2})$ is convex in k on the respective interval. Concerning the latter, it is sufficient to show that $\frac{d^2}{dk^2}\; s_k\left(\frac{1}{2}\right)$ is positive. This derivative is given by

$$\begin{aligned}
\tfrac{d^2}{dk^2}\; s_k\left(\tfrac{1}{2}\right) \;&=\; 2^{-2k}\cdot\Big(a(k) - b(k)\Big) \qquad \text{where}\\
a(k) \;&:=\; 4\,\ln(2)\,(1 + k\,2^{k-1})\,,\\
b(k) \;&:=\; 4\,\ln^2(2)\,k + (\ln^2(2) + 1)\cdot 2^k + \ln^2(2)\,k^2\cdot 2^{k-1}\;.
\end{aligned}$$

Obviously, a and b are increasing in k. This allows the following estimates

$$\begin{aligned} a(k) \geq\ \ a(0)\ \ &> b(0.3) \geq b(k) && \text{for} \quad 0 < k \leq 0.3\,, \\ a(k) > a(0.3) &> \ b(k')\ \ \geq b(k) && \text{for} \quad 0.3 < k \leq k'\,, \end{aligned}$$

showing that $\frac{d^2}{dk^2}\, s_k\left(\frac{1}{2}\right)$ is positive for the considered values of k. Hence, $s_k(\frac{1}{2})$ is convex. This concludes the proof of the observation that $s_k(\frac{1}{2})$ is strictly negative for $0 < k < 1$ and strictly positive for $k > 1$.

For the proof of (i), we show that the function r_k has a unique minimum and maximum for $0 < k < 1$ and $k > 1$, respectively. We argue that its first derivative r_k' has a unique root on $]0, 1[$. One can then easily observe that this corresponds to a maximum and minimum, respectively. Searching for roots of r_k', by (2.13), it is sufficient to investigate s_k. Therein, the term $(1-p)^k$ is concave for $0 < k < 1$ and convex for $k > 1$, and hence, $1 - (1-p)^k$ is convex and concave, respectively. We will show below that also the term

$$h_k(p) \ := \ -kp^k\Big(kp + (1-p)^k\Big)$$

is convex for $0 < k < 1$ and concave for $k > 1$, and thus, so is s_k. Observing additionally that $s_k(0) = 0$, and $s_k(1) > 0 > s_k(\frac{1}{2})$ for $0 < k < 1$ and $s_k(1) < 0 < s_k(\frac{1}{2})$ for $k > 1$, respectively, it follows immediately that s_k has a unique root on $]0, 1[$. The bounds on $s_k(\frac{1}{2})$ were provided by the above observation.

It remains to analyze $h_k(p)$, whose second derivative with respect to p is given by

$$h_k''(p) \ = \ -k^2p^{k-2}\bigg(\ p\Big(k(k+1) - 2\,(1-p)^{k-1}\Big)\ +\ (k-1)(1-p)^{k-2}(1-2p)^2\bigg)$$

for $0 < p < 1$. The sign of the last term $(k-1)(1-p)^{k-2}(1-2p)^2$ is solely determined by the factor $k-1$. Considering $k(k+1) - 2\,(1-p)^{k-1}$, we can similarly observe that the term is strictly negative for $0 < k < 1$ and strictly positive for $k > 1$. This follows directly by trivial bounds of $k(k+1)$ and $(1-p)^{k-1}$ on the respective intervals of k. In summary, this shows that $h_k''(p)$ is strictly positive for $0 < k < 1$ and strictly negative for $k > 1$, and so, $h_k(p)$ is convex and concave, respectively. This concludes the proof of (i).

For showing points (ii) and (iii), we make use of (i). By (i), we know that r_k attains its unique extremum on $[0, 1]$ for some value $p_k \in]0, 1[$, and moreover, p_k is the unique root of the derivative r_k' on this interval. Now, for any value p, the sign of $r_k'(p)$ tells us whether p lies before or after the extremal value p_k, and accordingly, p yields a lower or upper bound on p_k, respectively.

Utilizing this insight, for the proof of (ii) it suffices to show that $r_k'(\frac{1}{2})$ is negative for $0 < k < 1$ and positive for $k > 1$. This is exactly what was shown in the observation at the very beginning of the proof.

In the remainder of the proof, we show (iii), bounding p_k for $k > 4$. As for (ii), we use the observation that the derivative's sign implies bounds on p_k. In case of $k > 1$, a value p with negative $r_k'(p)$ corresponds to an upper bound on p_k. According to equation (2.13), the condition $k^2p^{k+1} > 1$ is sufficient for this to hold. Transposing this inequality yields the stated upper bound $\sqrt[k+1]{1/k^2}$ on p_k. Note, that the upper bound naturally holds for $0 < k < 1$ as well, since it is larger than 1 in this case.

For showing that $p = \sqrt[k+1]{1/(k^2+1)}$ is a lower bound on p_k for $k \geq 4$, it remains to prove that $r_k'(p) > 0$. According to (2.13), this is equivalent to $s_k(p) > 0$. For our specific value of p, we obtain

$$s_k\left(\sqrt[k+1]{\tfrac{1}{k^2+1}}\right) \;=\; \left(1 - \left(\sqrt[k+1]{k^2+1} - 1\right)^k \cdot \left(\sqrt[k+1]{k^2+1} + \frac{k}{\sqrt[k+1]{(k^2+1)^{k-1}}}\right)\right) \cdot \frac{1}{k^2+1}\,.$$

Denoting

$$c(k) := \left(\sqrt[k+1]{k^2+1} - 1\right)^k \qquad \text{and} \qquad d(k) := \sqrt[k+1]{k^2+1} \;+ \frac{k}{\sqrt[k+1]{(k^2+1)^{k-1}}}\,,$$

it suffices to show that $c(k) \cdot d(k)$ does not exceed 1. We show $c(k) < \frac{2}{5}$ and $d(k) \leq \frac{5}{2}$. Resolving $c(k) < \frac{2}{5}$ for k^2+1, it is straightforward to see that this inequality holds for $k \geq 4$.

Regarding $d(k)$, it holds that $d(4) \leq \frac{5}{2}$, so that it suffices to show that $d(k)$ is decreasing in $k \geq 4$. We compute

$$\begin{aligned}
\tfrac{d}{dk}\, d(k) \;&= (k^2+1)^{-\frac{k-1}{k+1}} \left(1 - 2\cdot\tfrac{k-1}{k+1}\cdot\tfrac{k^2}{k^2+1} + 2\,\tfrac{k}{k+1}\cdot\tfrac{1}{\sqrt[k+1]{k^2+1}} - \tfrac{(k^2+1)^{\frac{k}{k+1}} + 2\,k}{k^2+1\;+\;2\,k}\cdot\ln(k^2+1)\right) \\
&\leq (k^2+1)^{-\frac{k-1}{k+1}} \left(1 - 2\cdot\tfrac{3}{5}\cdot\tfrac{16}{17} + 2\,\tfrac{1}{\sqrt[k+1]{k^2+1}} - \tfrac{(k^2+1)^{\frac{k}{k+1}}}{k^2+1}\cdot\ln(k^2+1)\right) \\
&= (k^2+1)^{-\frac{k-1}{k+1}} \left(-\tfrac{11}{85} + \left(2 - \ln(k^2+1)\right)\tfrac{1}{\sqrt[k+1]{k^2+1}}\right) \;<\; 0\,,
\end{aligned}$$

where the estimates are based on $k \geq 4$. This derivative being negative implies that $d(k)$ is decreasing which completes the proof. □

Note that the lower bound on p_k actually holds also for values of k down to roughly $k = 3.658$. However, in order to restrict the length of the analysis to a reasonable degree, we did not make great efforts to compute the exact analytic value of k for the bound to hold, and opted for the next larger integral value of 4. We continue with the characteristics of the approximation factor α_k.

Lemma 2.15 (Properties of α_k) *The tight approximation factor α_k observes the following:*

(i) α_k *is increasing in* k *for* $k > 1$.

(ii) $\delta^-_{\alpha_k} \cdot k^{\frac{k-1}{k+1}} \;\leq\; \alpha_k \;\leq\; \delta^+_{\alpha_k} \cdot k^{\frac{k-1}{k+1}}$ *for* $k \geq 4$
with $\delta^-_{\alpha_k} := \sqrt[k+1]{\frac{k^2}{k^2+1}}$ *and* $\delta^+_{\alpha_k} := \frac{\sqrt[k+1]{(k^2+1)^k}\cdot k + k^2 + 1}{\sqrt[k+1]{(k^2+1)^k}\cdot k + k^2}$.

(iii) $\lim\limits_{k\to\infty} \alpha_k \;=\; \infty$

(iv) $k - \alpha_k \;\geq\; \ln k - \frac{1}{2k}$ *for* $k \geq 4$.

Proof. Considering point (i), we know that $\alpha_k = r_k(p_k)$ for $k > 1$. We argue that α_k is increasing in k by showing that the first derivative of $r_k(p)$ with respect to k is positive for any $p \in \left[\frac{1}{2},\, \sqrt[k+1]{1/k^2}\right]$. The latter is sufficient to assume due to Lemma 2.14. We denote by

$$v_k(p) \;:=\; (k+1)\,p + (1-p)^{k+1} \qquad \text{and} \qquad w_k(p) \;:=\; 1 + k\,p^{k+1}$$

the nominator and denominator of r_k, respectively. Then,

$$\tfrac{d}{dk} r_k(p) \;=\; \frac{\tfrac{d}{dk}\big(v_k(p)\big)\cdot w_k(p) - \tfrac{d}{dk}\big(w_k(p)\big)\cdot v_k(p)}{w_k^2(p)}$$

is positive if and only if

$$\tfrac{d}{dk}\, v_k(p) \;>\; \frac{v_k(p)}{w_k(p)} \cdot \tfrac{d}{dk}\, w_k(p) \;=\; r_k(p)\cdot \tfrac{d}{dk}\, w_k(p)\,. \tag{2.15}$$

We first address

$$\tfrac{d}{dk}\, v_k(p) \;=\; p \;-\; \ln\Big(\tfrac{1}{1-p}\Big)\cdot (1-p)^{k+1}\,,$$

and show that it is positive for the respective values of p. The term $\ln(1/(1-p))\cdot(1-p)^{k+1}$ is decreasing in p which can be easily verified by its derivative with respect to p. Hence,

$$\tfrac{d}{dk}\, v_k(p) \;\geq\; \tfrac{d}{dk}\, v_k(\tfrac{1}{2}) \;>\; 0 \qquad \text{for all}\;\; p \geq \tfrac{1}{2}\,.$$

In contrast,

$$\tfrac{d}{dk}\, w_k(p) \;=\; p^{k+1}\,\Big(1 - k\cdot \ln\big(\tfrac{1}{p}\big)\Big)$$

can be either positive, negative or zero. Since $r_k(p)$ and $\frac{d}{dk}\, v_k(p)$ are strictly positive, the inequality (2.15) is naturally satisfied whenever $\frac{d}{dk}\, w_k(p)$ is not strictly positive. Thus, it remains to show (2.15) in case of positive $\frac{d}{dk}\, w_k(p)$ which is equivalent to

$$p \;>\; e^{-\frac{1}{k}}\,. \tag{2.16}$$

If for some $k > 1$, the upper bound $\sqrt[k+1]{1/k^2}$ on p_k is smaller than $e^{-\frac{1}{k}}$, then this implies that $\frac{d}{dk}\, w_k(p)$ is negative for all relevant values of p. Hence, we only need to consider such values of k for which $e^{-1/k}$ is at most as large as the upper bound on p_k, i.e., values of k for which

$$k^{\frac{2k}{k+1}} \;\leq\; e\,.$$

Since the term $k^{\frac{2k}{k+1}}$ is increasing in k, and it is already larger than e for $k = 2.1$, it remains to consider $1 < k < 2.1$ and $p \geq \max\{e^{-1/k}, \frac{1}{2}\}$.

Recall that $r_k(p_k) \geq r_k(p)$ for any p, and moreover, Corollary 2.2 implies that $r_k(p_k)$ never exceeds k. Utilizing this bound and knowing that $\frac{d}{dk}\, w_k(p) > 0$ due to $p > e^{-1/k}$, it suffices to show the following inequality instead of (2.15)

$$\frac{\tfrac{d}{dk}\, v_k(p)}{\tfrac{d}{dk}\, w_k(p)} \;>\; k\,.$$

This can be rewritten as

$$p\,(1 - k\,p^k) \;>\; \ln\Big(\tfrac{1}{1-p}\Big)\,(1-p)^{k+1} \;-\; k^2\,\ln\Big(\tfrac{1}{p}\Big)\,p^{k+1}\,. \tag{2.17}$$

Due to the upper bound on p_k from Lemma 2.14, the left hand side of (2.17) is positive. Regarding the right hand side, we show that it is decreasing in k, and it is negative for the minimum k-value 1, which immediately implies the above inequality. Consider the derivative of the right hand side with respect to k, which is given by

$$-\ln\left(\tfrac{1}{1-p}\right)^2 (1-p)^{k+1} \;-\; k\,\ln\left(\tfrac{1}{p}\right)\, p^{k+1}\,\left(2-k\,\ln\left(\tfrac{1}{p}\right)\right)\,.$$

Since we assumed $\frac{d}{dk}\, w_k(p)$ to be positive, also $2 - k\,\ln(1/p)$ is positive, and hence, the derivative is negative for $k > 1$ which implies that the right hand side of (2.17) is decreasing. Finally, for $k = 1$ the right hand side is non-positive if and only if

$$\tfrac{1}{p^2}\cdot\ln\left(\tfrac{1}{1-p}\right) \;\le\; \tfrac{1}{(1-p)^2}\cdot\ln\left(\tfrac{1}{p}\right) \quad\Leftrightarrow\quad \ln\left((1-p)^{-\frac{1}{(1-(1-p))^2}}\right) \;\le\; \ln\left(p^{-\frac{1}{(1-p)^2}}\right).$$

Since $x \mapsto x^{-1/(1-x)^2}$ can be shown to be increasing on $\left[\frac{1}{2}, 1\right]$, and due to $1 - p \le p$, the inequality is satisfied. This completes the proof of (i).

According to (ii), we continue with proving the stated lower and upper bound on α_k for $k \ge 4$. By Lemma 2.14 (i), we know that r_k attains its unique maximum on $[0, 1]$ for some value $p_k \in]0, 1[$, and hence, r_k' is zero at this point. By equation (2.13), this is equivalent to

$$(1-p)^k = \frac{1-k^2p^{k+1}}{1+kp^k}\;.$$

Replacing this term in the definition of r_k in (2.12) accordingly, we obtain the function

$$\begin{aligned} u_k(p) \;&:=\; \frac{(k+1)p \;+\; (1-p)\frac{1-k^2p^{k+1}}{1+kp^k}}{1+kp^{k+1}} \\ &=\; \frac{1 \;+\; kp \;+\; kp^{k+1} \;+\; k^2p^{k+2}}{(1+kp^{k+1})(1+kp^k)} \;=\; \frac{1+kp}{1+kp^k}\;. \end{aligned} \tag{2.18}$$

By construction, the functions u_k and r_k are equal for p_k, and hence, the approximation factor can now be described by u_k as

$$\alpha_k \;=\; \frac{1+kp_k}{1+kp_k^k} \;=\; \frac{1+\frac{1}{kp_k}}{1+kp_k^k}\cdot kp_k\;.$$

Utilizing the lower and upper bound on p_k from Lemma 2.14 (iii), we obtain

$$\alpha_k \;\le\; \frac{1+\frac{1}{k\sqrt[k+1]{\frac{1}{k^2+1}}}}{1+k\sqrt[k+1]{\frac{1}{k^2+1}}^{\,k}}\cdot k\sqrt[k+1]{\tfrac{1}{k^2}} \;=\; \frac{k\,(k^2+1)^{\frac{k}{k+1}}+k^2+1}{k\,(k^2+1)^{\frac{k}{k+1}}+k^2}\cdot k^{\frac{k-1}{k+1}}$$

and

$$\alpha_k \;\ge\; \frac{1+\frac{1}{k\sqrt[k+1]{\frac{1}{k^2}}}}{1+k\sqrt[k+1]{\frac{1}{k^2}}^{\,k}}\cdot k\sqrt[k+1]{\tfrac{1}{k^2+1}} \;=\; \sqrt[k+1]{\tfrac{k^2}{k^2+1}}\cdot k^{\frac{k-1}{k+1}}\;.$$

Since the lower bound on p_k holds only for $k \geq 4$, so do these bounds on α_k.

Due to $\lim_{k\to\infty} \delta^-_{\alpha_k} = 1$ and $\lim_{k\to\infty} k^{\frac{k-1}{k+1}} = \infty$, the lower bound on α_k implies directly point (iii).

We conclude the proof showing (iv). With the upper bound on α_k from (ii), it follows that

$$\begin{aligned} k - \alpha_k &\geq k - \frac{\sqrt[k+1]{(k^2+1)^k} \cdot k + k^2 + 1}{\sqrt[k+1]{(k^2+1)^k} \cdot k + k^2} \cdot k^{\frac{k-1}{k+1}} \\ &= k - k^{\frac{k-1}{k+1}} - \frac{1}{\sqrt[k+1]{(k^2+1)^k} \cdot k + k^2} \cdot k^{\frac{k-1}{k+1}} \end{aligned}$$

By a series of equivalent transformations, we observe that

$$\begin{aligned} k - k^{\frac{k-1}{k+1}} \geq \ln k \quad &\iff \quad \ln(k - \ln k) \geq \ln\left(k^{\frac{k-1}{k+1}}\right) = \tfrac{k-1}{k+1} \ln k \\ &\iff \quad \frac{2}{k+1} \geq \frac{\ln k - \ln(k - \ln k)}{\ln k}\,. \end{aligned} \tag{2.19}$$

Since the logarithm is concave and increasing for positive input, its slope at $k - \ln k$ is larger than the slope connecting $\ln(k - \ln k)$ and $\ln k$. Thus,

$$\frac{\ln k - \ln(k - \ln k)}{\ln k} \leq \frac{1}{k - \ln k}\,.$$

It can be easily checked that the last term is not larger than $\frac{2}{k+1}$ for $k \geq 4$ which yields equation (2.19). Point (iv) finally follows by observing that

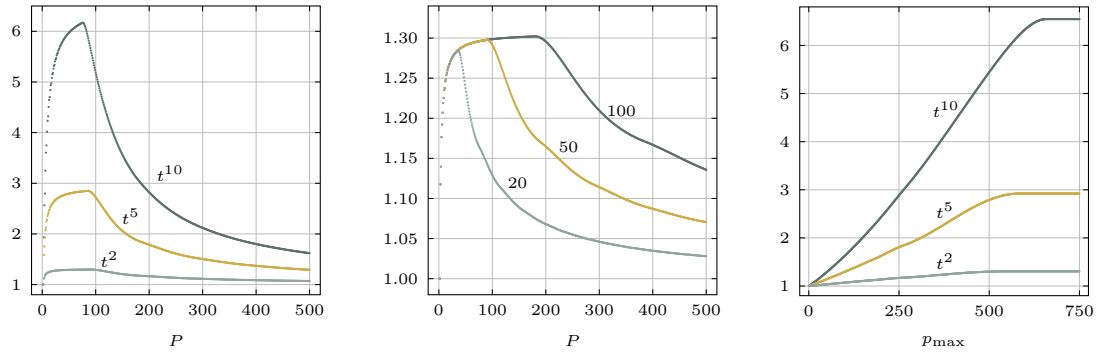
$$\frac{1}{\sqrt[k+1]{(k^2+1)^k} \cdot k + k^2} \cdot k^{\frac{k-1}{k+1}} \leq \frac{1}{\left(\sqrt[k+1]{k^{2k}} + k\right) \cdot k} \cdot k \leq \frac{1}{2k}\,.$$

□

2.3 Parameterized analysis

In this final section, we refine the analysis of Smith's rule in order to make it more suitable to realistic problem instances. To this end, we introduce the parameter $p_{\max} > 0$, the maximum processing time, and P, the total processing time of all jobs, assuming that the processing times are integral. These parameters allow us to ban infinitesimally small and very large jobs as they appear in the unparameterized analysis. Note that our analysis of the approximation factor also covers the case where, instead of the restriction to integral processing times, some lower bound D on the greatest common divisor of all processing times is fixed. The corresponding approximation factor is then achieved by scaling $p_{\max}$ and P down by D and by replacing $f(t)$ by $f(t/D)$. Throughout the analysis, the two parameters $p_{\max}$ and P will be assumed to be fixed integers.

Due to this discretization, the tie breaking policy of Smith's rule is becoming a relevant issue. The proof of Lemma 2.6 in the preceding section exploits the fact that problem instances with ties can be approximated arbitrarily close by instances without ties, but such continuity arguments are not possible in the presence of the integrality



(a) Cost functions $f(t) = t^2, t^5, t^{10}$ and fixed $p_{\max} = 50$.

(b) Cost function $f(t) = t^2$ and values $p_{\max} = 20, 50, 100$.

(c) Cost functions $f(t) = t^2, t^5, t^{10}$ and fixed $P = 1000$.

Figure 2.4: Tight approximation factor of Smith's rule for monomial cost functions in dependency of the maximum and total processing time $p_{\max}$ and P, respectively; see also Table 2.3.

restriction on processing times. In what follows we continue to analyze the variant of Smith's rule having the worst possible tie breaking rule, and remark that the approximation factors for other tie breaking rules may well be smaller.

The analysis is similar to the unparameterized case above. Also here we can show that in worst case instances all jobs have density 1, and the largest ratio is obtained when comparing the schedules that sort the jobs in increasing and decreasing order of the job's weight, respectively.

Observation 2.16 *Lemma 2.7 and Lemma 2.8 hold when restricting to instances with integral processing times, maximum processing time p_{max} and total processing time P. The results carry over without any modification of the proofs.*

Lemma 2.9 in the unparameterized analysis has stated that worst case instances consist of one regular job and one line job. The refined analysis will be similar. Instead of a regular job of length p the instances will contain $\lfloor p/p_{\max} \rfloor$ jobs each having a length of $p_{\max}$, plus one length $p \bmod p_{\max}$ job, where p is an integer between 0 and P. Instead of a line job we will have $P - p$ jobs each having length 1. As we will prove in the next theorem, one can determine the tight approximation factor of Smith's rule for given parameters $p_{\max}$ and P by finding the value of p which maximizes the ratio between $\mathrm{INC}(p, p_{\max}, P)$ and $\mathrm{DEC}(p, p_{\max}, P)$, the schedules which process the jobs of the instance determined by $p, p_{\max}, P$ in increasing and decreasing order of their processing times, respectively. In Figure 2.4 and Table 2.3 and we show approximation factors computed for different cost functions and parameters.

Theorem 2.17 *Given the maximum processing time p_{max} and the total processing time P, the tight approximation ratio of Smith's rule for problem $1 \,||\, \sum w_j f(C_j)$ under the assumption of integral processing times is given by*

$$\sup \left\{ \frac{\mathrm{HDF}(I)}{\mathrm{OPT}(I)} \;\middle|\; \max_{j \in J} p_j = p_{max}, \sum_{j \in J} p_j = P \right\} = \max \left\{ \frac{\mathrm{INC}(p, p_{max}, P)}{\mathrm{DEC}(p, p_{max}, P)} \;\middle|\; p = 0, \ldots, P \right\}$$

in case of convex cost functions f, and for concave f it holds that

$$\sup \left\{ \frac{\mathrm{HDF}(I)}{\mathrm{OPT}(I)} \;\middle|\; \max_{j \in J} p_j = p_{max}, \sum_{j \in J} p_j = P \right\} = \max \left\{ \frac{\mathrm{DEC}(p, p_{max}, P)}{\mathrm{INC}(p, p_{max}, P)} \;\middle|\; p = 0, \ldots, P \right\}.$$

	k		
P	2	5	10
5	1.235	2.355	4.233
10	1.268	2.618	5.179
15	1.281	2.715	5.617
20	1.287	2.765	5.833
25	1.291	2.795	5.974
30	1.293	2.818	6.070
35	1.295	2.834	6.137
40	1.297	2.845	6.192
45	1.298	2.854	6.232
50	1.299	2.861	6.267
55	1.299	2.867	6.294
60	1.300	2.872	6.317
65	1.300	2.877	6.337
70	1.301	2.880	6.353
75	1.301	2.883	6.368
80	1.302	2.886	6.300
85	1.302	2.889	6.095
90	1.302	2.873	5.841
95	1.300	2.826	5.579
100	1.296	2.760	5.327
200	1.167	1.800	2.860
300	1.116	1.512	2.139
400	1.088	1.375	1.813
500	1.071	1.296	1.630
$p_{\max} = 50$			

	$p_{\max}$		
P	20	50	100
10	1.268	1.268	1.268
20	1.287	1.287	1.287
30	1.293	1.293	1.293
40	1.289	1.297	1.297
50	1.246	1.299	1.299
60	1.205	1.300	1.300
70	1.179	1.301	1.301
80	1.164	1.302	1.302
90	1.148	1.302	1.302
100	1.133	1.296	1.303
110	1.122	1.281	1.303
120	1.113	1.262	1.303
130	1.106	1.242	1.303
140	1.098	1.225	1.304
150	1.092	1.210	1.304
160	1.087	1.197	1.304
170	1.082	1.187	1.304
180	1.077	1.179	1.304
190	1.073	1.172	1.303
200	1.070	1.167	1.298
250	1.057	1.136	1.254
300	1.047	1.116	1.211
400	1.036	1.088	1.168
500	1.029	1.071	1.137
$k = 2$			

	k		
$p_{\max}$	2	5	10
25	1.018	1.070	1.140
50	1.036	1.144	1.294
75	1.054	1.219	1.458
100	1.072	1.298	1.634
125	1.089	1.378	1.820
150	1.106	1.461	2.017
175	1.121	1.542	2.221
200	1.137	1.628	2.435
225	1.154	1.723	2.661
250	1.169	1.810	2.891
275	1.179	1.881	3.118
300	1.192	1.963	3.351
325	1.207	2.060	3.599
350	1.223	2.167	3.857
375	1.239	2.280	4.122
400	1.255	2.395	4.391
425	1.270	2.509	4.661
450	1.282	2.615	4.930
500	1.300	2.793	5.459
550	1.306	2.902	5.951
600	1.306	2.925	6.352
650	1.306	2.925	6.558
700	1.306	2.925	6.563
750	1.306	2.925	6.563
$P = 1000$			

Table 2.3: Numerical values corresponding to the three plots in Figure 2.4. All tables show tight approximation factors of Smith's rule for different cost functions t^k and parameters $p_{\max}$ and P. The first subtable shows for three cost functions how the approximation factor changes for increasing P while $p_{\max}$ is fixed. In contrast, in the second subtable, the cost function is fixed and three values of $p_{\max}$ are considered. The last subtable is analogously to the first one, but with $p_{\max}$ and P exchanged.

Proof. Since the parameters $p_{\max}$ and P are assumed to be fixed throughout the proof, we use the simplified notations $\mathrm{INC}(p)$ and $\mathrm{DEC}(p)$ instead of $\mathrm{INC}(p, p_{\max}, P)$ and $\mathrm{DEC}(p, p_{\max}, P)$, respectively. The cost of the decreasing schedule is given by

$$\mathrm{DEC}(p) \;=\; \sum_{i=1}^{\left\lfloor \frac{p}{p_{\max}} \right\rfloor} p_{\max} \cdot f(i \cdot p_{\max}) \;+\; (p \bmod p_{\max}) \cdot f(p) \;+\; \sum_{j=1}^{P-p} f(p+j)\;,$$

while for the increasing schedule it holds that

$$\begin{aligned}\mathrm{INC}(p) \;=\; &\sum_{j=1}^{P-p} f(j) \;+\; (p \bmod p_{\max}) \cdot f(P - p + p \bmod p_{\max}) \\ &+ \sum_{i=1}^{\left\lfloor \frac{p}{p_{\max}} \right\rfloor} p_{\max} \cdot f(P - p + p \bmod p_{\max} + i \cdot p_{\max})\,.\end{aligned}$$

As in the preceding section we give an explicit proof for the case of convex cost functions and mark all words and symbols with an asterisk that have to be replaced with their

opposite in order to obtain a proof for concave cost functions. The central argument is provided by the following claim.

Claim: Consider a problem instance I and a corresponding schedule S. Let i, j, k be three jobs that are scheduled consecutively in S and which have the characteristics $p_i = w_i$, $p_j = w_j$, and $p_k = w_k \geq p_i$. Let I_{ij} be the modified problem instance in which jobs i and j have been merged into a single job with processing time and weight $p_i + p_j$, and let S_{ij} be the schedule for I_{ij} obtained from S by replacing i and j with the merged job. Let I_{jk} and S_{jk} be defined analogously. Moreover, let $\bar{S}$ be the reverse schedule of S. Then

$$\frac{S_{jk}}{\bar{S}_{jk}} \geq^* \frac{S_{ij}}{\bar{S}_{ij}} \quad \vee \quad \frac{S}{\bar{S}} \geq^* \frac{S_{ij}}{\bar{S}_{ij}} \ .$$

For the proof of the claim, let t be the point in time where job i starts to be processed in S, and let $\bar{t}$ be the point in time where $\bar{S}$ begins to process job k. Moreover, denote

$$\Delta_{ij} := S_{ij} - S, \qquad \Delta_{jk} := S_{jk} - S, \qquad \bar{\Delta}_{ij} := \bar{S}_{ij} - \bar{S}, \qquad \bar{\Delta}_{jk} := \bar{S}_{jk} - \bar{S} \ .$$

Exploiting the convexity* of f we can calculate

$$\frac{\Delta_{jk}}{\Delta_{ij}} = \frac{p_j \cdot (f(t + p_i + p_j + p_k) - f(t + p_i + p_j))}{p_i \cdot (f(t + p_i + p_j) - f(t + p_i))} \geq^* \frac{p_j}{p_i}\frac{p_k}{p_j} = \frac{p_k}{p_i} \tag{2.20}$$

and

$$\frac{\bar{\Delta}_{jk}}{\bar{\Delta}_{ij}} = \frac{p_k \cdot (f(\bar{t} + p_k + p_j) - f(\bar{t} + p_k))}{p_j \cdot (f(\bar{t} + p_k + p_j + p_i) - f(\bar{t} + p_k + p_j))} \leq^* \frac{p_k}{p_j}\frac{p_j}{p_i} = \frac{p_k}{p_i} \ . \tag{2.21}$$

Therefore

$$\frac{\Delta_{jk}}{\bar{\Delta}_{jk}} \geq^* \frac{\Delta_{ij}}{\bar{\Delta}_{ij}} \ . \tag{2.22}$$

Now again using that the proposition $A \vee B$ is logically equivalent to $\neg A \Rightarrow B$, the lemma is shown by the following implication chain.

$$\begin{aligned} \frac{S}{\bar{S}} <^* \frac{S_{ij}}{\bar{S}_{ij}} = \frac{S + \Delta_{ij}}{\bar{S} + \bar{\Delta}_{ij}} \quad &\Longrightarrow \quad \frac{S + \Delta_{ij}}{\bar{S} + \bar{\Delta}_{ij}} \leq^* \frac{S + r \cdot \Delta_{ij}}{\bar{S} + r \cdot \bar{\Delta}_{ij}} \quad \text{for } r \geq 1 \\ &\Longrightarrow \quad \frac{S_{ij}}{\bar{S}_{ij}} = \frac{S + \Delta_{ij}}{\bar{S} + \bar{\Delta}_{ij}} \leq^* \frac{S + \Delta_{jk}}{\bar{S} + \bar{\Delta}_{jk}} = \frac{S_{jk}}{\bar{S}_{jk}} \ , \end{aligned}$$

where the last implication follows with $r = \Delta_{jk}/\Delta_{ij}$, which is not smaller than 1 due to equation (2.20) and $p_k > p_i$, and by additionally utilizing equation (2.22).

In the case of concave cost functions, the last implication follows from the concave variant of (2.22) and the fact that $\bar{\Delta}_{jk} \geq \bar{\Delta}_{ij}$, which is a consequence of the concave variant of (2.21) and again $p_k > p_i$. This completes the proof of the claim.

In what follows, we finally prove Theorem 2.17. From Observation 2.16 we know that the worst case ratio is always established by problem instances where each job has a density of 1, the worst possible Smith's rule schedule is equal to INC* and the optimal schedule is equal to DEC*. Thus, it suffices to show the existence of a worst case instance that has at most one job whose processing time and weight is strictly between 1 and $p_{\max}$. We denote the latter kind of jobs as *medium jobs*.

We use a potential function Θ mapping problem instances to integers in our analysis. Defining $p_{\text{med}}(I)$ as the total processing time of all medium jobs in instance I and $n_{\text{med}}(I)$ as their number, define

$$\Theta(I) \;:=\; (P+1)\cdot p_{\text{med}}(I) - n_{\text{med}}(I)\,.$$

We will show that for any problem instance I with more than one medium job there is another problem instance I' with $\text{INC}(I')/\text{DEC}(I') \geq^* \text{INC}(I)/\text{DEC}(I)$ and $\Theta(I') < \Theta(I)$. As the potential $\Theta(I)$ is integral and clearly upper and lower bounded for fixed $p_{\max}$ and P, it follows that there is a worst case instance with at most one medium job. The potential function has been chosen such that changes in p_{med} always dominate any change in n_{med}. This is true because n_{med} is upper bounded by P.

Consider a problem instance I with more than one medium job, and let x and y be the first two jobs that are scheduled consecutively by INC with $1 < p_x \leq p_y < p_{\max}$. Define $p_z := \min\{p_{\max} - p_y, p_x - 1\}$, so that decreasing p_x by p_z and increasing p_y by p_z has the effect that p_x becomes 1 or p_y becomes $p_{\max}$. The resulting problem instance I' remains legal with respect to parameters $p_{\max}$ and P. Moreover, it holds that $\Theta(I') < \Theta(I)$ because the total processing time of all medium jobs decreases.

Let further I'' be the problem instance obtained from I by splitting x into a job of processing time and weight $p_x - p_z$ and a job of processing time and weight p_z. Observe that also I'' is a legal problem instance. Also here $\Theta(I'') < \Theta(I)$ because either one of the two new jobs is a processing time 1 job and thus the total processing time of all medium jobs decreases, or both new jobs are medium jobs and thus p_{med} does not change and n_{med} increases.

It remains to show that one of these two modifications does not decrease* the ratio INC/DEC. We assume here that the positions of the jobs x and y in $\text{INC}(I')$ and $\text{DEC}(I')$ remain the same as in $\text{INC}(I)$ and $\text{DEC}(I)$. This is without loss of generality, because considering the actual schedules $\text{INC}(I')$ and $\text{DEC}(I')$ instead would only further increase* $\text{INC}(I')/\text{DEC}(I')$. For the same reason it can be assumed that $\text{INC}(I'')$ and $\text{DEC}(I'')$ are obtained from by the original schedules for I by replacing x with the two new jobs.

We can directly apply the claim with $\text{INC}(I) = S_{ij}$, $\text{INC}(I') = S_{jk}$, and $\text{INC}(I'') = S$ to show that

$$\frac{\text{INC}(I)}{\text{DEC}(I)} \leq^* \frac{\text{INC}(I')}{\text{DEC}(I')} \qquad \text{or} \qquad \frac{\text{INC}(I)}{\text{DEC}(I)} \leq^* \frac{\text{INC}(I'')}{\text{DEC}(I'')}\,. \qquad \square$$

Chapter 3

Complexity and exact algorithms for min-sum scheduling

In this chapter, we address complexity issues as well as exact algorithms for single machine scheduling with generalized min-sum objective. Concerning the former, we give a first strong NP-hardness result. While so far only the weak NP-hardness for convex cost was known, we now show that the problem is strongly NP-hard for a very restricted class of piece-wise linear cost functions, which, in terms of the interpretation as processor speed, translates into a setting with two alternating speed levels. Turning toward exact algorithms, we discuss structural insights in the form of order rules which help to speed up or to allow exact computations in the first place. Throughout the past decades, great effort has been made to develop fast exact algorithms for the case of quadratic cost functions, and in this context a considerable number of different kinds of order rules have been proposed. Following this line of research, we also address the case of quadratic cost. For this cost function, we enhance the map of known order rules by proving an extended version of a rule that has been conjectured by Mondal and Sen [MS00] more than a decade ago.

Publication remark: The results in this chapter are based on joint work with Tobias Jacobs which was presented at the *Proceedings of the 14th Meeting on Algorithm Engineering & Experiments* (ALENEX 2012) [HJ12a]. Parts of the results are also based on [HJ12b].

The computational complexity of single machine scheduling under a generalized min-sum objective is a long standing open problem. Prominent examples for which the complexity is completely unknown are quadratic or concave cost functions, see e.g., [MS00, ABMP10, SW10]. The few cases for which the complexity is (partially) understood are linear and exponential cost functions, for which the problem is in P [Smi56, RS84], and convex cost functions, for which the problem was shown to be weakly NP-hard [Yua92], leaving open the question of strong NP-hardness. In fact, so far, no problem variant was known for which the problem is strongly NP-hard. In Section 3.1, we give first results in that direction by showing strong NP-hardness for piece-wise linear and monotone cost functions.

In the two sections thereafter, we turn towards exact algorithms for the problem variant with quadratic cost functions. This topic has already been studied for several years by different groups of people, see e.g., [Ali93, BK80, DCST+95, GS84, MS00, RS84, SF62, Tow78]. Most of these works utilize branch-and-bound approaches with pruning rules based on order rules. We extend this set of known rules by proving an extended version of a global order rule conjectured by Mondal and Sen [MS00] in 2000. The resulting improvement to the map of known order rules is illustrated in Figure 3.2. After

the first publication of the results presented here, Dürr and Vasquez [DV14] succeeded to close the remaining gap by showing that in fact the local order relations in Figure 3.2 (b) hold on a global level as well. The influence of the different rules on the performance of various algorithms is evaluated in an extensive experimental study in Chapter 4.

3.1 Hardness for piece-wise linear cost functions

In this section we show the strong NP-hardness of the problem $1 \,||\, \sum w_j\, f(C_j)$ for piece-wise linear and monotone cost functions. In particular, it suffices for the hardness that f alternates between two different slopes that can be chosen arbitrarily. As mentioned earlier, this correlates to a processor running at two different speeds. The result is proven via reduction from the strongly NP-complete 3-Partition problem.

3-Partition	
Given:	Set A of $3\,m$ elements from $\mathbb{N}^+$ with $B/4 < a < B/2$ for all $a \in A$, where $B := \frac{1}{m} \sum_{a \in A} a$.
Task:	Decide whether A can be partitioned into m disjoint sets $A_1, \ldots, A_m$ with $\sum_{a \in A_i} a = B$ for $i = 1, \ldots, m$.

Theorem 3.1 *The problem $1 \,||\, \sum w_j f(C_j)$ is strongly NP-hard for piece-wise linear, monotone cost functions f.*

Proof. Given an instance A of 3-Partition, we construct an instance I of $1 \,||\, \sum w_j f(C_j)$ with piece-wise linear, monotone cost function f. For each element $a_j \in A$, $j = 1, \ldots, 3\,m$, we add a job j in I having processing time $p_j = a_j$ and weight $w_j = a_j$. The cost function f is defined to be piece-wise linear, alternating between two different slopes r and s with arbitrary $r > s \geq 0$. For each $\ell \in \mathbb{N}^+$ the slope during time interval $[(\ell-1)\,B,\ (\ell-1)\,B+1[$ is r, and the slope during $[(\ell-1)\,B+1,\ \ell\,B[$ is s; see Figure 3.1. The cost threshold is set to

$$C \;:=\; s \cdot \sum_{1 \leq i \leq j \leq 3m} a_i\, a_j \;+\; \frac{(r-s)\,B\,m\,(m+1)}{2}\ .$$

The equivalence of the problems is established by showing that any schedule where some new job begins at time $(\ell-1)\,B$ for each $\ell = 1, \ldots, m$ has cost C, and every other schedule has larger cost. This will complete the proof.

As the processing times are integers, no job ever ends inside a slope r interval. Therefore we can as well assume that the slope is s everywhere, and at each time $(\ell-1)\,B$ for some $\ell \in \mathbb{N}^+$ there is a point of discontinuity where the constant $(r-s)$ is added to the cost function. So f can be expressed as

$$f(t) \;=\; s \cdot t \;+\; (r-s) \cdot \left\lceil \frac{t}{B} \right\rceil\ .$$

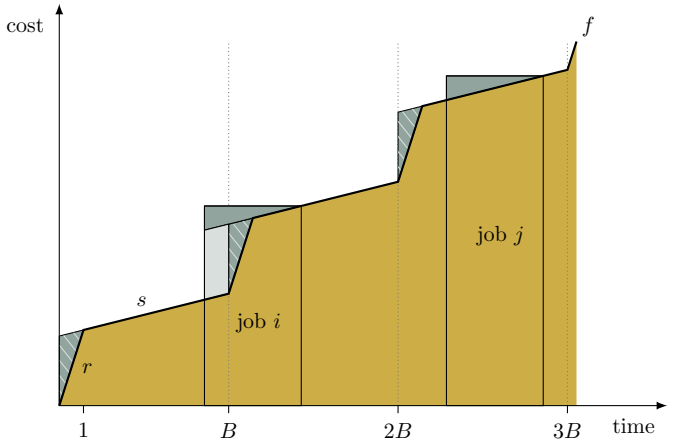


Figure 3.1: Geometric interpretation of instances as they occur in the reduction in the proof of Theorem 3.1. See Section 2.2 for a general explanation of this interpretation. Every schedule's cost is lower bounded by the yellow area under the graph of f. Due to the integrality of the processing times in the reduction, additionally, every schedule has to pay for the striped area. Moreover, every job j accounts for cost to the amount of its corresponding darkgray triangle, no matter when it is scheduled. However, if, as for job i, some multiple of B lies in the interior of its processing interval, the lightgray area needs to be paid for in addition. 3-Partition is reduced to the question whether or not there is a schedule without any such lightgray area.

Let $f = f_1 + f_2$ with $f_1 : t \mapsto s \cdot t$ and $f_2 : t \mapsto (r-s)\lceil t/B \rceil$. As $w_j = p_j$, the cost of a schedule σ with respect to f_1 is

$$\sum_{j=1}^{3m} w_{\sigma(j)} \cdot s \sum_{i=1}^{j} w_{\sigma(i)} \;=\; s \cdot \sum_{1 \le i \le j \le 3m} a_i \, a_j \;.$$

This expression is independent of the order in which the jobs are scheduled, and it is equal to the first summand of C. Thus, for minimizing the cost with respect to f we can ignore f_1 and determine a schedule minimizing the cost with respect to f_2.

Function f_2 can be further split up into $f_2 = f_2^1 + f_2^2 + \ldots$, where

$$f_2^\ell(t) \;:=\; \begin{cases} 0 & , \; t \le (\ell-1)\,B \\ r-s & , \; t > (\ell-1)\,B \;. \end{cases}$$

For $\ell = 1, \ldots, m$, let W_ℓ be the total weight of all jobs with completion time greater than $(\ell-1)\,B$. As the total processing time and weight of all jobs is $m\,B$, it clearly holds that $W_\ell \ge (m-\ell+1)\,B$ and so the cost of every schedule with respect to f_2^ℓ is $(r-s)\,W_\ell \ge (r-s)(m-\ell+1)\,B$. Furthermore, this holds with equality if and only if a new job starts at time $(\ell-1)\,B$. Therefore, the total cost with respect to f_2 is at least

$$\sum_{\ell=1}^{m} (r-s)(m-\ell+1)\,B \;=\; \sum_{\ell=1}^{m} B\,\ell\,(r-s) \;=\; \frac{(r-s)\,B\,m\,(m+1)}{2} \;,$$

which is exactly the second summand of C, and this cost is only reached if a new job starts at each time $(\ell-1)\,B$ for $\ell = 1, \ldots, m$. □

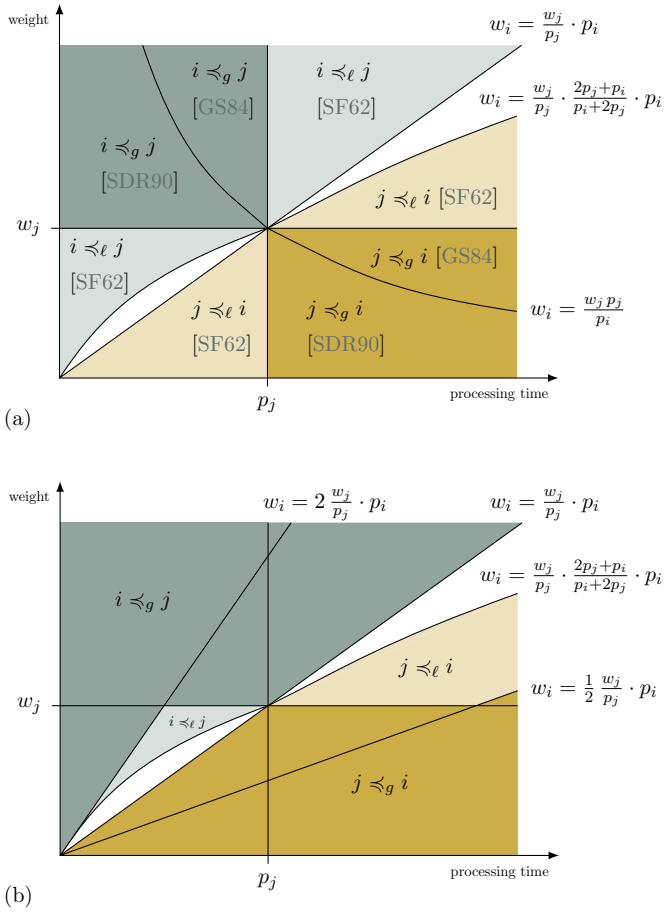


Figure 3.2: The figures show the known order rules for the problem $1 \,||\, \sum w_j C_j^2$ in comparability maps with respect to a fixed job j which is compared to any other job i. In such a map, each job is represented by a point in $\mathbb{R}^2$ where the first component is its processing time and the second its weight. For jobs in the white area, even the local order depends on the time they are scheduled. Figure (a) shows the results known from previous research, whereas Figure (b) integrates the new insights of this thesis.

3.2 Global order rules

This section is majorly dedicated to the proof of the global order rule conjectured by Mondal and Sen [MS00], or rather an extended version of it as we will see in Section 3.2.2. Before, in Section 3.2.1, we introduce various types of order rules whose benefits we will evaluate in our experiments in Chapter 4.

3.2.1 Types of order rules

We consider five types of order rules, three of which are of the type introduced in Definition 1.2, i.e., they only depend on the parameters of the two jobs involved. For the fourth rule, decisions depend on preprocessing steps made for the whole instance. The last rule will finally combine the first four rules. In the following, the rules are described in more detail, see also Figure 3.2 and Table 3.1.

In the descriptions of the rules, we will refer to two general jobs i and j with processing times p_i and p_j and weights w_i and w_j, respectively.

rule	condition	validity	
basic	$p_i \leq p_j \ \wedge \ w_i \geq w_j$	holds globally for f monotonically non-decreasing	[SDR90]
weight	$w_i \geq w_j \ \wedge \ w_i/p_i \geq w_j/p_j$	holds globally for $f(t) = t^2$ holds locally for f convex	[Thm. 3.2] [Lem. 3.3]
β-gap	$w_i/p_i \geq \beta \cdot w_j/p_j$	2-gap holds globally for $f(t) = t^2$ holds locally for polynomials with positive coefficients of degree $\beta > 1$	[Thm. 3.2] [Lem. 3.4]
decomposition	–	holds globally for $f(t) = t^2$	[Szw98]
combined	–	holds locally/globally whenever all above rules do so	

Table 3.1: Types of order rules. The *condition* is formulated in such a way that the stated term implies that i locally/globally precedes j in terms of Definition 1.2.

Basic rule. Intuitively speaking, the basic rule states that short and important jobs have to be scheduled before long and less important ones. Formally, this can be formulated as follows. Whenever $p_i \leq p_j$ and $w_i \geq w_j$ then i globally precedes j, i.e., $i \preccurlyeq_g j$. Sen et al. [SDR90] showed that this rule holds for every non-decreasing cost function. The proof is straightforward by using a simple exchange argument.

Weight rule. The weight rule is satisfied whenever $w_i \geq w_j$ and $w_i/p_i \geq w_j/p_j$ implies that $i \preccurlyeq_\ell j$ or even $i \preccurlyeq_g j$. The global variant of this rule has been conjectured by Mondal and Sen [MS00] for quadratic cost, and we prove it in Theorem 3.2 while proving in Lemma 3.3 that the local weight rule holds for convex cost functions in general.

Note that an inverse variant of the local weight rule holds for concave functions. In this rule the condition $w_i/p_i \geq w_j/p_j$ is replaced by $w_i/p_i \leq w_j/p_j$ and the proof follows analogously to Lemma 3.3.

β-gap rule. A β-gap rule is satisfied whenever $w_i/p_i \geq \beta \cdot w_j/p_j$ implies that i has to be scheduled locally/globally before j in an optimal schedule. In Theorem 3.2, we show that quadratic cost functions observe the global variant of the 2-gap rule, while in Lemma 3.4, we argue that the local β-gap rule is satisfied by the more general class of polynomials with positive coefficients and degree $\beta > 1$. Recall that this property has been used in the proofs of Section 2.1.

Decomposition. The decomposition technique has been proposed by Szwarc [Szw98]. It is particularly designed for quadratic cost functions, and it is majorly based on local comparability. However, it is not a proper order rule in terms of Definition 1.2, since it requires preprocessing steps that depend on the whole instance.

Recall that $i \preccurlyeq_\ell j$ means that in any optimal schedule where the jobs i and j are adjacent, i must precede j. For quadratic cost functions, the defining inequality of local comparability (1.1) implies that for any pair of adjacent jobs i, j there is at most one point in time t_{ij} where the local order changes, i.e., i must be processed before j when the first of these jobs starts before time t_{ij}, and vice versa when they are processed after t_{ij}. This critical point is given by a value of t that satisfies (1.1) with equality, which is unique in case of quadratic cost.

Let P be the total processing time of all jobs in a problem instance. Any schedule takes place within the time interval $[0, P]$. Therefore $i \preccurlyeq_\ell j$ holds if $t_{ij} > P - p_i - p_j$, and $j \preccurlyeq_\ell i$ holds whenever $t_{ij} < 0$. In the first step, the decomposition method determines for all job pairs i, j if one of these two relations hold. It then tries to decompose the instance into two subinstances by deriving global order rules from the local ones. Assume for simplicity that the jobs are indexed such that $w_1/p_1 \geq \ldots \geq w_n/p_n$. The decomposition method tries to identify a job k such that $i \preccurlyeq_\ell j$ for all $i \leq k < j$. When such a job is found, it is clear that in any optimal schedule all jobs $1, \ldots, k$ are scheduled before the jobs $k+1, \ldots, n$. In other words, the global order rule $i \preccurlyeq_g j$ holds for every $i \leq k < j$. We then also know that $1, \ldots, k$ are processed inside the time interval $[0, P']$ with $P' := p_1 + \ldots + p_k$, and the jobs $k+1, \ldots, n$ are scheduled in the interval $[P', P]$. Now the decomposition procedure is recursively applied to these two job subsets and respective time intervals. Note that the more restricted time intervals can lead to new local order relations.

Combined rule. This rule is simply a combination of the four rules above. If any of those rules implies an ordering relation, then so does the combined rule.

3.2.2 Proof of local and global order rules

In this section, we will prove the global weight and 2-gap rule for quadratic cost functions, the former being conjectured by Mondal and Sen [MS00]. Along the way, we will also show that the local weight rule is satisfied by convex cost functions and the β-gap rule by polynomials with positive coefficients of degree β.

Theorem 3.2 *Problem $1 \,||\, \sum w_j C_j^2$ observes the global weight rule and the global 2-gap rule, i.e., for any pair of jobs i and j, we have $i \preccurlyeq_g j$ if one of the following sufficient conditions hold:*

(a) $w_i \geq w_j$ *and* $\dfrac{w_i}{p_i} \geq \dfrac{w_j}{p_j}$, *or*

(b) $\dfrac{w_i}{p_i} \geq 2 \cdot \dfrac{w_j}{p_j}$.

Furthermore, it holds that $i =_\ell j$ or $i =_g j$ if and only if $p_i = p_j$ and $w_i = w_j$.

Proof. Both rules are proven simultaneously by induction on the number of jobs that are scheduled between a job pair satisfying the respective rule. More specifically, we show for increasing values of ℓ that there is no optimal schedule where for a pair of non-identical jobs i, j satisfying (a) or (b) the job j is processed before i and there are at most ℓ other jobs between them. The base case $\ell = 0$ are the local variants of both rules. Their proofs are given in Lemma 3.3 and 3.4 for more general cost functions.

By the induction hypothesis, we assume that the theorem holds when, for some fixed ℓ, there are less than ℓ jobs scheduled between two jobs i and j satisfying property (a) or (b). Based on these assumptions, we first show the induction step for (a) in Lemma 3.5. This lemma is then used in the induction step for (b) in Lemma 3.6. The proof of these two lemmas will complete the proof of the two global rules.

Finally, note that $i =_g j$ implies $i =_\ell j$ and thus, inequality (1.1) to be satisfied with equality for any $t \geq 0$. In case of quadratic cost, this is equivalent to

$$2\,t(w_j\,p_i - w_i\,p_j) \;=\; w_i\,p_j(p_j + 2\,p_i) - w_j\,p_i(p_i + 2\,p_j)\,,$$

which in turn implies that i and j are identical. □

We proceed to prove the two local base case lemmas.

Lemma 3.3 *Problem* $1 \,||\, \sum w_j\, f(C_j)$ *with convex cost function* f *observes the local weight rule.*

Proof. Whenever $i \preccurlyeq_\ell j$ holds for jobs i and j, then it still holds after the processing time of i has been made shorter. In terms of the weight rule this allows us to assume w.l.o.g. that $w_i/p_i = w_j/p_j$ instead of $w_i/p_i \geq w_j/p_j$. Thus, either the two jobs are identical, or $w_i > w_j$, and hence, $p_i < p_j$. Using these assumptions, we can now rewrite the defining inequality for local comparability in (1.1) as

$$\frac{f(t+p_i+p_j)-f(t+p_j)}{p_i} \;\leq\; \frac{f(t+p_i+p_j)-f(t+p_i)}{p_j}\,,$$

where $t \geq 0$ is some arbitrary point in time. We now define the function

$$h : x \mapsto f(t+p_i+p_j) - f(t+p_i+p_j-x)\,.$$

As the first term is a constant and f convex on $\mathbb{R}_{\geq 0}$, the function h is concave on $\mathbb{R}_{\geq 0}$. Utilizing h, the above inequality reads as $h(p_i)/p_i \leq h(p_j)/p_j$, which holds true due to the concavity of f and $p_i < p_j$. □

Lemma 3.4 *Problem* $1 \,||\, \sum w_j\, f(C_j)$ *observes the local* β*-gap rule for polynomials* f *with positive coefficients and degree* $\beta > 1$.

Proof. In order to prove the lemma, we first show that it suffices to assume f to be a monomial $t \mapsto t^\beta$. Consider an arbitrary monomial that is contained in f and which satisfies the gap rule with respect to its degree. By the definition of local comparability, we know that inequality (1.1) holds for this particular monomial, and this is the case for all monomials. Hence, choosing the most restrictive gap rule, i.e., the one corresponding to the highest degree, it follows by summation that (1.1) is also satisfied for f. Furthermore, the leading coefficient can be omitted due to the standard scaling argument.

Now, assuming f to be a monomial $t \mapsto t^\beta$, we need to show that given two jobs i and j with $w_i/p_i \geq \beta \cdot w_j/p_j$ it holds that $i \preccurlyeq_\ell j$. Due to the scalability Observations 2.3 and 2.10, we can assume w.l.o.g. that $w_j = p_j = 1$ and hence, $w_i/p_i \geq \beta$. Furthermore, since f is convex, the general local weight rule of the previous lemma implies the correctness of the current lemma for the case $p_i \geq 1$ already, since this implies $w_i > w_j$. Hence, it suffices to prove the lemma for $p_i < 1$.

Analogously to the previous lemma, we consider the defining condition (1.1) for $i \preccurlyeq_\ell j$. Resolving it for w_i, we get the necessary and sufficient condition

$$w_i \geq g_t(p_i) \quad \text{for each } t \geq 0, \text{ where} \quad g_t : p_i \mapsto \frac{(t+p_i+1)^\beta - (t+1)^\beta}{(t+p_i+1)^\beta - (t+p_i)^\beta}\,.$$

If we interpret job i as the point $(p_i, w_i) \in \mathbb{R}^2$, the function curve of g_t represents all realizations of job i where (1.1) is satisfied with equality for that specific value of t, and all values of (p_i, w_i) lying strictly above all these curves guarantee $i \preccurlyeq_\ell j$. It is sufficient to show that the straight line $g : p_i \mapsto \beta\, p_i$ lies completely above g_t for any t and $p_i < 1$, because any point (p_i, w_i) with $\frac{w_i}{p_i} \geq \beta$ is located on or above g.

We rewrite g_t as g and some additive term. It then remains to show that the additive term is negative

$$g_t(p_i) \;=\; g(p_i) \;-\; \frac{p_i}{(t+p_i+1)^\beta - (t+p_i)^\beta} \cdot h_t(p_i)$$

with

$$\begin{aligned} h_t(p_i) \;&:=\; \beta\,\Big((t+p_i+1)^\beta - (t+p_i)^\beta\Big) \;-\; \frac{(t+p_i+1)^\beta - (t+1)^\beta}{p_i} \\ &\geq\; \beta\,\Big((t+p_i+1)^\beta - (t+p_i)^\beta\Big) \;-\; \beta\,(t+p_i+1)^{\beta-1} \\ &=\; \beta\,(t+p_i)\,\Big((t+p_i+1)^{\beta-1} - (t+p_i)^{\beta-1}\Big) \;\geq\; 0\,, \end{aligned}$$

where the first estimate uses the fact that the last term describes a slope which is not larger than the slope of function $x \mapsto x^\beta$ in point $t+p_i+1$. Finally, this shows that $g_t(p_i) \leq g(p_i)$ which completes the proof. □

In the remainder of this section we prove the two lemmas which cover the induction step of Theorem 3.2.

Lemma 3.5 *Assume that for some fixed ℓ Theorem 3.2 holds for any pair of jobs i, j which satisfy (a) or (b) and between which less than ℓ other jobs are scheduled. Then rule (a) holds also for the case when there are ℓ jobs scheduled between i and j.*

Proof. Assume for contradiction that there is an optimal schedule where job pair i, j satisfies (a), job j is processed before i, there are ℓ jobs in between, and i and j are not identical. We make four simplifying assumptions. Firstly, we assume that $w_j = p_j$. This is without loss of generality, because the optimality of a schedule is invariant to weight and cost scaling as argued in Observation 2.10 and 2.3. Secondly, we can assume that i is the very last job, since after removing all jobs scheduled after i we obtain an optimal schedule for the remaining jobs for which we then show the contradiction. Thirdly, we assume w.l.o.g. that $p_i > p_j$. If p_i were smaller or equal to p_j, then, since (a) ensures $w_i \geq w_j$, the global basic rule from Section 3.2.1 would immediately lead to a contradiction. Due to $w_i/p_i \geq w_j/p_j = 1$, it follows moreover that $w_i \geq p_i$. We can even assume $w_i = p_i$, because lowering the weight of i preserves the optimality of the schedule under consideration. In fact, if there was a better schedule for the instance with w_i decreased, a straightforward calculation shows that this schedule would also be less costly for the original instance. We relabel the jobs between i and j by $1, \ldots, \ell$ and refer to the subschedule $1, \ldots, \ell$ by S.

We now define a function Δ measuring the cost increase or decrease when the subschedule S is interchanged with a job k having identical processing time and weight x—the job k being an abstraction of the jobs i and j. The function Δ depends on the jobs in S, the value of x, and the starting time t of k or S, depending on which one is processed first. However, we suppress the dependence on everything but x by writing

$$\begin{aligned} \Delta(x) \;=\; &\sum_{r=1}^{\ell} w_r \Big(t + \sum_{m=1}^{r} p_m\Big)^2 \;+\; x\,\Big(t + \sum_{m=1}^{\ell} p_m + x\Big)^2 \\ &-\; x\,(t+x)^2 \;-\; \sum_{r=1}^{\ell} w_r \Big(t + x + \sum_{m=1}^{r} p_m\Big)^2 . \end{aligned}$$

Whenever Δ is negative (positive), it is strictly cheaper to schedule S before (after) the job k. The function Δ being zero means that both possibilities have equal cost. The second derivative of Δ in x is $2\sum_{r=1}^{\ell}(2\,p_r - w_r)$.

This is the point where we require the induction hypothesis. Property (b) and the fact that $w_j = p_j$ imply that $w_r < 2\,p_r$ for $r = 1, \ldots, \ell$. Therefore, the second derivative of Δ is strictly positive, and hence, Δ is strictly convex and has at most two roots. One of those roots is at $x = 0$, so there is some $x_0 \geq 0$ such that $\Delta(x) < 0$ for $x \in (0, x_0)$, $\Delta(x_0) = 0$, and $\Delta(x)$ is strictly increasing for $x \geq x_0$. Note that possibly $x_0 = 0$ and Δ might be completely non-negative.

The optimal schedule under consideration contains j, S, i as a contiguous subschedule. If $\Delta(p_j) < 0$, then one obtains a cheaper schedule by interchanging j and S, a contradiction to optimality. If $\Delta(p_j) \geq 0$, then $\Delta(p_i) > \Delta(p_j)$. In that case consider the alternative schedule obtained by the following operations: Firstly, interchange j and S, secondly, interchange j and i, and thirdly, interchange S and i. The second operation decreases the cost, due to the local version of the weight rule of Lemma 3.3. The first operation does not decrease the cost due to $\Delta(p_j) \geq 0$, but the increase is more than compensated for by the third operation because $\Delta(p_i) > \Delta(p_j)$. After all, we obtain a cheaper schedule, which contradicts the optimality of the original one. □

Lemma 3.6 *Assume that for some fixed ℓ, Theorem 3.2 holds for any pair of jobs i, j which satisfy (a) or (b) and between which less than ℓ other jobs are scheduled. Then rule (b) holds also for the case when there are ℓ jobs scheduled between i and j.*

Proof. Assume for contradiction that there is an optimal schedule where job pair i, j satisfies (b), job j is processed before i, and there are ℓ jobs in between. Like in the proof of Lemma 3.5, we assume w.l.o.g. that $w_j = p_j = 1$, and we rename the ℓ jobs between j and i to $1, \ldots, \ell$ and use S as an abbreviation for that sequence, assuming no conflicts with the labels of i and j. The job i satisfies $w_i \geq 2\,p_i$. Moreover, we can assume that $p_i < 1$ because otherwise the job pair i, j would satisfy property (a), which is impossible due to Lemma 3.5. Furthermore, is must hold that $w_i < 1$, because otherwise we would have $p_i < p_j$ and $w_i \geq w_j$, and the basic rule from Section 3.2.1 would immediately prove suboptimality. Summarizing, it suffices to analyze the situation $w_j = p_j = 1 > w_i \geq 2\,p_i$.

In the following notation we ignore all jobs scheduled before and after the jobs j, S, i. Slightly abusing notation, we refer to the schedule and also to its cost by $[j, S, i]$, analogously for other permutations of this job subset.

Claim: $\quad [j, i, S] - [j, S, i] \;<\; [i, j, S] - [i, S, j]\,.$

If that claim is true, then the suboptimality of $[j, S, i]$ can be shown by calculating

$$[i, S, j] \;<\; [i, j, S] - [j, i, J_{1\ldots\ell}] + [j, S, i] \;<\; [j, S, i]\ ,$$

where the second inequality is due to the local version of the weight rule from Lemma 3.3.

To prove the claim, let t be the point in time when job i completes in the optimal schedule $[j, S, i]$. Let further p_S be the total processing time of all jobs in S. We regard the cost caused by these ℓ jobs as a function F of the completion time of the last job ℓ.

The left hand side of the claimed inequality accounts for the cost difference of transforming $[j,i,S]$ into $[j,S,i]$. By this transformation the jobs in S become processed earlier, their cost decreasing by g_L (gain), and job i becomes more expensive, requiring to pay an additional amount of ℓ_L (loss). On the right hand side of the inequality, $[i,j,S]$ is transformed into $[i,S,j]$, corresponding to analogous variables g_R and ℓ_R. Recalling that we denote the cost function of the jobs in S by F, these variables compute as

$$\begin{aligned} g_L &:= F(t) - F(t-p_i)\,, & & & \ell_L &:= w_i\left(t^2 - (t-p_S)^2\right), \\ g_R &:= F(t) - F(t-1)\,, & & \text{and} & \ell_R &:= t^2 - (t-p_S)^2\,. \end{aligned}$$

To conclude this proof we require the estimate $g_L < w_i \cdot g_R$, which is equivalent to

$$F(t) - F(t-p_i) \;<\; w_i\left(F(t) - F(t-1)\right).$$

The loss terms satisfy $\ell_L = w_i\,\ell_R$. We are going to show below that $g_L < w_i \cdot g_R$, which implies the claim as follows:

$$\begin{aligned} 0 \;&\leq\; [j,i,S] - [j,S,i] \;&=\;& g_L - \ell_L \\ &<\; w_i \cdot (g_R - \ell_R) \\ &\leq\; g_R - \ell_R \;&=\;& [i,j,S] - [i,S,j]\,. \end{aligned}$$

The first inequality is due to the optimality of schedule $[j,S,i]$, the second one follows from $\ell_L = w_i \cdot \ell_R$ and the above proposition that $g_L < w_i \cdot g_R$, and the third inequality follows from $w_i \leq 1$.

It remains to show the proposition $g_L < w_i \cdot g_R$, which is equivalent to proving

$$F(t) - F(t-p_i) \;<\; w_i\left(F(t) - F(t-1)\right).$$

All we need to know about F is that it is a non-negative quadratic function that is strictly increasing on $[t-1,\infty)$. As we are reasoning about the difference of function values, the same argumentation will hold after the function curve of F has been shifted vertically. We can also shift the function horizontally if we shift the points of evaluation t, $t-p_i$, and $t-1$ along with it. For the sake of simpler calculations, we do transformations of that kind, such that we obtain $t-1 = F(t-1) = 0$. Then F can be written as $F(t) = a\,t\,(t+b)$ with $a > 0$. Furthermore, F cannot have a positive root and therefore $b \geq 0$.

Utilizing $w_i \geq 2\,p_i$ and $t - 1 = 0$, we can write

$$F(t) - F(t-p_i) \;=\; F(1) - F(1-p_i) \;\leq\; a\,(1+b) - a\,\left(1 - \tfrac{w_i}{2}\right)\left(1 - \tfrac{w_i}{2} + b\right),$$

the right hand side of which can be reformulated to

$$a\,(1+b) - a\,\left(1 - w_i + \left(1 - \tfrac{w_i}{2}\right)\,b + \tfrac{w_i^2}{4}\right).$$

This quantity is strictly smaller than

$$a\,(1+b) - a\left(1 - w_i + (1-w_i)\,b\right) \;=\; w_i\,a\,(1+b) \;=\; w_i\,(F(1) - 0)\,,$$

which is equal to $w_i\,(F(t) - F(t-1))$. □

3.3 Exact algorithms

In this section we describe the exact algorithms that we have implemented in order to evaluate the different global order rules. The rules are treated as a black box in this section.

3.3.1 Graph search

We consider an exact algorithm combining the insights of the two major approaches discussed in previous articles. On the one hand, there are standard branch-and-bound methods whose intelligence consists of utilizing order rules, the latest work being by Mondal and Sen [MS00]. On the other hand, Sen, Bagchi and Ramaswamy [SBR96] and Kaindl, Kainz and Radda [KKR01] propose a best-first graph search (BFGS), exploring the graph in forward ($\mathrm{BFGS_f}$) and backward ($\mathrm{BFGS_b}$) manner, respectively. See e.g., [HNR68] for a general description of the very popular best-first search algorithm A^*. The BFGS approaches in [SBR96, KKR01] utilize the lower bound shown by Townsend [Tow78]. However, regarding the use of order rules for pruning the search space, they only exploit the rather weak rule by Bagga and Kalra [BK80] which uses the local version of the weight rule for a limited decomposition of the problem instance. To the best of our knowledge, global order rules have not yet been integrated into BFGS methods for the problem $1\,||\,\sum w_j\,C_j^2$.

Sen, Bagchi and Ramaswamy [SBR96] showed in their experiments that, comparing (forward) best-first graph search $\mathrm{BFGS_f}$, best-first tree search and depth-first search, $\mathrm{BFGS_f}$ is by far the fastest approach on a set of randomly generated instances. The experiments by Kaindl, Kainz and Radda [KKR01] demonstrated that $\mathrm{BFGS_b}$ is even faster. Opting for the fastest variants, we will consider $\mathrm{BFGS_f}$ and $\mathrm{BFGS_b}$ which we further speed up by integrating our rules. However note that for larger instances depth-first search might become the method of choice due to its lower memory consumption.

Graph-based search with A*. We consider $\mathrm{BFGS_f}$ and $\mathrm{BFGS_b}$ based on the best-first search algorithm A^* [HNR68]. We extend the methods proposed in [SBR96, KKR01] by integrating global order rules to prune the search space. When describing the algorithm, we focus on $\mathrm{BFGS_b}$ since it is slightly less intuitive and faster than $\mathrm{BFGS_f}$. At the end of this section, we discuss on the modifications that are necessary for $\mathrm{BFGS_f}$.

In the graph G considered in this context, nodes represent subsets of jobs S. Slightly abusing notation, we refer to the nodes by the corresponding job sets. Let J denote the set of all jobs in the instance. From each node $S \subseteq J$ there is a directed arc to node $S' = S \setminus \{j\}$ for all $j \in S$, i.e., each arc corresponds to a job j, and its cost $c(S, S')$ is set to the cost incurred by job j when scheduling it directly after the jobs in S', i.e.,

$$c\big(S, S'\big) \;=\; c\big(S, S\backslash\{j\}\big) \;=\; w_j\Big(\sum_{i\in S} p_i\Big)^2.$$

Now, each path from J to $\emptyset$ in G naturally corresponds to a schedule for $1\,||\,\sum w_j\,C_j^2$, because the arcs of the path represent a permutation $\pi(1), \pi(2), \ldots, \pi(n)$ of the jobs. As we perform backward search, the corresponding schedule is $\pi(n), \pi(n-1), \ldots, \pi(1)$. In other words, each partial path from J to some node S represents a partial schedule where the jobs $J \setminus S$ are processed after the jobs from S. As the cost of a path equals

the cost of the corresponding schedule, there is a one-to-one relation between minimum cost paths from J to $\emptyset$ and optimal schedules for $1 \,||\, \sum w_j C_j^2$.

The remainder of the algorithm is basically to run A^* on this graph. Three variants of an admissible lower bound heuristic h guiding the search are described below. For the sake of memory efficiency the graph is built dynamically, generating new nodes as we approach them during the search. Furthermore—and that is the point where we extend the algorithms in [SBR96, KKR01]—a node S is infeasible when there is a pair of jobs $i \in J \setminus S$, $j \in S$ with $i \preccurlyeq_g j$. During the search infeasible nodes are ignored.

For the forward variant of $\mathrm{BFGS_f}$, we consider the same graph as in the backward variant but with all edges reversed, and hence, with root node $\emptyset$. Now, path from $\emptyset$ to J correspond to schedules for $1 \,||\, \sum w_j C_j^2$, and here the permutation π does not need to be inverted.

We continue with a discussion of the used lower bounds, and refer to [HNR68] for further details on the A^* procedure itself.

Lower bounds. In order to test the impact of the different order rules without external influence of sophisticated lower bounds, we consider a rather basic lower bound besides Townsend's lower bound [Tow78] which we describe below. In the context of the above graph search procedure, a lower bound is a function h on subsets of jobs S. For the backward variant $\mathrm{BFGS_b}$ this corresponds to providing a bound on the cost of a schedule for the jobs in S, starting at time 0. Inversely, for $\mathrm{BFGS_f}$ this bound needs to be computed with respect to $J \setminus S$ and time $\sum_{j \in S} p_j$. So in fact, h can be formulated as a function on the start time t of a schedule and on the jobs $S \subseteq J$ to be scheduled. The *basic lower bound* (B) simply is given by the total cost of all jobs in S when starting them at time t.

Townsend's lower bound (T) is based on the local weight rule. He first computes a schedule by sorting the jobs by non-increasing density w_j/p_j. Assume for simplicity that this order is $1, \ldots, n$. The optimal schedule can be obtained from that schedule by a series of local interchange operations where the order of two adjacent jobs i, j with $i < j$ is changed to j, i. An upper bound on the cost reduction of such an operation is $(w_j - w_i)\, p_i\, p_j$. Townsend computes this upper bound for each job pair i, j, and whenever it is positive it is subtracted from the cost of schedule $1, \ldots, n$. This gives a lower bound on the optimal cost. The relation with the local weight rule is due to the fact that the amount subtracted for job pair i, j is zero whenever the weight rule for this job pair is satisfied. Also observe that Townsend's lower bound is independent of the time t at which the schedule under consideration starts.

When referring to a combination of a graph search algorithm with a specific lower bound, we add the lower bound's name as superscript to the algorithm name, e.g., $\mathrm{BFGS_f^B}$ or $\mathrm{BFGS_b^T}$.

Finally, we point out a third lower bound, which stems from our investigations in Section 2.3. There, we have provided a tight analysis of Smith's rule with respect to parameters of the instance. Assuming integral processing times, the tight approximation factor α_S for a set of jobs S has been computed with respect to the maximum processing time and the total processing time in S. Now, in order to determine a lower bound on the optimal cost $\mathrm{OPT}(S)$ of a schedule for S (starting at time zero), we utilize this approximation factor. Denoting by $\mathrm{HDF}(S)$ the cost of a Smith's rule schedule for S, by the definition of the approximation factor, it holds that $\mathrm{HDF}(S) \leq \alpha_S \cdot \mathrm{OPT}(S)$. This

yields the lower bound $\mathrm{OPT}(S) \geq \frac{1}{\alpha_S} \cdot \mathrm{HDF}(S)$ to which we refer as *approximation factor lower bound*. Note that even though our analysis in Section 2.3 was only given for the case that the schedule starts at time zero, the analogue results hold when the schedule starts at time $t > 0$.

Implementation note. Since global order rules are evaluated extremely often during a run of the algorithm, we transfer their evaluation to a preprocessing step by computing an $n \times n$ comparability matrix. By this, the time for evaluating rules becomes negligible later on, and moreover, it is independent of the rule used. For checking the feasibility of a node S' reached from a (feasible) node S during the main computation, one only has to test the comparability matrix for the job pairs i, j with $i \in S'$, $j \notin S'$ and $i, j \in S$ ($i, j \notin S$ for forward search). The number of such pairs is smaller than n.

3.3.2 Quadratic integer programming

There are several possibilities to express $1 \,||\, \sum w_j C_j^2$ in terms of a mathematical program. The reason why we have chosen the quadratic integer programming (QIP) formulation below is that it admits to directly add order rules in the form of linear constraints to the program. We used CPLEX 12.1 for its solution.

For each pair of distinct jobs $i < j$ we introduce a binary variable x_{ij} which is defined to be 1 if i is scheduled before j. In order to ensure that any feasible solution represents a total order, we add constraints enforcing transitivity by preventing each triple of jobs to occur in cyclic order. Furthermore, we add a variable y_j for each job j representing its completion time, and a constraint which serves as lower bound on y_j depending on the jobs scheduled before j. The order rules can be easily integrated by fixing the respective variables x_{ij}. The objective function results as $\sum_{j=1}^{n} w_j y_j^2$, where w_j denotes the weight of job j. The full quadratic integer program is given by:

$$\min. \quad \sum_{j=1}^{n} w_j y_j^2$$

$$\begin{aligned}
\text{s.t.} \qquad 1 \leq\ & x_{ij} + x_{jk} + (1 - x_{ik}) \ \leq\ 2 && 1 \leq i < j < k \leq n, \\
\sum_{i<j} x_{ij} p_i + p_j + \sum_{i>j} (1 - x_{ji}) p_i \ \leq\ & y_j && 1 \leq j \leq n \\
x_{ij} \ =\ & 1 && i \preccurlyeq_g j \\
x_{ij} \ =\ & 0 && j \preccurlyeq_g i \\
x_{ij} \ \in\ \{0,1\}, \quad y_j \ \geq\ & 0 && 1 \leq i < j \leq n\ .
\end{aligned}$$

Chapter 4

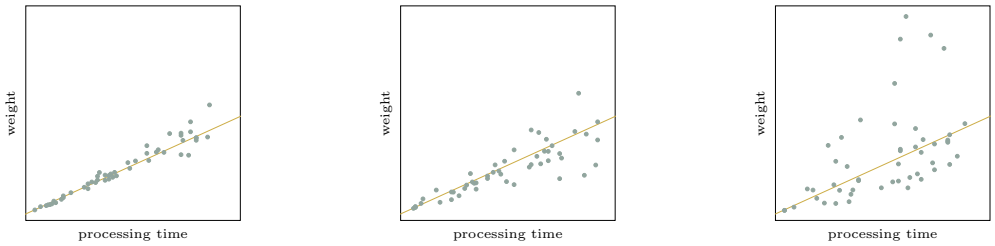
Experiments

In this final chapter of the first part of the thesis, we aim at empirically evaluating the results and insights of the previous chapters. In an extensive computational study we systematically analyze the influence of different global order rules on the performance of exact algorithms in case of quadratic cost functions. Concerning the algorithms, we consider different variants of a best-first graph search algorithm as well as a quadratic integer programming formulation that admits to add order rules as additional linear constraints. Moreover, we compare the performance of Townsend's lower bound and our approximation factor bound. Finally, we also evaluate the empirical approximation guarantee of Smith's rule for different monomial cost functions. Our experiments are based on sets of problem instances that have been generated using a new method which allows us to adjust a certain degree of difficulty of the instances.

Publication remark: As the results of the previous chapter, also the results of this chapter are based on [HJ12a].

After having proved the global weight and 2-gap rule for quadratic cost functions in the previous chapter, we now aim for evaluating their influence on exact algorithms. In an extensive computational study, we systematically analyze the performance of six global order rules in combination with different variants of a graph search procedure and with our quadratic integer program formulation, both introduced in the previous chapter. In addition, we gain first insights concerning the experimental performance of Townsend's lower bound in comparison to our approximation factor bound, showing that, even though our bound is performing well, Townsend's bound seems to be invincible for quadratic cost. Finally, our worst-case analysis of the approximation factor of Smith's rule suggests the natural question for its empirical performance. As we will observe in our experiments, the optimality gap is less than 1.7 % on average for monomials up to degree 5, while, for monomials of degree 2, the gap even below 0.2 %.

To the best of our knowledge, all previous experimental work has been based on instances that were generated by choosing both the weight and the processing time uniformly and independently at random. From Figure 3.2, one would intuitively expect that those instances are easier to tackle than instances with a strong correlation between the processing time and the weight of a job. We have generated our test set using a new method where this correlation can be set as a parameter. Our experiments reveal that the difficulty of problem instances indeed increases with the correlation.



(a) $\xi = 0.2$ (b) $\xi = 0.4$ (c) $\xi = 0.8$

Figure 4.1: Three instances generated for parameters $\xi = 0.2$, 0.4 and 0.8. Each job j is depicted as point $(p_j, w_j) \in \mathbb{R}^2$ with respect to its processing time p_j and its weight w_j. The background line corresponds to jobs with density 1.

set	n	ξ	$p_{\max}$	# inst. per fixed n or ξ	total # inst.
$\mathcal{S}_\xi^{\mathrm{T}}$	25	0.100, 0.101, 0.102, ..., 1.000	100	3	2703
$\mathcal{S}_\xi^{\mathrm{B}}$	20	0.100, 0.101, 0.102, ..., 1.000	100	3	2703
$\mathcal{S}_\xi^{\mathrm{Q}}$	10	0.100, 0.101, 0.102, ..., 1.000	100	3	2703
$\mathcal{S}_n$	1,2,...,35	0.5	100	10	350

Table 4.1: Parameters of the different test sets.

4.1 Data set

Our experiments are based on random instances that have been generated using a combination of uniform and normal distribution. The random instance generator is configured by the three parameters $(n, p_{\max}, \xi)$. Parameter n is the number of jobs in the instance, the parameter $p_{\max}$ specifies the maximum processing time, and $\xi > 0$ controls the correlation of processing times and weights, with a small value indicating strong correlation. The length p_j of each job j is chosen as a uniform random integer between 1 and $p_{\max}$. The weight w_j is then calculated from p_i as $w_j := p_j \cdot 2^{\mathcal{N}(0,\xi^2)}$, where $\mathcal{N}(0, \xi^2)$ indicates that the exponent is chosen according to normal distribution with mean 0 and standard deviation ξ. Note that assuming the minimum processing time to be 1 is without loss of generality when considering monomial cost functions f. This follows from Observation 2.10 which implies time scalability in this case.

Representing each job j as a point $(p_j, w_j) \in \mathbb{R}^2$, Figure 4.1 gives an impression of the structure of instances generated with respect to different values of ξ. The larger the value ξ the further the jobs are spread around the angle bisector. Reconsidering the global order rules depicted in Figure 3.2 (b), one gets an intuition why ξ may be an important characteristic for the difficulty of an instance: If all jobs lie close to the angle bisector, then, for a fixed job, most of the other jobs lie in the area where the known global order rules are not applicable. In contrast, if the jobs are evenly spread over the plane, the rules will fix the relative order of many more job pairs, and by this, they allow to heavily prune the search space. In previous computational experiments, as e.g., performed by Mondal and Sen [MS00], Sen, Bagchi and Ramaswamy [SBR96] or Kaindl, Kainz and Radda [KKR01], processing times and weights have been generated independently at random with respect to uniform distribution. This results in computationally rather simple instances even in the presence of many jobs.

We run our experiments on four data sets. The set $\mathcal{S}_n$ is used to study the behavior of best-first graph search (BFGS) and our quadratic integer programming approach (QIP)

with respect to an increasing number of jobs, both algorithms being described in Section 3.3. The remaining three sets are generated to investigate the impact of ξ. We use the sets $\mathcal{S}_\xi^{\mathrm{B}}$, $\mathcal{S}_\xi^{\mathrm{T}}$, and $\mathcal{S}_\xi^{\mathrm{Q}}$ to analyze BFGS with basic lower bound and Townsend lower bound, and to analyze QIP, respectively. The parameters according to which these sets were generated are summarized in Table 4.1. The parameter ranges have been chosen depending on the memory efficiency of the algorithms, so that the benefits of the different order rules become visible and still the experiments can be performed on a standard PC. As a comparative study of the algorithms is not our main purpose, we believe that using an individual data set for each of them leads to the most meaningful results.

4.2 Experimental results

The main focus of our experiments is on evaluating the influence of different order rules on exact algorithms for quadratic cost functions. Additionally, we compare the performance of Townsend's lower bound with the approximation factor bound, and we assess the quality of solutions computed with Smith's rule in comparison to the worst-case approximation guarantees.

4.2.1 Influence of order rules on exact algorithms

In this section we analyze the influence of the different order rules summarized in Table 3.1 on the exact algorithms introduced in Section 3.3. Concerning the exact algorithms, we (partially) observed the effect that the backward variants of BFGS were more efficient than the forward variants, as already realized by Kaindl, Kainz and Radda [KKR01]. And clearly, using Townsend's lower bound, one outperforms the variants utilizing only the basic bound. For this reason, we consider $\mathrm{BFGS}_{\mathrm{b}}^{\mathrm{T}}$ as the default variant of BFGS in the following discussion, and we additionally investigate $\mathrm{BFGS}_{\mathrm{b}}^{\mathrm{B}}$ and $\mathrm{BFGS}_{\mathrm{f}}^{\mathrm{T}}$ differing from it by the lower bound and the search direction, respectively. Our QIP approach was not able to handle the same instance sizes as BFGS, and moreover, we were facing numerical problems in terms of the QIP claiming optimality of suboptimal solutions. However, this does not suggest that integer programming approaches are less efficient or less appropriate in general. The proposed formulation was rather chosen for its property of allowing to easily add order rules than for its efficiency.

We measure the performance of the different algorithm variants and order rules in terms of the number of generated nodes, as—at least for the BFGS variants—this measure is independent of the concrete machine on which the experiments are run. Recall that the global comparability matrix is computed in a preprocessing step, and the corresponding overhead in terms of runtime is negligible. Due to the performance variability of CPLEX, however, the results of the QIP approach are not machine-independent.

Our results are presented in Figure 4.2 and 4.3 and the corresponding Tables 4.2 and 4.3, demonstrating the performance in dependency of the parameter ξ and of the number of jobs n, respectively. The plots read as a matrix whose columns correspond to the different order rules and its rows to the exact algorithms. As it turned out, the correlation ξ is a significant parameter in this context. Overall, we make the following observations.

Highly correlated instances are generally difficult. The number of generated nodes always seems to decrease with ξ, which can be observed on all plots of Figure 4.2. The first row

of the figure shows that the number of comparable job pairs typically increases with ξ, which partly explains the influence of this parameter. However, also the performance of algorithms using Townsend's lower bound and the performance of QIP improve with ξ even when no rules are utilized.

The number of comparable job pairs influences all algorithms. There is a clear correlation of the number of comparable job pairs and the number of nodes generated by the algorithms. This effect is especially strong, when considering BFGS without the advanced lower bound. Figure 4.2(b) reads almost as a mirrored image of Figure 4.2(a). The correlation is less pronounced for QIP due the additional elaborate techniques utilized within the QIP solver.

The benefit of the 2-gap rule heavily depends on the parameter ξ. The 2-gap rule performs rather bad in general, and in particular bad for small ξ. This can be explained by comparing Figures 3.2 (b) and 4.1, where one can observe that especially for small ξ, there is a high probability that for every job almost all other jobs lie within the "cone" not covered by the 2-gap rule.

The global weight rule leads to good and reliable performance. As already observed by Mondal and Sen [MS00]—and being the reason for conjecturing the global weight rule—comparing the basic rule with the weight rule, one clearly sees the benefit of strengthening the basic rule to the weight rule; see columns 1 & 3 in Figure 4.2. The effectiveness can be explained by the fact that the weight rule allows to compare jobs with similar ratio of weight and processing time; see Figure 3.2 (b). This also is the reason for the weight rule being by far the most efficient rule for small values of ξ. As one can observe from Figure 4.2, in particular the point clouds of the graph search algorithms using this rule are very compact, which indicates that the performance is reliable.

The effectiveness of the decomposition rule has a high variance. From the fifth column of Figure 4.2, one can see different clusterings effects in the plots for the decomposition rule. One observation is that the number of comparable jobs is typically either small or large, and also the performance of the algorithms tends to be either very bad or very good. An explanation of this effect is the recursive nature of the rule: Whenever the problem can be successfully decomposed into two subproblems, there seems to be a high probability that these subproblems can be further decomposed. In particular, many of the low correlation instances could be almost completely decomposed.

One can also observe clusters along certain horizontal lines. In Figure 4.2, the points on these lines correspond to the instances that could be decomposed only once. When such an instance splits into a set of k jobs to be processed first and another set of size $n-k$ to be processed thereafter, this immediately results in $k \cdot (n-k)$ comparable pairs of jobs, corresponding to a horizontal line at exactly this height.

Combining the rules yields the best results. The combined rule of course profits from the advantages of all other rules, and hence, as one would expect, it performs best. For small ξ, the good performance stems from the weight rule, and for large ξ it additionally benefits from the decomposition.

The weight rule turns back the advantage of backward search. Comparing the number of generated nodes by backward and forward search in Table 4.2 (c) and (d), one can observe that in particular for the harder instances with small values of ξ there are settings where $\mathrm{BFGS}_{\mathrm{b}}^{\mathrm{T}}$ generates some hundred thousands nodes less than the $\mathrm{BFGS}_{\mathrm{f}}^{\mathrm{T}}$. This has also been observed by Kaindl, Kainz and Radda [KKR01]. However, as soon as one

utilizes the weight rule, the effect reverses and forward search is slightly better for all settings of ξ. Finding a theoretical explanation for this observation remains an open question.

The effects of the rules are also observed when the number of jobs varies. In order to investigate the asymptotic behavior of the algorithms, a series of tests has been run for instances with fixed $\xi = 0.5$ and varying number of jobs; see Figure 4.3. Also here it turns out that the weight rule leads to good and stable results, which are only improved by the decomposition on some instances.

4.2.2 Comparison of lower bounds

Even though not being the main focus of our experiments, we also give a very brief evaluation of the three lower bounds proposed in Section 3.3. We evaluate the lower bounds, assuming schedules to start at time zero with no jobs scheduled yet, and consider the relative cost to the optimum.

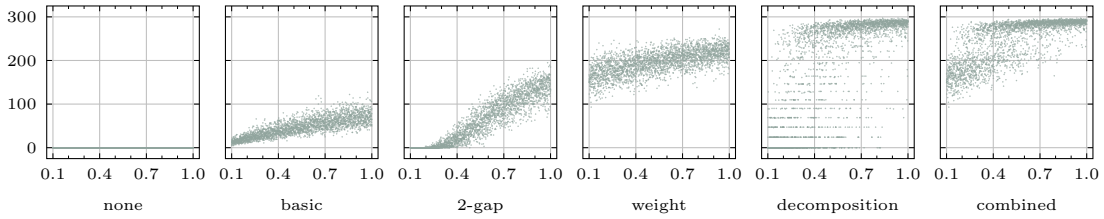
In Figure 4.4 (a), we show the behavior depending on the number of jobs n. Considering the basic lower bound, it gets worse with increasing n, which is obvious as it assumes all jobs to start at time zero. On the other hand, the approximation factor bound gets better with increasing n, which is exactly what one would expect from our parameterized analysis of Smith's rule in Section 2.3: Considering Figure 2.4 (a) and (b), one can observe that for a fixed maximum processing time, the approximation factor gets smaller when the total processing time increases, and inversely, the lower bound gets larger. Even though also slightly showing the effect of performing better for larger instances, Townsend's lower bound is much more stable. Considering the behavior of the bounds depending on the correlation parameter ξ, as shown in Figure 4.4 (b), one can observe that the performance of the bounds is almost independent of ξ.

Finally, comparing the three bounds, Townsend's lower bound clearly outperforms the approximation factor bound, achieving for almost all considered instances factors above 0.97. In contrast, the approximation factor bound gets rarely better factors than 0.96, and, additionally, it is also computationally more expensive. Still, it achieves factors of 0.94 and higher for instances with 25 and more jobs in most of the cases, and the factor seems to be further increasing with n. Moreover note that Townsend's bound is restricted to quadratic cost while the approximation factor bound allows for arbitrary monomial cost. The basic lower bound cannot compete with the other bounds achieving factors under 0.2 on almost all instances.

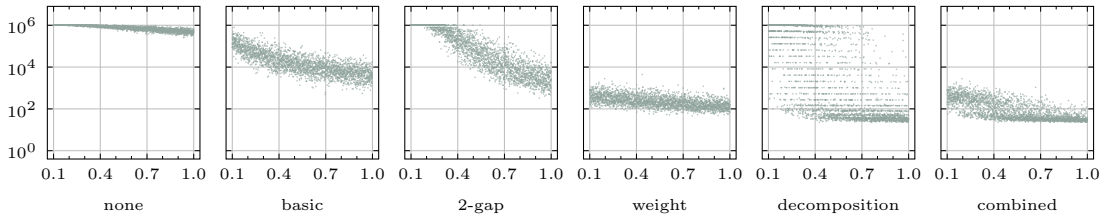
4.2.3 Experimental performance of Smith's rule

In Section 2.2, we have provided a method for computing tight approximation factors of Smith's rule for convex cost functions. For monomials t^2, t^3, t^4 and t^5 we obtain worst-case factors of 1.31, 1.76, 2.31 and 2.93, respectively. Utilizing our parameterized analysis in Section 2.3, these factors further reduce to 1.06, 1.12, 1.18 and 1.23 for the considered test set $\mathcal{S}^{\mathrm{T}}_{\xi}$, respectively.

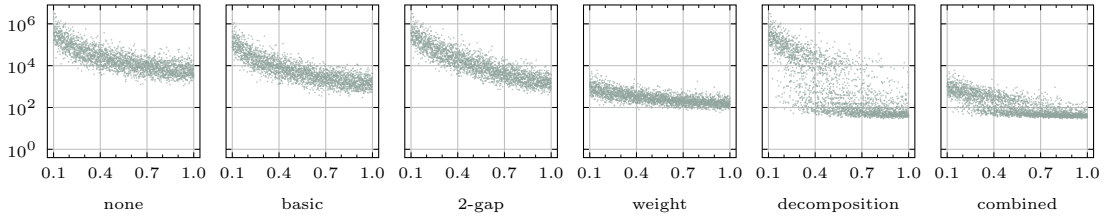
However, the results of our experiments shown in Figure 4.5 and Table 4.4 demonstrate that even these very small worst-case bounds on the performance guarantee are an overestimate of the outcome of the actual cost on non-malevolent instances. In fact, Smith's rule is very often close to optimal. Even though the optimality gap is very small for all considered monomials, one can still observe the approximation ratio slightly degrading for higher degree monomials.



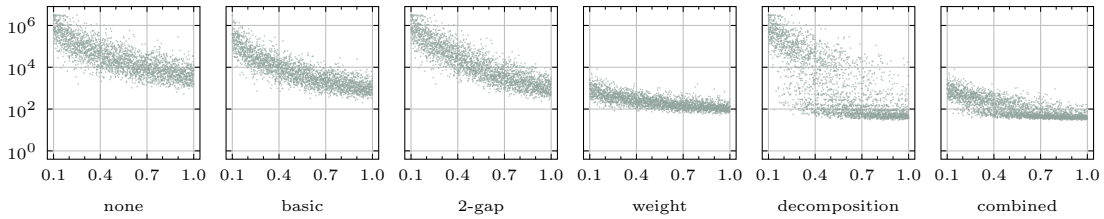
(a) Number of comparable job pairs in dependency of ξ for set $\mathcal{S}_\xi^{\mathrm{T}}$. This parameter is independent of the used algorithm.



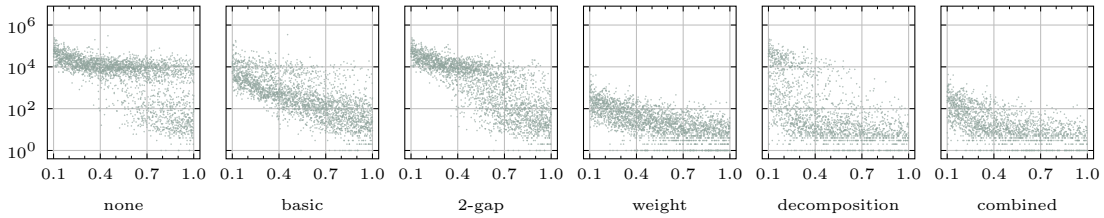
(b) Number of generated nodes with $\mathrm{BFGS}_{\mathrm{b}}^{\mathrm{B}}$ in dependency of ξ for set $\mathcal{S}_\xi^{\mathrm{B}}$.



(c) Number of generated nodes with $\mathrm{BFGS}_{\mathrm{b}}^{\mathrm{T}}$ in dependency of ξ for set $\mathcal{S}_\xi^{\mathrm{T}}$.



(d) Number of generated nodes with $\mathrm{BFGS}_{\mathrm{f}}^{\mathrm{T}}$ in dependency of ξ for set $\mathcal{S}_\xi^{\mathrm{T}}$.



(e) Number of generated nodes with QIP in dependency of ξ for set $\mathcal{S}_\xi^{\mathrm{Q}}$.

Figure 4.2: Study on the influence of ξ on the performance of several variants of algorithms. Each row shows the results for one particular method (except the first one, showing the number of comparable job pairs), whereas the columns correspond to the different rules. Note that except for the first row all plots are shown with respect to logarithmic y-scale. The corresponding numerical values can be found in Table 4.2.

(a) Number of comparable job pairs in dependency of ξ for set $\mathcal{S}_\xi^{\mathrm{T}}$. This parameter is independent of the used algorithm.

	none		decomposition		basic		weight		2-gap		combined	
ξ	∅	σ	∅	σ	∅	σ	∅	σ	∅	σ	∅	σ
[0.1,0.2[	0	0	32	59	17	6	166	24	0	0	171	31
[0.2,0.3[	0	0	82	94	27	9	177	22	2	4	198	40
[0.3,0.4[	0	0	143	111	35	10	183	21	12	11	225	44
[0.4,0.5[	0	0	189	103	42	11	191	21	33	18	247	39
[0.5,0.6[	0	0	210	98	49	13	197	20	57	22	258	34
[0.6,0.7[	0	0	247	74	57	15	206	18	85	22	273	24
[0.7,0.8[	0	0	262	51	61	14	210	20	103	24	278	19
[0.8,0.9[	0	0	269	50	68	16	214	17	122	24	282	15
[0.9,1.0]	0	0	279	27	70	15	219	18	137	22	286	9

(b) Number of generated nodes with $\mathrm{BFGS}_{\mathrm{b}}^{\mathrm{B}}$ in dependency of ξ for set $\mathcal{S}_\xi^{\mathrm{B}}$.

	none		decomposition		basic		weight		2-gap		combined	
ξ	∅	σ	∅	σ	∅	σ	∅	σ	∅	σ	∅	σ
[0.1,0.2[	1034765	11925	676701	420323	133124	98308	461	310	1034061	1985	441	319
[0.2,0.3[	997780	33195	441626	440484	63078	55730	387	256	869509	219662	314	267
[0.3,0.4[	938984	55633	299069	402138	35561	32958	324	201	459519	295785	207	188
[0.4,0.5[	866710	76134	157229	307812	23541	20238	313	308	187798	181561	138	151
[0.5,0.6[	785472	94806	78038	216704	15877	13549	258	173	76318	108707	93	104
[0.6,0.7[	692690	104086	35482	140864	11401	9192	218	137	27924	38837	64	81
[0.7,0.8[	645734	114820	19409	88665	9121	7253	197	122	17504	28448	54	55
[0.8,0.9[	565937	102707	1510	17620	6945	5265	171	110	7203	8949	40	22
[0.9,1.0]	511263	106555	2304	24183	5745	4633	158	95	4302	4770	38	22

(c) Number of generated nodes with $\mathrm{BFGS}_{\mathrm{b}}^{\mathrm{T}}$ in dependency of ξ for set $\mathcal{S}_\xi^{\mathrm{T}}$.

	none		decomposition		basic		weight		2-gap		combined	
ξ	∅	σ	∅	σ	∅	σ	∅	σ	∅	σ	∅	σ
[0.1,0.2[	306370	385620	270536	376950	113563	155257	906	737	306362	385626	889	746
[0.2,0.3[	88879	89949	62858	80947	29724	33245	570	362	87808	89720	506	391
[0.3,0.4[	45671	49102	22735	44584	15396	19229	425	231	40882	45805	292	250
[0.4,0.5[	27925	31294	10445	28004	8358	8858	343	181	21548	25345	187	177
[0.5,0.6[	18636	16088	4686	12170	5409	4387	288	140	12002	12066	130	128
[0.6,0.7[	13208	11866	2355	8895	3530	2789	240	119	6621	7225	84	83
[0.7,0.8[	10262	8081	771	2861	2778	1818	216	131	4186	3076	70	78
[0.8,0.9[	8023	7339	618	3026	2145	1631	193	81	2954	2788	56	39
[0.9,1.0]	8021	8087	209	1039	1921	1529	178	89	2117	1662	48	23

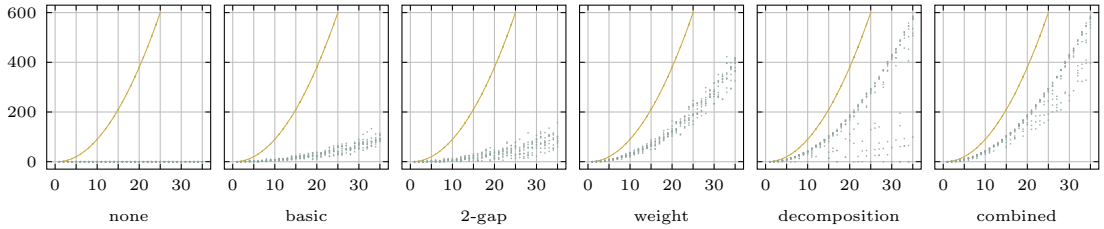
(d) Number of generated nodes with $\mathrm{BFGS}_{\mathrm{f}}^{\mathrm{T}}$ in dependency of ξ for set $\mathcal{S}_\xi^{\mathrm{T}}$.

	none		decomposition		basic		weight		2-gap		combined	
ξ	∅	σ	∅	σ	∅	σ	∅	σ	∅	σ	∅	σ
[0.1,0.2[	737509	807688	659819	791695	207754	300922	853	776	737505	807691	839	783
[0.2,0.3[	225974	339591	168480	311072	49115	73362	489	350	220431	337973	443	368
[0.3,0.4[	112643	158024	66357	135290	22721	31912	359	210	92513	132429	263	220
[0.4,0.5[	58294	99801	24792	87010	10241	13661	271	145	35780	54953	168	148
[0.5,0.6[	33949	50120	11269	32077	6348	6802	227	127	16856	25629	126	127
[0.6,0.7[	18459	31556	3873	19710	3538	3911	183	101	7420	15061	81	78
[0.7,0.8[	14288	19799	1549	7708	2714	2731	169	128	4527	5809	70	77
[0.8,0.9[	8778	8914	1027	4552	1845	1551	143	62	2549	2483	56	35
[0.9,1.0]	7262	17281	335	2019	1418	1716	129	77	1576	1515	49	22

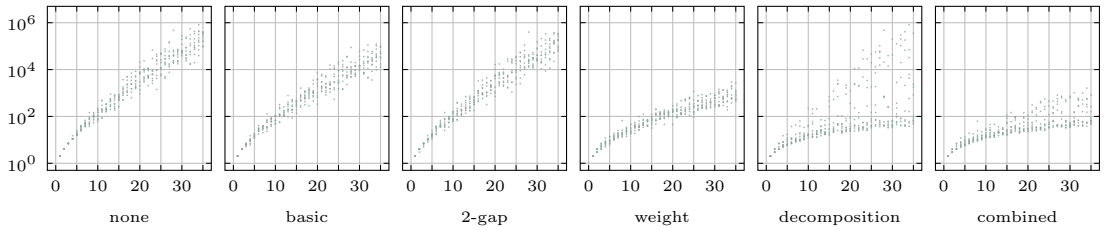
(e) Number of generated nodes with QIP in dependency of ξ for set $\mathcal{S}_\xi^{\mathrm{Q}}$.

	none		decomposition		basic		weight		2-gap		combined	
ξ	∅	σ	∅	σ	∅	σ	∅	σ	∅	σ	∅	σ
[0.1,0.2[	41859	35145	19041	30300	10041	19764	294	473	41887	35129	255	441
[0.2,0.3[	20754	14559	5763	11933	4615	7963	131	171	20696	14530	98	170
[0.3,0.4[	15204	16337	3184	10530	3027	6128	79	115	12782	13518	49	108
[0.4,0.5[	11965	19189	1088	3928	3124	20467	47	59	8244	6951	23	45
[0.5,0.6[	9294	7454	375	2003	1558	3540	31	41	5439	6787	13	34
[0.6,0.7[	7555	7659	126	780	849	2055	24	39	2572	5292	9	20
[0.7,0.8[	6173	8914	29	176	547	1721	18	28	1379	2853	6	15
[0.8,0.9[	4612	7261	15	114	557	2209	14	24	962	2866	4	8
[0.9,1.0]	3060	5596	8	30	270	1399	11	22	391	1531	3	12

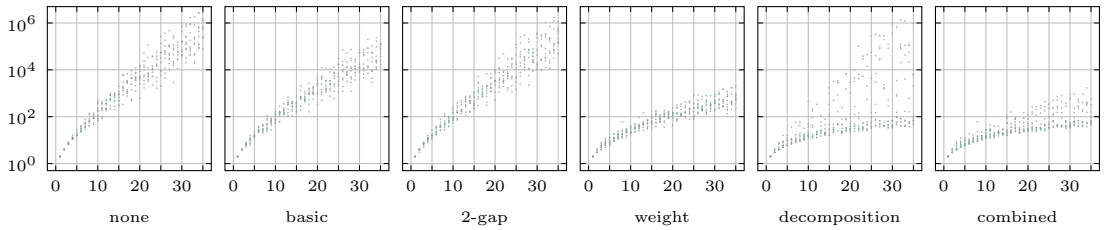
Table 4.2: Numerical values corresponding to the plots in Figure 4.2. Averages have been taken over all results in the respectice parameter range. The average and the standard deviation are denoted by ∅ and σ, respectively.



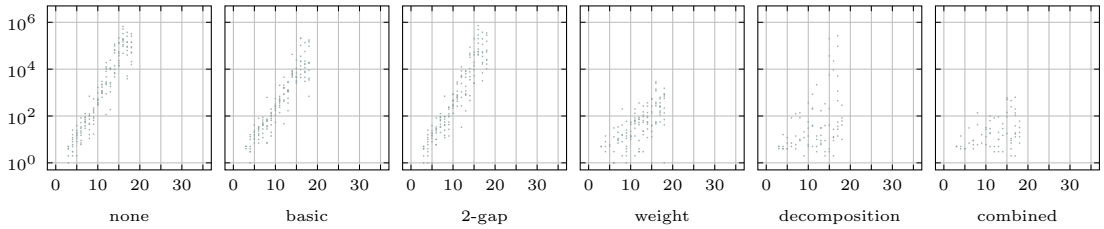
(a) Number of comparable job pairs for in dependency of n for set $\mathcal{S}_n$. The yellow line shows the maximum possible number of comparable pairs.



(b) Number of generated nodes with $\text{BFGS}_\text{b}^\text{T}$ in dependency of n for set $\mathcal{S}_n$.



(c) Number of generated nodes with $\text{BFGS}_\text{f}^\text{T}$ in dependency of n for set $\mathcal{S}_n$.



(d) Number of generated nodes with QIP in dependency of n for set $\mathcal{S}_n$. Due to numerical problems and since QIP did not terminate within reasonable runtime for $n > 18$ the computation was stopped at this point.

Figure 4.3: Study on the influence of n on the performance of several variants of algorithms. Each row shows the results for one particular method (except the first one, showing the number of comparable job pairs), whereas the columns correspond to the different rules. Note that except for the first row all plots are shown with respect to logarithmic y-scale. The corresponding numerical values can be found in Table 4.3.

(a) Number of comparable job pairs for in dependency of n for set $\mathcal{S}_n$. This parameter is independent of the used algorithm.

	none		decomposition		basic		weight		2-gap		combined	
ξ	∅	σ	∅	σ	∅	σ	∅	σ	∅	σ	∅	σ
1–5	0	0	3	3	1	1	3	3	0	1	3	3
6–10	0	0	24	11	4	3	19	8	4	4	25	10
11–15	0	0	62	22	12	5	52	14	13	7	68	17
16–20	0	0	116	47	23	7	99	23	25	13	134	28
21–25	0	0	172	88	39	12	167	31	41	19	213	43
26–30	0	0	253	137	56	14	246	34	55	25	319	65
31–35	0	0	352	185	81	18	340	48	85	26	447	77

(b) Number of generated nodes with $\text{BFGS}_\text{b}^\text{T}$ in dependency of n for set $\mathcal{S}_n$.

	none		decomposition		basic		weight		2-gap		combined	
ξ	∅	σ	∅	σ	∅	σ	∅	σ	∅	σ	∅	σ
1–5	8	6	5	3	7	5	6	3	8	5	5	3
6–10	76	46	21	31	52	28	21	10	63	36	14	8
11–15	400	222	78	125	239	144	54	23	303	175	31	19
16–20	3363	3484	431	780	1268	1246	130	47	2292	2689	55	30
21–25	14150	14712	5201	12483	4513	4623	263	194	9207	8902	147	185
26–30	59986	76865	23683	72717	14199	14875	435	197	39923	57776	219	221
31–35	186898	163108	65884	145362	37033	30402	846	578	118101	100221	341	363

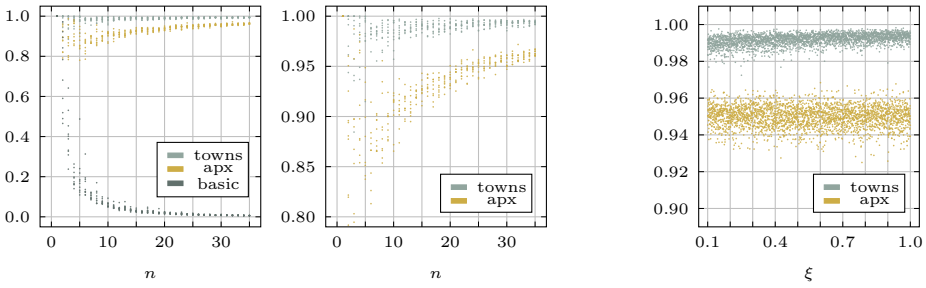
(c) Number of generated nodes with $\text{BFGS}_\text{f}^\text{T}$ in dependency of n for set $\mathcal{S}_n$.

	none		decomposition		basic		weight		2-gap		combined	
ξ	∅	σ	∅	σ	∅	σ	∅	σ	∅	σ	∅	σ
1–5	9	6	5	3	8	5	6	3	8	6	5	3
6–10	86	82	26	62	53	32	20	9	64	47	14	8
11–15	532	351	117	276	259	146	50	23	369	238	30	18
16–20	4157	3486	739	1533	1180	912	110	41	2395	2218	54	30
21–25	25456	43240	12726	40534	4936	5520	203	134	13855	21566	130	140
26–30	119941	174395	59417	152481	17163	18471	350	198	65075	88857	206	207
31–35	544937	761127	188273	540548	49158	50195	599	405	269153	403295	267	256

(d) Number of generated nodes with QIP in dependency of n for set $\mathcal{S}_n$.

	none		decomposition		basic		weight		2-gap		combined	
ξ	∅	σ	∅	σ	∅	σ	∅	σ	∅	σ	∅	σ
1–5	5	8	1	3	4	6	1	2	5	8	1	2
6–10	2804	5044	364	2445	186	367	16	32	1551	3721	8	22
11–15	98024	119358	18824	101553	75204	418929	118	146	69049	96878	51	99
16–18	387835	348327	80588	249440	159864	174137	566	652	331572	325101	151	260

Table 4.3: Numerical values corresponding to the plots in Figure 4.3. Averages have been taken over all results in the respective parameter range. The average and the standard deviation are denoted by ∅ and σ, respectively.



(a) Instances of set $\mathcal{S}_n$. (b) Instances of set $\mathcal{S}_\xi^{\mathrm{T}}$.

Figure 4.4: Comparison of the basic lower bound (basic), Townsend's lower bound (towns) and the approximation factor lower bound (apx) relative to the optimal cost.

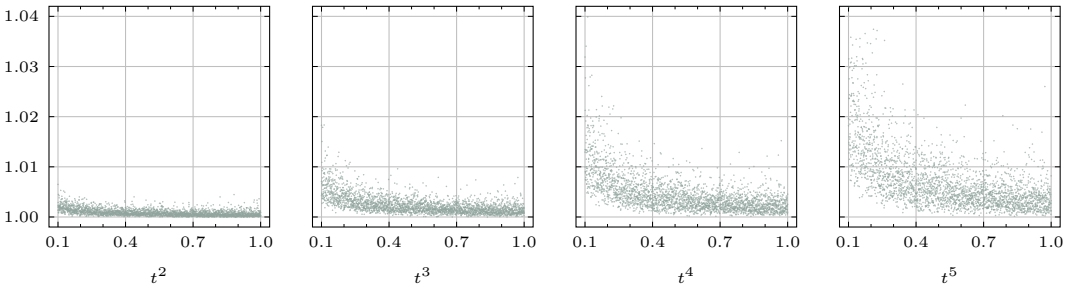


Figure 4.5: Experimental performance guarantee of Smith's rule for the problem $1 \,||\, \sum w_j C_j^k$ on set $\mathcal{S}_\xi^{\mathrm{T}}$, plotted in dependency of the parameter ξ. The plots correspond to the monomial degrees $k = 2, 3, 4$ and 5. The corresponding numerical values can be found in Table 4.4.

	t^2		t^3		t^4		t^5	
ξ	$\varnothing$	σ	$\varnothing$	σ	$\varnothing$	σ	$\varnothing$	σ
$[0.1, 0.2[$	1.00182	0.00092	1.00540	0.00276	1.01044	0.00543	1.01690	0.00884
$[0.2, 0.3[$	1.00123	0.00067	1.00346	0.00178	1.00659	0.00357	1.01063	0.00597
$[0.3, 0.4[$	1.00096	0.00055	1.00263	0.00127	1.00488	0.00247	1.00777	0.00408
$[0.4, 0.5[$	1.00086	0.00055	1.00224	0.00129	1.00402	0.00223	1.00627	0.00343
$[0.5, 0.6[$	1.00083	0.00054	1.00205	0.00112	1.00363	0.00201	1.00565	0.00322
$[0.6, 0.7[$	1.00066	0.00048	1.00170	0.00096	1.00306	0.00177	1.00478	0.00294
$[0.7, 0.8[$	1.00064	0.00049	1.00161	0.00100	1.00285	0.00169	1.00442	0.00264
$[0.8, 0.9[$	1.00063	0.00055	1.00145	0.00091	1.00247	0.00157	1.00376	0.00255
$[0.9, 1.0[$	1.00064	0.00059	1.00153	0.00108	1.00260	0.00189	1.00391	0.00300

Table 4.4: Numerical values of the performance guarantee of Smith's rule for the plots in Figure 4.5. Averages have been taken over all results in the respective parameter range. The average and the standard deviation are denoted by $\varnothing$ and σ, respectively.

Chapter 5

Conclusions

One major focus of this part of the thesis was a fundamental understanding of the performance guarantee of Smith's rule for the problem $1 \,||\, \sum w_j\, f(C_j)$ under convex and concave cost functions f, and monomials as a prominent subclass in particular. All our results hold for the case when the jobs are jointly released at time zero. A natural question is whether similar techniques can be used to analyze the performance also for non-uniform release dates (and job preemption). However, already one of our first arguments—the assumption of uniform densities—breaks down when the jobs can be released at different points in time.

Concerning the complexity, we have shown that the problem $1 \,||\, \sum w_j\, f(C_j)$ is strongly NP-hard for monotone and piece-wise linear cost functions, which is the first strong NP-hardness result for this general problem. As the cost functions in the reduction are neither concave nor convex, the computational complexity of the problem for concave cost functions—and in particular (concave and convex) monomials C_j^k, $k > 0$—remains fully open while for convex functions, for which the problem is known to be weakly NP-hard [Yua92], it is still an open question whether the problem is strongly NP-hard. We believe that a proof of NP-hardness for these cases must have a fundamentally different structure than the proof given in Section 3.1, because there, the hard instances cannot only consist of jobs with density 1. Restricting to instances with uniform density, optimal schedules for convex or concave cost functions simply result from sorting the jobs by their weights.

On the positive side, Megow and Verschae [MV13] proposed a PTAS for general cost functions and an FPTAS for piece-wise linear functions in case of the number of linear segments being a constant. Together with the hardness result mentioned above, this gives a clear picture of the complexity status for this class of cost functions: For the number of linear segments being a constant, the problem allows for an FPTAS while being weakly NP-hard, and on the other hand, for an arbitrary number of of segments, the problem admits a PTAS while being strongly NP-hard. Hence, in both cases the complexity results are tight.

Note that strong NP-hardness implies the non-existence of an FPTAS only if the objective function value is integral, and of course, if $\mathrm{P} \neq \mathrm{NP}$ [GJ79]. In our case, by applying scaling arguments to the weights, processing times and piece-wise linear cost function, one can assume w.l.o.g. that the former two values and the different slopes of the function are all integral, yielding integral objective function values as well.

Addressing index algorithms, such as Smith's rule, Rothkopf and Smith [RS84] showed that there are in fact only two classes of cost functions which allow for optimal index algorithms: linear and exponential cost functions. Considering convex and

concave cost functions, we have given a thorough analysis of the performance guarantee of Smith's rule, yet, it is unclear whether there are other index algorithms which perform better for particular cost functions or which even perform better in general. For monomial cost functions $f(t) = t^k$ with $k > 0$, a natural candidate for an index function might be $\delta(p_j, w_j) = w_j/p_j^k$. However, similarly to our analysis of Smith's rule, one can observe that the corresponding index algorithm cannot achieve an approximation guarantee better than 2^k, and it is not clear if any other index algorithm can improve upon Smith's rule. So in general, given a fixed cost function, it is an open problem whether Smith's rule is already the best possible index algorithm or if there exist other index algorithms that achieve smaller approximation guarantees for that particular function.

Considering order rules, we have shown that the weight rule holds for quadratic cost functions on a global basis, and we have seen in a computational study that this simple rule helps to drastically reduce the search space in exact algorithms. In additional experiments on higher degree monomials, we observed that, at least on our test sets, the use of the global weight rule never led to suboptimal solutions. This makes us believe that it holds for general monomials of degree at least 1, or even for convex cost functions, and inversely, that monomials of positive degree smaller than 1, or even concave functions, observe the inverse variant of the weight rule. However, note that one can easily construct examples of general cost functions, neither being convex not concave, for which neither the weight rule nor its inverse variant hold, not even locally.

Part II

Integrated Sequencing and Allocation

Chapter 6

Introduction

During the industrial revolution in the late 19th century, production processes were widely reorganized from individual manufacturing to assembly line production. The high efficiency of this new concept helped to drastically increase the productivity so that nowadays, it is one of the standard manufacturing concepts in several branches of industry. While over the years, assembly lines improved and got technically more advanced, production planning at the same time became more and more complex. The awareness for the central importance of elaborate schedules, though, raised only slowly. Still, surprisingly many companies do not exploit the full potential of their production lines due to suboptimal planning. It is not unusual that even today production plans are designed manually without any support of planning or optimization tools. Of course, this does not necessarily imply poor plans. In the contrary, due to the long-term experience of the planners, these schedules are very often of impressive quality. However, if the problem becomes too complex it is basically impossible for a human being to oversee the problem as a whole, and hence, to manually produce good plans.

The addressed complex scheduling problems share major problem characteristics. One of the most prominent points is the sequencing aspect which arises naturally from the architecture of assembly lines. Due to complex side constraints, however, the overall scheduling problem usually goes far beyond basic sequencing problems known from theoretic literature. In most of the cases, the actual processing sequence is just one of many strongly interdependent decisions. Once a sequence is chosen, still certain allocations have to be made—e.g., the allocation of resources or the performance of security regulations. Such decisions may depend not only on pairs of adjacently scheduled jobs but also on whole subsequences or even the entire sequence in the worst case. As a consequence, the quest for a good processing sequence can only be answered accurately by taking into account the ensuing allocation problem as well.

Currently, for most of those problems (sufficiently fast) exact algorithms are out of reach. Computationally expensive heuristics are usually also not an option, since planning tools are utilized on a daily basis, and for this reason underlie strict runtime limits. When designing simple heuristics, on the other hand, one is often faced with problems stemming from the tangle of side constraints. In most of the cases, it can hardly be avoided that the sequencing is tailored to only some of the many side constraints, leading in the end to rather poor results.

In this part of the thesis, we investigate a generic integrated approach which aims to avoid these difficulties. Based on a black box sequence evaluation, the algorithm elaborately examines a large number of processing sequences. In this way, the highly problem specific allocation problem is separated from the problem unspecific sequencing

task. From a user's perspective, this allows to solely focus on the allocation problem which is usually far more manageable than the overall scheduling problem. This makes the approach very user-friendly, and, as we will see in examples in the following chapters, also very successful. As a nice side effect, the approach generates not only one but several, hopefully good solutions such that expert planners can take the final decision according to criteria possibly not even covered by the actual model. Since it is unlikely to be able to provide a priori performance guarantees, we assess the performance of the algorithms via instance-based lower bounds.

Before addressing related algorithmic challenges, we will first give an idea how different planning problems and the underlying allocation problems might look like.

Automotive industry. The automotive industry is one well-known example for assembly line production. Cars are usually manufactured while hanging or lying on assembly belts. Due to the constant speed of the lines, and hence, the limited time for each working step, it is not possible to install certain options like sunroofs or special navigation devices at every car in the sequence but only at a certain fraction. Violations of the constraints are measured by a penalty function, representing the cost for additional workers required or the additional time needed. This is a classic problem introduced in [PK86] already in the 1980s. It has prominently attained attention when being addressed in the 2005 ROADEF Challenge[1] in cooperation with Renault. The results of this competition are summarized in [SCNA08]. In this problem, the underlying allocation problem is trivial. Once the processing sequence is chosen, only the penalty cost needs to computed, and no further decisions have to be taken.

Food-processing industry. The food industry is characterized by a large and still increasing variety of products which is especially noticeable at the last production stage, the packaging or filling lines. Here, due to special company labels, different destinations or packet sizes, one base product again splits up into several subproducts. This makes the last stage very often the bottleneck of the whole production. As a result of the large number of products, (sequence-dependent) setup operations play a central role in production planning. Focusing only on this aspect, the problem of computing a minimum setup processing sequence can be formulated as *asymmetric traveling salesman problem* (ATSP). However, there are usually additional side constraints that make the problem much more complex than ATSP. In most of the cases, this results in non-trivial allocation problems. Classic constraints in this area stem from the limited shelf-life of the products or from hygiene regulations. An overview of typical constraints in food industry and at packaging lines, respectively, can be found in e.g., [AvD09, vDGS93]. Additional problem-specific constraints from cheese, ice cream or dairy production are discussed in [CvB93, KPG12, KPG10].

In Chapter 10, we will address a related problem from filling lines in dairy industry which is based on a cooperation with Sachsenmilch. There, to avoid the expiry of products on stock, the inventory should never exceed a certain threshold, leading on the planning side to a minimum time period to keep between the processing of two lots of the same product. On the other hand, hygiene regulations enforce regular cleaning procedures at maximum intervals. Since the cleaning procedures can replace standard setup work, this leads to a tradeoff between scheduling the cleaning procedures as late

[1] http://challenge.roadef.org/2005/en/

as possible while alternatively, using them to replace time-consuming setup work or to satisfy minimum time periods between jobs of the same type. This gives the allocation problem a packing/covering flavor with respect to different criteria.

Steel industry. The final branch of industry we want to point out is steel industry. It provides complex sequencing problems at almost every stage of production. Some of many examples where such problems occur are the strand casting process as one of the first production stages, the hot and cold rolling mills later on, and the coil coating lines at the very end; see e.g., [BT06, Cow03, TLRY01]. A classic constraint that appears in many of the related scheduling problems is the so called coffin shape. It asks for an ordering of the items so that their width slightly increases for the first few items and decreases afterwards. The reasons for demanding this is that the slabs or coils are grooving the used rollers at their borders, and hence, scheduling the items in the wrong order would lead to visible grooves on subsequent items. Consequently, if the items are not processed in a coffin shape manner, the roller needs to be replaced which delays the further production. There are similar ordering criteria with respect to other characteristics, which in total lead again to an ATSP-like subproblem (if ignoring the initial increase of the width in the coffin shape). Still, similar to the problems from food industry, there are usually further side constraints that go far beyond ATSP. A class of such constraints is due to the temperature of the processed items which plays a crucial role in many production steps. To avoid the items to cool down, it is necessary to limit the waiting times between different processing stages, and hence, it may be necessary to take into account preceding stages when planning.

In the following chapters, we will discuss a complex coil coating problem in full detail. As we will see, the non-ATSP part of the allocation problem stems from so-called shuttle-coaters. This work is the main contribution of the second part of this thesis. It is based on a cooperation with Salzgitter Flachstahl and PSI Metals.

Keeping in mind these examples, we turn towards the challenges of actually implementing our generic algorithmic approach for a concrete problem. When doing so, the most elaborate part is usually the design of algorithms for the allocation problem which—to serve as a proper subroutine—have to be sufficiently fast. Due to the large number of sequence evaluation calls, we cannot afford allocation algorithms with more than linear or quadratic runtime. In many cases, this allows again only heuristics. Their quality goes very often hand in hand with a good understanding of the problem, not only from the experimental but also from the theoretical perspective. Structural insights gained in the analysis can provide essential ideas for the design of algorithms.

Such insights may be, for instance, the problem's complexity, whether or not the problem allows exact pseudo-polynomial time algorithms, or to which extend we can expect to approximate optimal solutions. In our particular case, it is mostly easy to observe that the overall scheduling problem is NP-hard or even inapproximable. The hardness can be typically shown by straightforward reductions from variants of the ATSP if restricting to setup costs between neighboring jobs. Hence, utilizing this complexity as hardness measure, one may conclude that all problems are just similarly hard. However, this does not reflect the very different degrees of difficulty observed in practice. These differences are mainly due to the underlying allocation problem which can range from trivial to yet another NP-hard problem as we have seen above. Thus, in our context, the complexity of the allocation problem appears to be a more meaningful measure for the general hardness than the complexity of the overall problem itself. This demonstrates

that the allocation problem in fact plays a central role, not only for its importance in the proposed heuristic approach, but also as an essential characteristic of the overall problem. For this very reason, the different allocation problems will be the main target of our investigations in the subsequent chapters.

Organization of Part II. This part of the thesis addresses the general topic of integrated sequencing and allocation problems. In the remainder of this chapter, we discuss the problem at an abstract level. A high-level problem formulation is given in Section 6.1, and our generic algorithmic approach is presented in Section 6.3. In the subsequent chapters, we concretize the approach for two problems from industry, a coil coating problem from steel industry and a scheduling problem for filling lines in dairy production. They are discussed in the Chapters 7 to 9 and in Chapter 10, respectively. Concerning the coil coating problem, a detailed problem description and a graph theoretical model for the underlying allocation problem is given in Chapter 7. Thereafter in Chapter 8, we analyze an independent set problem which is closely related to our allocation problem, before finally presenting the overall algorithmic approach and the experimental results in Chapter 9. A more detailed summary of these chapters was given in the main introduction on page 4.

6.1 Abstract problem formulation

Encapsulating the structure of the above examples, we consider basic (ATSP-like) sequencing problems with integrated abstract fixed-sequence allocation problems. The latter can be seen as a scheduling problem for a fixed sequence of jobs. However, in order to distinguish it from the overall problem, which we refer to as *scheduling problem*, the term *allocation problem* is used instead. Even though our approach is not limited to this objective, the problems discussed in this thesis address the makespan objective, i.e., the maximum completion time among all jobs. At an abstract level, we can formally describe the considered scheduling problem as follows:

Integrated sequencing and allocation problem

Given: A set of n jobs $J := \{1, 2, \dots, n\}$ with processing time $p_j > 0$ for job $j \in J$ and further job characteristics. Sequence-dependent setups of length $s_{ij} \geq 0$ for any pair $i, j \in J$. Cost functions $c_\pi : \mathcal{A}_\pi \to \mathbb{R}_{\geq 0}$ for any permutation of jobs $\pi \in \Pi_n$ where $\mathcal{A}_\pi$ denotes the set of feasible allocations for π, respectively.

Task: Compute a *schedule* consisting of a *sequence* $\pi \in \Pi_n$ and an *allocation* $A(\pi) \in \mathcal{A}_\pi$ for sequence π which minimizes the *makespan*

$$C_{\max}(\pi, A(\pi)) \;=\; \sum_{j \in J} p_j \;+\; \sum_{j=1}^{n-1} s_{\pi(j)\pi(j+1)} \;+\; c_\pi(A(\pi))\,.$$

The term $\sum_{j=1}^{n-1} s_{\pi(j)\pi(j+1)} + c_\pi(A(\pi))$ is referred to as *cost* of the schedule.

Even though the allocation problem is abstract at this point, we typically assume that it involves decisions that incorporate not only neighboring jobs but also larger parts of the processing sequence π. Consequently, also the additional allocation time $c_\pi(A(\pi))$ will be non-local in this sense. In contrast, the term $\sum_{j=1}^{n-1} s_{\pi(j)\pi(j+1)}$ is purely local while $\sum_{j\in J} p_j$ is even independent of the schedule.

6.2 Related work

It is practically impossible to give a full overview of work that falls into the general setting of integrated sequencing and allocation problems. For this reason, we restrict to a short very general introduction at this point. More detailed discussions of related work are given for the considered problems from steel and dairy industry in the respective chapters.

In general, sequencing and allocation problems with setup times and makespan minimization constitute a widely studied field, see [ANCK08] for a recent survey. In most of these problems, however, setups are purely *local*, i.e., depend only on two successive jobs on the same machine and thus are closely related to variants of the ATSP, see [BSV08]. Also more general problems with setups typically involve only local setups. One such example are scheduling problems with communication delays and precedence constraints, see [BGT97], where a delay only occurs between pairs of jobs with a non-transitive precedence constraint among them when they are scheduled on different machines.

In contrast, our setup costs need to be calculated in view of several, possibly many preceeding jobs. Such *non-local* setups have only sporadically been considered in scheduling, e.g., by [KK08]. They are more typical in multi product assembly lines. But here one no longer considers sequencing problems with makespan minimization, but aims at finding batches of products that minimize the cycle time of the robotic cells, see e.g., [RDLD00].

Setups concurrent with production—as they will in occur in the coil coating problem discussed in the following chapters—also occur in parallel machine scheduling problems with a common server considered by [HPS00]: Before processing, jobs must be loaded onto a machine by a single server which requires some time (the setup time for that job) on that server. Good sequences of jobs on the parallel machines minimize the time jobs have to wait to be setup due to the single server being busy on a different machine. In this model, too, setups are purely local, and once sequences for the machines have been determined, scheduling the server constitutes a trivial task.

Considering the computational complexity of integrated sequencing and allocation problems, it is in most cases easy to observe that the problem is NP-hard or even inapproximable. Ignoring allocation aspects, the problem is equivalent to the path variant of the ATSP. Straightforward reductions from the symmetric *traveling salesman problem* (TSP) allow to transfer results from this classic problem to its path variant, implying that the problem is MAX SNP-hard even in the symmetric case when all edge length are 1 or 2 [PY93]. Moreover, for the symmetric case with general edge lengths, it follows that there exists no constant factor approximation algorithm unless P$\neq$NP [SG76]. Excellent references on TSP include the textbook by Applegate et al. [ABCC06] and the collection by Gutin and Punnen [GP02]. Despite the hardness of ATSP, Helsgaun's efficient implementation of the Lin-Kernighan Heuristic [Hel00] (LKH) can be expected

to compute optimal solutions for instances of up to roughly 100 and 1000 cities in approximately one second and one minute, respectively. For the sake of completeness, we formally define TSP and ATSP.

(Asymmetric) traveling salesman problem (TSP/ATSP)

Given: A set of n cities $\{1, 2 \ldots, n\}$ and distances $d_{ij} \geq 0$ between cities $i, j = 1, \ldots, n$ where in the standard (symmetric) case $d_{ij} = d_{ji}$ for any pair i, j while in the asymmetric case $d_{ij} \neq d_{ji}$ is allowed.

Task: Find a permutation σ of the cities minimizing the total tour length

$$\sum_{j=1}^{n-1} d_{\sigma(j)\sigma(j+1)} \; + \; d_{\sigma(n)\sigma(1)} \,.$$

In the path variant, there are additionally two assigned cities s and t given in which the tour has to start and end, respectively.

6.3 Generic algorithmic framework

We conclude this chapter by introducing our general algorithmic framework for integrated sequencing and allocation problems. For the sequencing part, we utilize a genetic algorithm which by its nature combines solutions, or *individuals*, in such a way that beneficial characteristics of solutions persist, while costly characteristics are eliminated [AL97]. The set of solutions is commonly referred to as *population*. In a *crossover*, the combination of two parents from the current population brings forth a new individual, while a *mutation* creates a new individual by modifying one already present. In an iteration, or *generation*, the population is first enlarged through crossovers and mutations, before a subset of all these individuals is selected for survival. See [MNT03] for a recent successful application to a different production sequencing problem.

During a run of our algorithm, we maintain a constant population size across generations. For each individual, we (heuristically) solve the allocation subproblem to assess its cost. Individuals with better makespans survive. The algorithm stops if a given termination criterion is fulfilled. The corresponding pseudo code is given in Algorithm 6.1, and its different components are described in more detail below.

Good parameter choices for this approach are usually the result of rigorous testing. Apart from this tuning process, the initial population and the allocation subroutine are the only components which are purely problem-specific, and hence, which need to be fully adapted.

Initial population. The initial population plays a central role in the success of our approach. Choosing an appropriate and diverse population usually increases the chance to find good solutions and moreover, to do so much faster. In contrast to the final sequence we aim to compute, for the different sequences in the initial population we rather look for sequences that are specifically good with respect to only some of the many constraints. One natural such sequence is the one we obtain when ignoring all side constraints and

Alg. 6.1: Algorithmic framework for integrated sequencing and allocation problems

repeat
  perform mutation on $m_\mathrm{m} \leq m$ randomly chosen sequences from S;
  perform crossover on $m_\mathrm{c} \leq m$ randomly chosen pairs of sequences from S;
  add sequences resulting from mutation and crossover to S;
  assess sequences in S utilizing the allocation subroutine;
  delete all but the m best sequences from S;
until *termination criterion fulfilled*;
return S;

focusing only on minimizing the total sequence-dependent setup durations s_{ij}. In this restricted form, the problem can be formulated as ATSP with distances s_{ij}, and a solution can be computed utilizing LKH. If necessary, we add further randomly generated sequences. After all, a highly diverse initial population provides a good starting point for later recombinations in the algorithm.

Termination criterion. An obvious termination criterion is the runtime limit, if given by the customer. From the algorithmic perspective, a more natural criterion is to stop after a certain number of iterations without improvement. This can avoid unnecessary long runtimes on the one hand, and on the other hand, it appears less likely to stop the algorithm right before the very last iterations which might still drastically improve the result. Depending on the problem, one might also consider to restart the algorithm if the solutions are not improving anymore.

Mutation. Mutations are conducted by inverting a random consecutive subsequence of an individual.

Crossover. For crossovers, we implement a classic mechanism for sequencing problems originally proposed by Mühlenbein et al. [MGSK88]: Upon the selection of two individuals from the current population, a *donor* d and a *receiver* r, a random consecutive subsequence of random length is taken from the donor to form the basis for the construction of the new individual, or *offspring* s. We complete s by continuing from its last element i with the elements following i in the receiver, until we encounter an element already contained in s. Now we switch back to the donor and continue completing s from d in the same way, going back to r again when encountering an element already added to the offspring. If we can continue with neither r nor d, we add to s the next element from r which is not in s yet, and try again.

Chapter 7

Integrated sequencing and allocation in coil coating

We consider a complex integrated sequencing and allocation problem from steel production. A sequence of coils of sheet metal needs to be color coated in consecutive stages. Different coil geometries and changes of colors necessitate time-consuming setup work. In most coating stages one can choose between two parallel color tanks. This can either reduce the number of setups needed or enable setups concurrent with production. Overall, the aim is to compute a minimum makespan schedule comprising the sequencing of coils and the allocation of color tanks and setup work. The current chapter provides a detailed description of this complex scheduling problem. Moreover, a graph theoretical model for concurrent setup allocations is presented. In Chapter 8, we further analyze this model and related optimization problems, before in Chapter 9, we report on our implementation of the generic algorithmic approach for integrated sequencing and allocation problems from Section 6.3 for this particular problem from steel production.

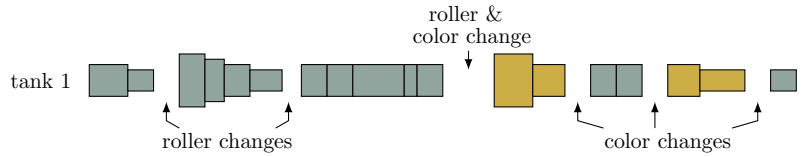
Publication remark: The results in this and the two following chapters are based on joint work with Felix G. König, Marco E. Lübbecke and Rolf H. Möhring, published in *Management Science* in 2011 [HKLM11].

In the following three chapters, we deal with a particular integrated sequencing and allocation problem as described on an abstract level in Chapter 6. The considered problem stems from the final processing step in sheet metal production, the coating of steel coils, which may be seen as a prototype of such an integrated problem: In addition to sequencing coils, a challenging *concurrent setup allocation problem* needs to be solved, and both problems strongly interdepend.

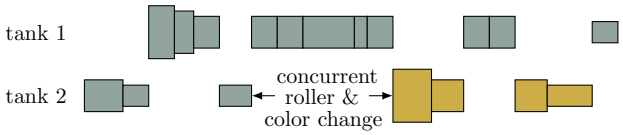
The coil coating process plays an essential role in shaping steel producers' extremely diverse product portfolio: The coils used for home appliances, for instance, already have their typical white coating when bought from the steel supplier, and the sheet metal used for car bodies already has an anti-corrosion coating before it arrives at the automotive plant for pressing. Already in the 1960s, associations were formed to promote the evolution of coil coating[1]. Progress in the development of new and improved coating materials and techniques fosters an ongoing diversification in pre-coated metal products, and in recent years, coil coating has been investigated from chemical and engineering perspective in several publications, e.g., [DBC99, DFR00, DBCF05, ZSL09].

As is typical for paint jobs, the coil coating process may be subject to long setup times, mainly for the cleaning of equipment, and thus very high setup cost. In order to

[1] http://www.coilcoating.org/, http://www.eccacoil.com/



(a) Solution for a classic coater with only one tank.



(b) Solution for a shuttle coater.

Figure 7.1: Coils of different widths as they are processed in the coating line (from left to right). Setup work (such as color or roller changes to be defined later) has to be performed whenever a coil has to be coated with a different coating or is wider than its predecessor on that tank.

reduce this cost, modern coil coating lines are equipped with so-called *shuttle coaters* or *quick (color) change coaters*, offered by different manufacturers as standard machinery; see e.g., GFG Peabody, Bronx International, or SMS Siemag[2]. Shuttle coaters possess two separate tanks allowing for holding two different coating materials at the same time. The advantage is twofold: The shuttle can be used to switch between two coatings on the same coater at almost no setup time and cost, or alternatively the unused tank can be set up while coating continues from the other tank in the meantime. We refer to this possibility to perform setup work during production—which is somewhat uncommon in scheduling literature—as *concurrent setup*; see Figure 7.1 for an illustrative example.

Literature regarding optimization in the planning process for coil coating is scarce at best: To the best of our knowledge, only Tang and Wang [TW09] consider planning for a coil coating line. They apply tabu search to a rather basic model without shuttle coaters. The present work is the first incorporating shuttles and concurrent setup work in a thorough mathematical investigation.

Our work is concerned with a typical coil coating line as operated by many major steel companies worldwide, consisting of a primer and a finisher coater, each comprising two coaters to coat the top and bottom side of a coil, see [MJ05, DBC99]. Each of the coaters may or may not be a shuttle coater, whose introduction significantly changes the flavor and the complexity of production planning: Which tank do we use for which coil, and how do we allocate concurrent setup work without exceeding available work resources? We aim to find a sequence (the order of the coils) and an allocation for that sequence (the tank assignment and allocation of setup work) that minimizes a joint objective (the makespan).

In order to precisely capture the allocation step, we introduce the *concurrent setup allocation problem*, which also fits other applications involving concurrent setup. By relating it to the independent set problem in certain generalized interval graphs called 2-*union graphs*, we obtain a dynamic program running in polynomial time for any fixed number of shuttle coaters, while proving NP-hardness when this number is part of the

[2] http://www.gfg-peabody.com/, http://www.bronxintl.com/, http://www.sms-siemag.com/en/1577.html

input. Moreover, we show corresponding results for the independent set problem in special 2-union graphs, which are of interest in their own right. These results are discussed in Chapter 8.

The graph theoretical model inspires a fast and good heuristic for the concurrent setup allocation problem, which we embed into the generic Algorithm 6.1. Altogether, we develop a practical heuristic which solves the integrated sequencing and allocation problem and computes detailed production schedules for the coil coating line. The quality of our schedules is assessed with the help of an integer program which we solve by branch-and-price. The algorithm and the results achieved are presented in Chapter 9.

Our algorithm has been added to PSI Metals' planning software suite[3], and is currently in use for a coil coating line with shuttle coaters at Salzgitter Flachstahl[4], Germany. There, it yields an average reduction in makespan by over 13 % as compared to the previous manual planning process. In addition, our lower bounds suggest that the makespan of the solutions computed by our algorithm is within 10 % of the optimal makespan for typical instances[5]. Since most setup cost calculations are incorporated into our methods as a black box, our algorithm can be adapted easily to other coil coating lines with different setup rules and a different number of shuttle coaters.

This work would not have been possible without our cooperative partners at Salzgitter Flachstahl and GESIS[6], the affiliated IT company. We very much appreciate the patience of Frank Barcikowski, Andreas Holdinghausen and Sigurd Schwarz for answering our countless questions concerning the even more countless details of setup cost, and for carefully verifying our solutions. This dialog was an integral part in making this project a textbook-like example of algorithm engineering. All technical details described in this chapter are the result of these discussions.

7.1 Problem formulation

Steel coils are a highly individualized product, and all non-productive time in coil coating depends on certain characteristics of coils. They usually have a length of 1–5 km, and their central attributes are naturally the coatings they receive in the four coating stages, chosen from a palette of several hundreds, and their width, usually 1.0–1.8 m. More intuitively, we will henceforth refer to coating materials as colors. We refrain from listing further coil attributes, since their list is very long; yet all relevant information is included in our calculations.

For a concise description of the optimization task, we shall briefly familiarize the reader with some coil coating terminology:

- A *coater* comprises all machinery necessary for applying one of the four layers of color to the coils, primer and finish on top and bottom.

- A *tank* holds a color to be used at a coater, and each tank has its own rubber *roller* for applying the color to the coil.

[3] http://www.psimetals.de/en/
[4] http://www.salzgitter-flachstahl.de/en/
[5] This success has made this contribution a finalist of the 2009 EURO Excellence in Practice Award.
[6] http://www.gesis.de/

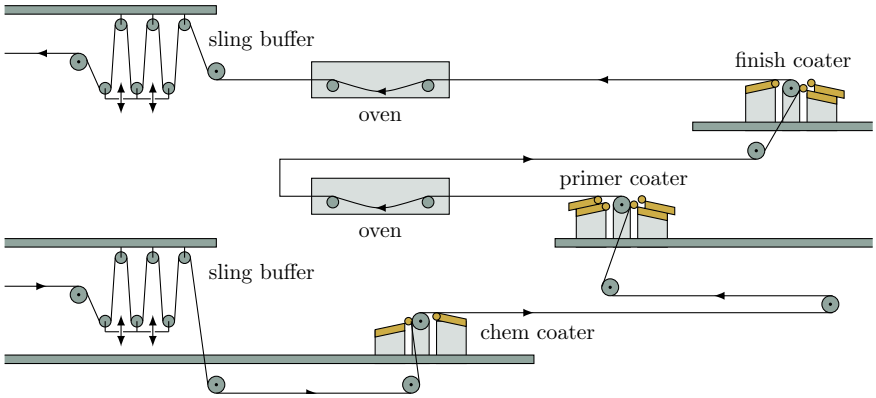


Figure 7.2: Schematic view of a coil coating line with chem, primer, and finish coater. Here, the chem and the bottom finish coaters are standard coaters, the remaining have shuttles.

- Naturally, a coater has at least one tank. A *shuttle coater* has two tanks, each with its own roller, which can be used alternatingly to apply color.

- A *color change* refers to cleaning a tank and filling it with a new color. A *roller change* needs to be performed when a coil with smaller width is preceding a coil with larger width on the same tank: Due to the wear the roller incurred at the edges of the former coil, the coating of the latter would bear imperfections.

Before entering production, each coil is unrolled and stapled to the end of its predecessor. In order to bridge non-productive time during setups, scrap coils are inserted in between actual coils, so essentially a never-ending strip of sheet metal is continuously running through the coil coating line. After undergoing some chemical conditioning of their surface, the coils run through a top and bottom primer coater, an oven, a top and bottom finish coater, and through a second oven. In the ovens, the respective coating layers are fixed. After the coating process, the coils are rolled up again, now ready for shipping. A schematic view of a typical coil coating line is depicted in Figure 7.2.

An instance of our optimization problem comprises a set $J := \{1, \dots, n\}$ of coils to be coated. Each coil j is characterized by its colors $c_j^{(k)} \in \mathbb{N}$ on coaters $k = 1, \dots, m$, its width $w_j > 0$, and its processing time $p_j > 0$, the time it takes for j to pass any given point on the coating line. Note that even though a coil passes the different stages of the coating line one after the other, it is feasible to think of a coil as being processed at all stages at once, as the time it takes a section of a coil to get from one stage to the next is negligible compared to coil lengths.

The optimization goal is to minimize the *makespan* for coating the given set of coils, i.e., the completion time of the last coil in the sequence. Concerning optimal solutions, this is equivalent to minimizing non-productive time, or *cost*, in the schedule, which ensues for two reasons described in the following paragraphs.

Transition coils. In order to satisfy certain technical restrictions, transition coils of different lengths, widths etc. may be required in between two consecutive coils in the sequence. The duration of transition coils necessary for each pair of coils $(i, j) \in J \times J$ if run consecutively is explicitly specified in an instance as s_{ij}, and we refer to it as

Figure 7.3: Optimum tank assignment for a fixed sequence of coils. The setup right before the last coil—none, in this example—depends on its predecessor on the tank, i.e., on the whole sequence.

local cost. According to the abstract definition of integrated sequencing and allocation problems on page 76, these are the sequence-dependent setup times.

Setups. Setups comprise color or roller changes. Both require the same amount of time denoted by t. If coils $i, j \in J$ are run consecutively on the same tank of a coater k, setups of total length $s_{ij}^{(k)} = s_{\text{cc}}^{(k)}(i,j) + s_{\text{rc}}^{(k)}(i,j)$ need to be performed in between, comprising the sum of durations of a possible color and roller change, where

$$s_{\text{cc}}^{(k)}(i,j) = \begin{cases} t & \text{, if } c_i^{(k)} \neq c_j^{(k)} \\ 0 & \text{, otherwise} \end{cases} \quad \text{and} \quad s_{\text{rc}}^{(k)}(i,j) = \begin{cases} t & \text{, if } w_i < w_j \\ 0 & \text{, otherwise.} \end{cases} \tag{7.1}$$

Due to the dependency of consecutive coils on a tank, it does not suffice to consider pairs of consecutive coils in the sequence in order to quantify necessary setup work. Larger sets of coils need to be taken into account—the entire sequence in the worst case, see Figure 7.3. Hence, we refer to non-productive time ensuing from setup work as *global cost*.

Finally, a *schedule* for the coil coating process consists of the following two parts:

- a *sequence* $\pi \in \Pi_n$, i.e., a permutation stating the processing order of the coils
- an *allocation* $A(\pi)$ comprising a *tank assignment* $T \in \{1,2\}^{n \cdot m}$ stating for each coil from which tank it is coated for each of the m shuttle coaters, and a *feasible setup allocation* for the set of setups $W = W(\pi, T)$ defined by π and T as described above. We elaborate on the structure of feasible allocations in the following section.

Note that there is a strong interdependence among the different parts of a solution: A tank assignment T is only meaningful in conjunction with a sequence π, and the set W of setups to be scheduled depends on T and π.

7.1.1 Concurrent setup allocation

We will now focus on the subproblem of computing an allocation for a given sequence π. Deciding on a tank assignment T settles the predecessors of coils on the tanks, yielding the set $W = W(\pi, T)$ of setups to be performed. These can either be scheduled between coils during non-productive time, or *concurrently* to production on an idle tank. While the former results in an increase in makespan, the latter does not. Consequently, there are two possibilities to save cost by good scheduling: On the one hand, the tank assignment can preserve used colors and/or rollers on an idle tank for later coils, thereby reducing the number of setups to be performed. On the other hand, setups can be performed concurrently with production on idle tanks, thereby keeping setups from increasing the makespan.

Setup allocations have to respect certain constraints in order to be feasible: While there may be several work teams to perform setups, at most one team can work on the same coater concurrently to production to ensure safety. Contrarily, during non-productive time—during which scrap coils are run, especially added for this purpose—the teams work together to finish work as fast as possible. This allows to assume one (fast) single setup resource which performs at speed $\lambda \geq 1$ where the increased speed is not only due to the joint resources but also due to the non-existing need of monitoring when no proper coil is run. As a consequence, the makespan of a solution is insensitive to deferring non-concurrent setups to the latest possible time. Hence, it suffices to only schedule concurrent setups explicitly. In contrast to scrap coils, transition coils do not allow for concurrent setup work, since transitions are critical and require very careful monitoring. Finally, the execution of one concurrent setup may not be preempted by another. However, it may be preempted to perform setups during non-productive time or to run transition coils. As a result of all these restrictions, transition coils can be ignored when scheduling concurrent setups.

A first idea of the different components of concurrent setup allocations for our coil coating problem is given in Figure 7.4 which will also lead us through the further definitions of this section. In order to capture the allocation problem more formally, we will define the *concurrent setup allocation problem*. Its definition is based on the *setup scheduling problem*.

Setup scheduling problem

Given: A sequence $\pi \in \Pi_n$ of jobs J with $n := |J|$ and where $p_j > 0$ denotes the processing time of job $j \in J$; the number m of (processing) *cells*; a *tool assignment*

$$T = \left(T_j^{(k)}\right)_{j=1,\dots,n}^{k=1,\dots,m} \in \{1,2\}^{n \cdot m}$$

telling for every job and every cell which of the cell's two interchangeable *tools* the job is run from; the set of global setups resulting from π and T

$$W(\pi,T) := \left\{ (i,j,k) \;\middle|\; \begin{array}{l} i,j = 1,\dots,n,\;\; k = 1,\dots,m,\;\; \pi(i) < \pi(j), \\ T_i^{(k)} = T_j^{(k)},\;\; T_\ell^{(k)} \neq T_i^{(k)} \;\; \forall\, \ell : \pi(i) < \pi(\ell) < \pi(j) \end{array} \right\}, \quad (7.2)$$

with durations $s_{ij}^{(k)} \geq 0$ for each $(i,j,k) \in W(\pi,T)$; the number r of *setup resources*.

Task: Determine a *concurrent setup schedule* $\mathcal{I}(\pi,T,r)$ which assigns each setup $(i,j,k) \in W(\pi,T)$ to a (potentially empty) concurrent setup interval

$$I_{ij}^{(k)} \subseteq \Big[\sum\nolimits_{\ell:\pi(\ell)\leq\pi(i)} p_\ell \;,\; \sum\nolimits_{\ell:\pi(\ell)<\pi(j)} p_\ell \Big[\quad \text{with} \quad \left| I_{ij}^{(k)} \right| \leq s_{ij}^{(k)} \,,$$

where $|I|$ denotes the length of an interval I—such that at any point in time $t \in \left[0, \sum_{j=1}^{n} p_j\right[$ at most r concurrent setups are performed, and such that the total setup volume during non-productive time is minimized. This volume is given by

$$\sum_{(i,j,k)\in W(\pi,T)} \left(s_{ij}^{(k)} \;-\; \left| I_{ij}^{(k)} \right| \right) .$$

Note that the set $W(\pi, T)$ contains exactly one setup for a pair of jobs and a cell if, according to sequence π and tool assignment T, the jobs are run consecutively on one of the cell's tools. In a feasible (concurrent) setup schedule these global setups are performed in the time interval the respective tool is idle between processing the two jobs.

Since we can assume one single λ-speed setup resource during non-productive time, exactly a $\frac{1}{\lambda}$-fraction of the total non-concurrent setup volume impacts the makespan. However, since this speed factor is irrelevant for the optimization problem at hand, we only take it into account when computing the overall cost of schedule.

Finally, the concurrent setup allocation problem extends the setup scheduling problem by the decision of the tool assignment.

Concurrent setup allocation problem (CSAP)

Given: A sequence $\pi \in \Pi_n$ of jobs J with $n := |J|$ and where $p_j > 0$ denotes the processing time of job $j \in J$; the number m of *cells*; setup durations $s_{ij}^{(k)} \geq 0$ for any pair of jobs $i, j \in J$ and any cell k which ensues if processing i and j consecutively on the same tool of coater k; the number r of *setup resources*.

Task: Determine an allocation $A(\pi)$ comprising a tool assignment $T \in \{1, 2\}^{n \cdot m}$ and a concurrent setup schedule $\mathcal{I}(\pi, T, r) = \{I_{ij}^{(k)} \mid (i, j, k) \in W(\pi, T)\}$ being a feasible solution to the setup scheduling problem, such that the total cost is minimized. The cost is given by

$$\sum_{(i,j,k) \in W(\pi,T)} \left(s_{ij}^{(k)} \; - \; \left| I_{ij}^{(k)} \right| \right) ,$$

where $W(\pi, T)$ denotes the set of global setups that result from π and T.

In our application, coaters correspond to the cells, and tanks are the tools. The number of setup resources is $r = 1$, i.e., there is one team of workers performing setups. Also, setup times have a special structure, see (7.1). Nevertheless, all of our results hold for the CSAP in general, i.e., arbitrary setup times and any r.

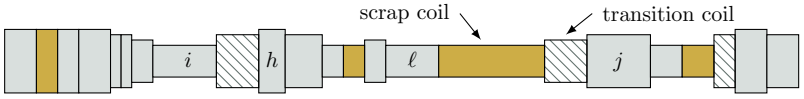
Moreover, our model and our algorithms for the CSAP remain valid even in a more general setting where each setup task can only be performed by a subset of the resources. However, for the sake of a clear presentation and cleaner notation, we omit further comments on this case.

A natural generalization of the CSAP would be to consider t tools for some general $t \in \mathbb{N}$. However, our results are strongly based on the number of tools being two.

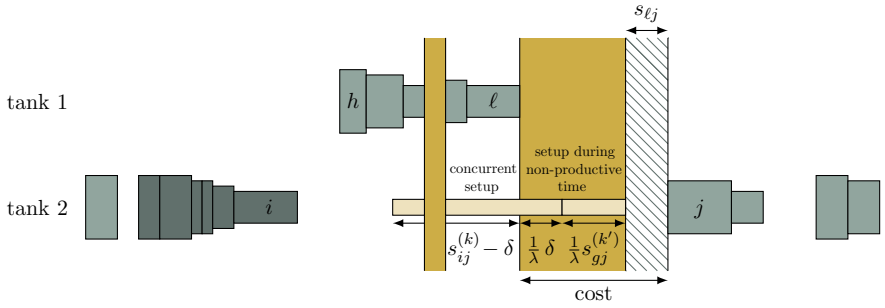
7.1.2 Cost computation

The cost of a coil coating schedule decomposes into time for transition coils and setups during non-productive time. As cost computation is complex, reconsider Figure 7.4 for an illustrative example. Again, concurrent setups incur no cost as they do not impact the makespan, and no setup may be performed during transition coils.

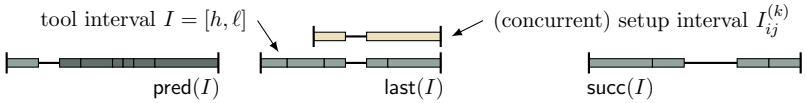
More formally, consider a sequence $\pi \in \Pi_n$, a corresponding allocation $A(\pi)$ comprising a tank assignment T and a concurrent setup schedule $\mathcal{I}(\pi, T, r)$ for $r(= 1)$ work teams,



(a) Sequence of stapled coils including scrap coils and transition coils as run through the coating line. The actual coils are shown without colors since their colors on the different coaters is usually different. The necessary lengths of the scraps coils depends on the schedule while transition coils are predefined by the sequence.



(b) Tank assignment and setup schedule at coater k, focusing on the non-productive time (cost) between coil ℓ and j. Between processing coil j and its tank predecessor i on coater k the color and roller need to be changed which requires time $s_{ij}^{(k)}$. This setup work is partially performed concurrently to production at no cost (interrupted by non-concurrent setup work), and its remaining setup volume δ is completed during non-productive time at higher speed λ. Additionally to $s_{ij}^{(k)}$, setup work for some other coater k' (not shown in the graphic) has to be performed right before j, stemming from the tank predecessor g of j on coater k'. Finally, a transition coil of length $s_{j\ell}$ needs to be run between coils ℓ and j, and no setup can be performed during this time.



(c) Corresponding tool and (concurrent) setup intervals at coater k as introduced in Section 7.2. Since transition coils and setup work during non-productive time do not allow (other) concurrent setups, it is sufficient to include only the length of proper coils in our interval model.

Figure 7.4: Exemplary schedule containing all setup and cost components.

and the set of global setups $W(\pi, T)$ stemming from π and T as defined in (7.2). Denoting by $I_{ij}^{(k)}$ the concurrent setup interval of $(i, j, k) \in W(\pi, T)$ according to $\mathcal{I}(\pi, T, r)$, the cost of schedule $(\pi, A(\pi))$ is given by

$$\sum_{j=2}^{n} s_{\pi(j-1),\pi(j)} \;+\; \sum_{(i,j,k)\in W(\pi,T)} \frac{1}{\lambda} \left(s_{ij}^{(k)} \;-\; \left| I_{ij}^{(k)} \right| \right),$$

the sum of the duration of all (local) transition coils and (global) setups performed non-concurrently. Recall that $\lambda \geq 1$ is the speed of the joint setup teams during non-productive time. Note that in order to compute the makespan, we have to add the total processing time of all coils $\sum_{j\in J} p_j$.

7.2 Interval model for concurrent setup allocation

In our solution approach, which we describe in Section 9.1, we repeatedly solve instances of CSAP resulting from different sequences of coils. Hence, an efficient algorithm for CSAP is of key importance. Since the processing sequence is a fixed input for every call of such an algorithm, we assume throughout this section that the jobs are numbered as they appear in this fixed sequence, i.e., $\pi = \mathrm{id} \in \Pi_n$.

We develop a representation of solutions as a family of weighted 2-dimensional intervals, where the first dimension is related to a tool assignment and the second to performing concurrent setup work. We call two such intervals, or axis-parallel rectangles, *independent*, if their projections onto neither of the axes intersect. As we will see, the rectangles are aligned in such a way that rectangles which correspond to tool assignments of different cells are always independent with respect to their first dimension. Similarly, every setup resource will have their own set of rectangles such that concurrent setup work performed by different resources is independent in the second dimension. However, different cells may compete for the same setup resource, and hence, the respective rectangles may conflict even though they belong to different cells. Figure 7.5 gives a first idea of this interval model which we will specify in this section in full detail.

In the end, there will be a one-to-one correspondence between solutions to CSAP on the one hand, and maximal independent sets of rectangles on the other, and a maximum weight independent set corresponds to an optimum solution of CSAP. The latter is based on rectangles' weights representing the cost savings which are achieved by the corresponding partial tool assignment and the concurrent setup work performed, compared to processing all jobs on the same tool without performing concurrent setup work.

7.2.1 Tool intervals

A tool interval—always associated with a particular cell—represents a maximal consecutive subsequence of jobs that are processed on the same tool. Thus, for each cell k, any two jobs $i, j \in J$, $i \leq j$, define a tool interval $I^{(k)} = [i, j]$ containing all jobs $i, \ldots, j$. Selecting such an interval will correspond to running all jobs in it on the same tool, and switching the tool directly before i and directly after j; see again Figure 7.4. Hence, a set of non-intersecting tool intervals, covering every job, defines a unique tool assignment for cell k, and the total processing time in interval $I^{(k)}$ is

$$p\big(I^{(k)}\big) := \sum_{\ell=i}^{j} p_\ell.$$

We call the last job before and the first job after $I^{(k)}$ its predecessor $\mathsf{pred}(I^{(k)})$ and successor $\mathsf{succ}(I^{(k)})$, respectively. Also, let $\mathsf{last}(I^{(k)})$ denote the last job in $I^{(k)}$. The weight $w^{(k)}_{\mathrm{tool}}(I^{(k)})$ of a tool interval $I^{(k)}$ comprises the reduction in cost resulting from processing all jobs in $I^{(k)}$ with one tool and $\mathsf{pred}(I^{(k)})$ and $\mathsf{succ}(I^{(k)})$ with the other, as compared to running all jobs in $I^{(k)}$ from the same tool as $\mathsf{pred}(I^{(k)})$ and $\mathsf{succ}(I^{(k)})$, and performing all necessary setup work before $\mathsf{succ}(I^{(k)})$ during non-productive time. Thus, taking into account the higher speed λ during non-productive time, we have

$$w^{(k)}_{\mathrm{tool}}\Big(I^{(k)}\Big) \;=\; \frac{1}{\lambda}\left(s^{(k)}_{\mathsf{last}(I^{(k)}),\mathsf{succ}(I^{(k)})} - s^{(k)}_{\mathsf{pred}(I^{(k)}),\mathsf{succ}(I^{(k)})}\right).$$

Note that certain tool intervals may have negative weight, i.e., selecting them actually increases cost. Due to the requirement that all of π be covered by tool intervals to obtain a well-defined tool assignment, such intervals cannot be ignored. For computational purposes however, negative weights can easily be eliminated, as we demonstrate in Section 7.2.4.

7.2.2 Setup intervals

A (concurrent) setup interval $I_s^{(k)}$ is always associated with a tool interval $I^{(k)} = [i, j]$, hence also with a particular cell k. Furthermore, it is tied to one of the r setup resources. It represents concurrent setup work performed from $\mathsf{pred}(I^{(k)})$ to $\mathsf{succ}(I^{(k)})$ on k's idle tool during jobs $i, \ldots, j$ by a certain setup resource. A setup interval's weight $w_{\text{setup}}(I_s^{(k)})$ equals its length and describes the amount of concurrent setup work performed. So, we always have

$$w_{\text{setup}}\left(I_s^{(k)}\right) \;\leq\; \min\left\{p(I^{(k)}),\; s^{(k)}_{\mathsf{pred}(I^{(k)}),\mathsf{succ}(I^{(k)})}\right\}.$$

Note that non-productive time is not included in setup intervals. Due to the assumption of one fast single resource during non-productive time, non-concurrent setup work does not need to be scheduled explicitly. Allowing only a single consecutive setup interval per tool interval ensures that concurrent setup work is never preempted by other concurrent setup work. However, it can be preempted by non-concurrent setups.

7.2.3 Saving rectangles

We now define a *saving rectangle* $R = (I^{(k)}, I_s^{(k)})$ as a combination of a tool interval $I^{(k)}$ and a (possibly empty) setup interval $I_s^{(k)}$. The total weight of a saving rectangle R is

$$w(R) \;=\; w_{\text{tool}}\left(I^{(k)}\right) \;+\; \tfrac{1}{\lambda}\, w_{\text{setup}}\left(I_s^{(k)}\right).$$

As for tool intervals, this weight may be negative.

Recall that a feasible solution to CSAP shall correspond to a maximal independent subset of all possible saving rectangles. Quite naturally, both dimensions of a saving rectangle, i.e., tool and setup intervals, are associated with a sense of time according to their position in $[0, \sum_{j=1}^{n} p_j[$. We will make use of this fact by considering different disjoint sections in the plane which are all associated with this sense of time, indicated in Figure 7.5 by the small time axis. These sections will allow us to position rectangles in the plane such that their projections onto one of the axes intersect if and only if the savings in makespan represented by their weight *conflict*, i.e., cannot be realized jointly.

Two saving rectangles conflict if and only if one of the following holds:

- They belong to the same cell and their tool intervals intersect: Tool intervals of one cell, whose intersection contains a job j, lead to conflicting tool usage, since both intervals assign j to a different tool.

- They belong to the same setup resource and their setup intervals intersect: Each resource can perform only one setup at a time.

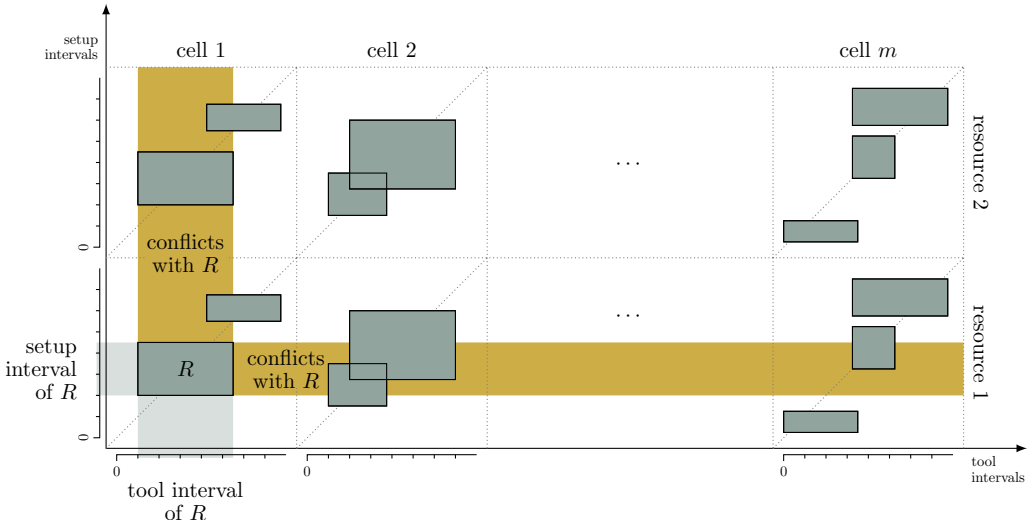


Figure 7.5: Some saving rectangles, appropriately positioned in the plane. Selecting rectangle R, i.e., running all jobs belonging to its tool interval on the same tool of cell 1 and performing concurrent setup during its setup interval, prohibits the selection of any rectangle whose projection onto one of the axes intersects that of R. The selection of non-disjoint tool intervals prevents a proper tool assignment, while intersecting setup intervals violate resource constraints. For simplicity, only two resources are visualized.

Consequently, when positioning rectangles in the plane, all rectangles associated with the same cell receive their own section of the x-axis, while all rectangles sharing the same setup resource are placed in a distinct section of the y-axis. Within sections, all rectangles are positioned in x and y direction according to their natural sense of time, see Figure 7.5 for an illustration. Note that by removing certain rectangles corresponding to one of the setup resources, one can easily model the setting where particular types of setup work can only be performed by/on appointed resources.

Now, any subset of pairwise independent saving rectangles, whose tool intervals cover all jobs on all cells, naturally corresponds to a feasible concurrent setup allocation with corresponding tool assignment. Moreover, if such a subset has maximum weight, the corresponding schedule is optimal: It realizes the greatest savings possible as compared to running all jobs on the same tool.

By the construction above, saving rectangles also have a very specific structure. We build upon these properties when analyzing the independent set problem ensuing from CSAP in Chapter 8:

Observation 7.1 (Characteristics of saving rectangles)

- *The projection of each rectangle onto the x-axis is contained completely in one of the distinct sections corresponding to the cells.*
- *If the projections onto the y-axis of two rectangles intersect, their projections onto the x-axis contain common jobs.*

The latter is due to the fact that time-wise, a rectangle's setup interval is always contained in its tool interval. Note that projections of rectangles onto the x-axis containing common jobs need not necessarily intersect when positioned as in Figure 7.5—they may well belong to different cells and thus lie in different sections of the x-axis.

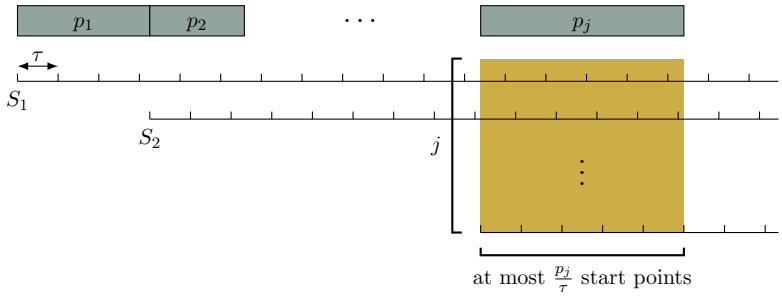


Figure 7.6: Bounding the number of potential start points of setup work. Note that in case of the processing time p_j being a multiple of τ, the indicated interval contains in fact $\frac{p_j}{\tau} + 1$ start points. However, since the last such point appears as the first potential start point corresponding to the next job, we can ignore it.

Finally, note that for general instances of CSAP, it may not be tractable to explicitly model all possibilities to perform concurrent setup work in an application, as the number of setup intervals could become super-polynomially large. In the case where all setup durations have a sufficiently large common divisor, however, we can prove a polynomial bound on the number of setup intervals that need to be considered for an optimal solution.

Lemma 7.2 *Given an instance of CSAP, let τ denote the greatest common divisor of all setup durations $s_{ij}^{(k)}$. Then there is a set D of at most*

$$N := \frac{\max_{j \in J} p_j}{\tau} \cdot \frac{n\,(n-1)}{2}$$

points in time such that there exists an optimal selection of saving rectangles $\mathcal{R}$, whose setup intervals all have start and end points in D. The set D is given by

$$\left\{ \sum_{i<j} p_i + \ell\,\tau \;\middle|\; j = 1, \ldots, n-1,\ \ell \in \mathbb{N}_0,\ \sum_{i<j} p_i + \ell\,\tau < \sum_{i \le n-1} p_i \right\}. \tag{7.3}$$

Proof. Let $S_j = \sum_{i<j} p_i$ denote the start time of job j in the processing sequence ignoring non-productive time. We will show that there is an optimal solution that schedules all concurrent setups at times of the form $S_j + \ell\,\tau$ for some $\ell \in \mathbb{N}_0$. Consider an optimal solution with a minimum number of setups not starting at times of this form. Let (i, j, k) be the first such setup. Let (i', j', k') be the setup which is performed before (i, j, k) by the same resource. W.l.o.g. we can assume there is no idle time between the two setups. If (i', j', k') is finished in non-productive time before job j', then (i, j, k) starts at time $S_{j'+1}$. Otherwise, (i', j', k') starts at time $S_{j''} + \ell'\,\tau$ for some job j'' and some $\ell' \in \mathbb{N}_0$ and finishes $s_{i'j'}^{(k')}/\tau \in \mathbb{N}_0$ multiples of τ later. Again, this is of the required form. Hence, we can assume that all setups start at times of the form $S_j + \ell\,\tau$.

It remains to bound the number start and end points of the described form. Moving through the sequence in the processing order of the jobs, the number of new potential start points during the processing of job j is bounded by $j\,p_j/\tau$, see Figure 7.6. Thus, the total number of potential start points for setup intervals necessary in an optimal solution is

$$\sum_{j=1}^{n-1} j\,\frac{p_j}{\tau} \;\le\; \frac{\max_{j \in J} p_j}{\tau} \cdot \frac{n\,(n-1)}{2} \;=\; N.$$

□

Thereby, the number of saving rectangles which need to be considered for an optimal solution is bounded by $m \cdot r \cdot n^2(N+1)$, the number of possibilities to pair each of the n^2 possible tool intervals of each cell with no or one setup interval of every setup resource. Note that N is polynomial in n, as long as the ratio $\max_{j \in J} p_j/\tau$ is bounded polynomially in n. This bound is also very coarse, since it takes into account all possibilities to pair *any* setup interval with a tool interval, while only setup intervals completely contained in it are relevant.

In our application the duration of all setups is a multiple of t, hence, Lemma 7.2 applies with $\tau = t$. In realistic instances, the maximum length of a coil is roughly twice t, so the number of start points for concurrent setup work is no more than n^2. Since setups comprise either a roller *or* a color change or a roller *and* a color change, possible setup durations are only t and $2t$. This leads to at most $2n^2$ potential setup intervals for each cell.

7.2.4 Building schedules from maximal independent sets

Given an independent set of saving rectangles $\mathcal{R}$ such that every job is covered by a tool interval on every cell, the construction of a feasible concurrent setup allocation with corresponding tool assignment is straightforward: The allocation for concurrent setup work is given explicitly by setup intervals, interrupted by non-concurrent setups and transition coils, and at the end of each tool interval, tools are switched on the corresponding cell. By construction, there is a one-to-one correspondence between the cost $c(\mathcal{R})$ of this schedule and the reduced cost according to the chosen saving rectangles $\mathcal{R}$, i.e.,

$$c(\mathcal{R}) \;=\; \sum_{i=1}^{n-1} \left(s_{i,i+1} \;+ \sum_{k=1}^{m} \tfrac{1}{\lambda}\, s_{i,i+1}^{(k)} \right) \;- \sum_{R \in \mathcal{R}} w(R),$$

where the first part is the cost for running all jobs from the same tool, depending only on $\pi = \mathrm{id}$.

Moreover, note that negative rectangle weights can be eliminated by adding a sufficiently large constant to the weight of each rectangle, weighted by its length in the tool dimension. This does not change the structure of the problem, and we force any maximum weight independent set to be maximal such that the selected rectangles have tool intervals covering all jobs.

Chapter 8

Algorithms and complexity for the concurrent setup allocation problem

In this chapter, we study the maximum weight independent set problem in a special class of 2-union graphs which we call m-composite. This problem is closely related to the concurrent setup allocation problem on m cells. For constant m, we show that the graph problem allows for a polynomial-time dynamic program which also applies to the allocation problem. On the other hand, when m is part of the problem input, we prove separately that both problems are strongly NP-hard.

Publication remark: As remarked in Chapter 7, the results in this chapter are based on [HKLM11].

In this chapter, we aim at gaining structural insights into the *concurrent setup allocation problem* (CSAP) introduced in Section 7.2. As observed there, feasible solutions to CSAP correspond to maximal subsets of non-conflicting saving rectangles. This leads to the maximum weight independent set problem in a special class of multiple-interval graphs, so called *m-composite 2-union graphs*. These graphs can be represented by a set of axis-parallel rectangles whose projections onto the axis intersect if and only if the corresponding vertices in the graph are adjacent. Hence, from the perspective of CSAP, two vertices share an edge if the associated saving rectangles conflict. As we will see, the graph parameter m corresponds to the number of cells in CSAP.

The complexity of both problems—the maximum weight independent set problem in m-composite 2-union graphs and CSAP—depends crucially on the parameter m being constant or part of the problem input. For constant m, we can design an exact dynamic program which solves the maximum weight independent set problem in polynomial time. Utilizing that the number of necessary start points for concurrent setup work is polynomial (if the greatest common divisor of all setup durations is sufficiently large), this dynamic program can be transferred to CSAP directly. If the parameter m is part of the problem input, we show by a reduction from 3-SAT that the maximum weight independent set problem in m-composite 2-union graphs is strongly NP-hard even in the unweighted case. For CSAP, we reduce from One-in-3-SAT, showing strong NP-hardness as well. The results on the maximum weight independent set problem in m-composite 2-union graphs and on the CSAP are presented in Section 8.2 and 8.3, respectively. Preceding the actual results, we will introduce the necessary notions of multiple interval graphs and discuss related work in Section 8.1.

8.1 Preliminaries on t-interval graphs and t-union graphs

All graph classes considered in this chapter belong to the class of *intersection graphs*. A graph $G = (V, E)$ with vertex set V and edge set E belongs to this class if the vertices can be represented by a family of sets such that two vertices from V share an edge in E if and only if the associated sets intersect.

Historically, before considering different variants of multiple-interval intersection graphs, research focused on the underlying families of intervals. According to Gyárfás and West [GW95], Tibor Gallai investigated combinatorial properties of interval systems already in the 1930s. In 1968, he suggested to consider what he called at that time families of *separated t-intervals*. They are a special case of the more general *t-intervals*.

Definition 8.1 (t-interval, separated t-intervals, t-rectangles) *A* t-interval *v is a union of t intervals $v^{(1)}, \dots, v^{(t)}$ in $\mathbb{R}$. Two t-intervals v and w intersect if they contain intervals $v^{(i)}$ and $w^{(j)}$, $1 \leq i, j \leq t$, such that $v^{(i)} \cap w^{(j)} \neq \emptyset$.*

If, for a family $\mathcal{F}$ of t-intervals, it is possible to identify pairwise disjoint sections $T_1, \dots, T_t \subset \mathbb{R}$ such that any $v \in \mathcal{F}$ contains exactly one interval from each section then $\mathcal{F}$ forms a family of t-separated intervals*. W.l.o.g. it can be assumed that $v^{(i)} \in T_i$ for $i = 1, \dots, t$. If interpreting a t-separated interval v as t-dimensional axis-parallel rectangle defined by interval $v^{(i)}$ in dimension $i \in \{1, \dots, t\}$, then this rectangle is called* t-rectangle*, and $v^{(i)}$ is referred to as* projection in the ith dimension*. Two t-rectangles* intersect *if the projections in any dimension do. Non-intersecting t-rectangles are called* independent*.*

Based on families of t-intervals, in 1979 Trotter and Harary [TJH79] introduced the class of so called *t-interval graphs*. Fifteen years later, the analogon for families of separated t-intervals was considered by Deo and Kumar [KD94]. Following the naming of Bar-Yehuda et al. [BYHN+06], we refer to these graphs as *t-union graphs*.

Definition 8.2 (t-interval graph, t-union graph) *An undirected graph G is called* t-interval graph *if it can be represented as an intersection graph on a family $\mathcal{F}$ of t-intervals. If $\mathcal{F}$ can be chosen as a family of separated t-intervals, then G is called* t-union graph*.*

As is common in literature, we assume that a suitable interval representation for G is given and use the notion of a vertex in G and its interval in the given representation interchangeably. The name *t-union graphs* stems from the alternative definition of this graph class as the edge-wise union of t standard interval graphs with a common node set. However, we will mainly think of t-union graphs as being represented by a family of t-rectangles. Finally note that 1-interval graphs as well as 1-union graphs coincide with standard interval graphs.

Butman et al. [BHLR10] study different classic optimization problems in general t-interval graphs. They propose approximation algorithms for the minimum vertex cover problem, the minimum dominating set problem and the maximum clique problem observing approximation guarantees of $(2-\frac{1}{t})$, t^2 and $(t^2-t+1)/2$, respectively. Moreover, they show that the maximum clique problem is NP-hard for $t \geq 3$. Due to the relation to CSAP we are mainly concerned with the maximum weight independent set problem defined below. Related work on this problem is discussed separately in Section 8.1.2.

Maximum (weight) independent set problem (MISP/MWISP)	
Given:	A graph $G = (V, E)$ containing vertices V and edges E, and weight w_v for each vertex $v \in V$.
Task:	Compute an independent set of maximum total weight, i.e., a subset $V' \subseteq V$ such that for any $v, w \in V'$ the edge (v, w) is not in E. In case of general vertex weights the problem is referred to as *maximum weight independent set problem* (MWISP) while for uniform weights it is referred to as *maximum independent set problem* (MISP).

8.1.1 The class of m-composite 2-union graphs

The interval model for CSAP from Section 7.2 allows to formulate CSAP as MWISP in a 2-union graph in a very straightforward way. The set of potential saving rectangles forms a family of 2-rectangles defining a 2-union graph. According to Section 7.2.4, a maximal independent set in this graph corresponds to a feasible tool assignment and concurrent setup schedule, and a maximum weight independent set with respect to the saving rectangles' weights is associated with a solution with maximum savings in makespan compared to a solution which never switches the tools and never performs setup work concurrently.

As summarized in Observation 7.1, saving rectangles observe a very special structure, and hence, so do the 2-union graphs which are associated with CSAP. In order to better capture CSAP, we constrain the class of 2-union graphs accordingly. This requires a certain notion of affinity among different disjoint sections of the x- and y-axes. Recall Figure 7.5 where the axes were divided into such sections in order to position the saving rectangles appropriately. In the tool assignment dimension, there was a separate section for each of the m cells while in the setup dimension, all r setup resources had their own section. Each section spanned the full time horizon of the considered instance, i.e., the total processing time of all jobs.

Definition 8.3 (L-relative interval) *Let $\mathcal{F}_L$ be a family of intervals contained in a section $L = [s_L, e_L]$ of the real line. For an interval $v = [s_v, e_v] \in \mathcal{F}_L$, we denote by*

$$v_L := [s_v - s_L, e_v - s_L]$$

the L-relative interval *of v. When referring to some canonical section of the real line, we call these intervals* section-relative.

For the definition of m-composite 2-union graphs we will utilize the notion of relative intervals only for the x-axis, i.e., in terms of saving rectangles for tool intervals. We will discuss afterwards why we do not take into account separate sections in the second dimension, i.e., for setup intervals, even though our positioning of the saving rectangles would suggest to do so.

Definition 8.4 (m-composite 2-union graph) *A 2-union graph G is* m-composite *if it can be represented by a family V of 2-rectangles such that*

1. *For any rectangle $v \in V$ its projection $v^{(1)}$ in the first dimension lies in one of m equal length disjoint intervals $L_1, \ldots, L_m$.*

2. *For any $u, v \in V$ with $u^{(1)} \in L_i$ and $v^{(1)} \in L_j$ their projections $u^{(2)}$ and $v^{(2)}$ in the second dimension satisfy the following:*

$$u^{(2)} \cap v^{(2)} \neq \emptyset \quad \Rightarrow \quad u^{(1)}_{L_i} \cap v^{(1)}_{L_j} \neq \emptyset\,. \tag{8.1}$$

Loosely speaking, when two rectangles intersect in their projections in the second dimension, then so would their projections in the first dimension, if they belonged to the same section. Note that 1-composite 2-union graphs are standard interval graphs since any intersection in the second dimension implies an intersection in the first dimension. Hence, the second dimension is redundant, and the first dimension fully defines the graph. Similarly, n-composite 2-union graphs which contain exactly one rectangle in each section of the first dimension are also standard interval graphs. Here, the rectangles are pairwise independent in the first dimension, and so, the graph is defined by the second dimension only.

Considering the rectangles in CSAP, the two properties of the definition are always satisfied due to the characteristics described in Observation 7.1 when choosing m to be number of cells in CSAP. Considering our positioning of saving rectangles, it might seem natural to demand that the rectangles lie not only in disjoint intervals with respect to the first dimension but also with respect to second dimension, representing the different setup resources. However, since our positive results hold for the more general graph class which does not ask for such distinct sections in the second dimension, and on the other hand, the negative results can be easily transferred to this setting, we investigate the graph class as defined in Definition 8.4.

Finally, we point out that the class of m-composite 2-union graphs is a generalization of the class of so called *strip graphs*.

Definition 8.5 (Strip graph) *A 2-union graph is a* strip graph *if it can be represented by a set of rectangles whose projections in the first dimension are of the form $[i, i+1[$ for $i \in \mathbb{N}$. The intervals $[i, i+1[$ are referred to as* strips*, and a strip graph utilizing at most m different strips is also referred to as m-strip graph.*

Hence, a strip graph is an m-composite 2-union graph whose corresponding rectangles all fully cover a whole section in the first dimension. Figure 8.1 summarizes the relation of all introduced classes of multiple-interval graphs.

8.1.2 Related work on the maximum weight independent set problem in multiple-interval graphs

After illustrating how to model CSAP as MWISP in m-composite 2-union graphs, we will now discuss related work on MWISP in the different classes of multiple-interval graphs; see Figure 8.1 for a summary. Considering strip graphs, the unweighted problem variant is often also referred to as *job interval selection problem*, see e.g., [Spi99, vBMNW12].

Tractable cases. In classic interval graphs, MWISP can be solved in polynomial time. The algorithm proposed by Frank [Fra75] is based on a perfect elimination scheme of the vertices in the graph, i.e., an ordering σ of the vertices such that all neighbors of a vertex v that appear in σ after v form a clique. It is known that interval graphs always allow for such a scheme, and it can be constructed in polynomial time; see e.g., the survey article by Möhring [Möh85].

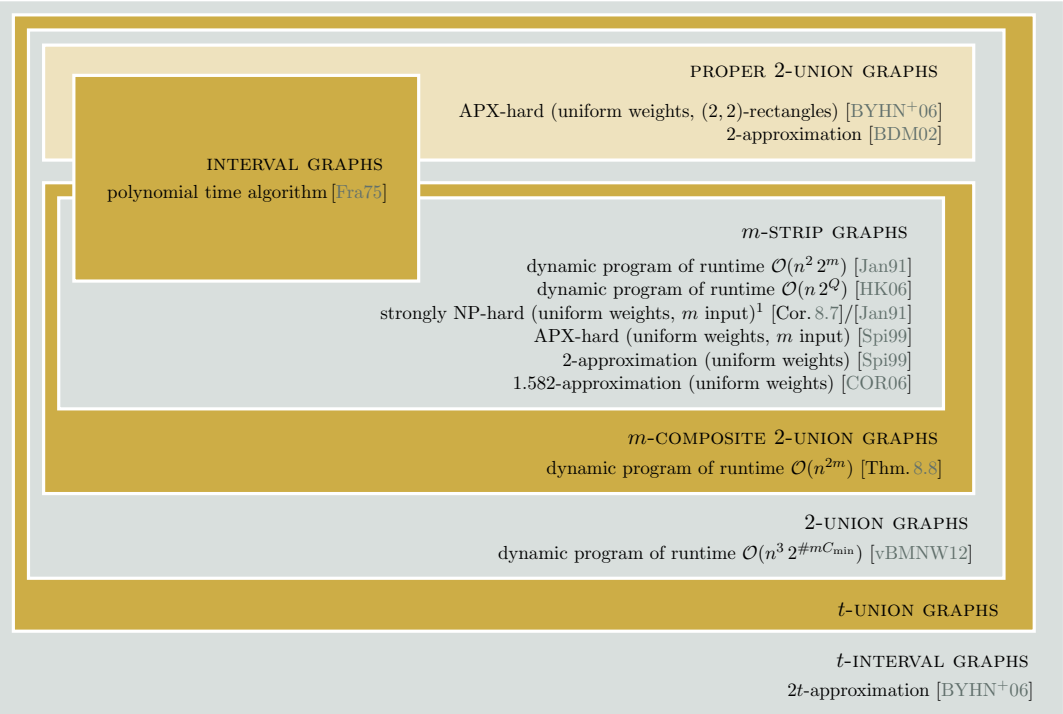


Figure 8.1: Relation of the different classes of multiple-interval graphs and the corresponding results for MWISP. The parameters Q and $\#mC_{\min}$ denote the maximum number of live strips and the minimum of the number of maximal cliques in the two interval graphs that define the 2-union graph, respectively.

Besides the above result for classic interval graphs, there are different fixed-parameter tractable dynamic programs for more general graph classes with respect to different parameters. For m-strip graphs, Jansen [Jan91] proposes one such dynamic program which runs in time $\mathcal{O}(n^2\,2^m)$. Also for strip graphs, Halldórsson and Karlsson [HK06] show that the problem is fixed-parameter tractable with respect to the maximum number Q of parallel so called *live strips*. A strip is *live* from the start of its first interval to the end of its last interval. The corresponding algorithm has a runtime of $\mathcal{O}(n\,2^Q)$. Moreover, van Bevern et al. [vBMNW12] showed recently that MWISP in 2-union graphs allows for a dynamic program with runtime $O(n^3\,2^{\#mC_{\min}})$, where $\#mC_{\min}$ is the minimum of the number of maximal cliques in the two interval graphs that define the 2-union graph.

Complexity. All hardness results we are aware of hold in the unweighted case, suggesting that the difficulty of the problem rather stems from the general structure of the graphs than from non-uniform weights. Accordingly, Jansen[1] [Jan91] showed by a reduction from 3-SAT that the unweighted MISP is strongly NP-hard on general strip graphs. This reduction is essentially identical to our proof in Theorem 8.6. Also for strip graphs, Spieksma [Spi99] showed later that MISP is also APX-hard. The proof is based on an L-reduction from the MAX SNP-hard maximum bounded 3-SAT problem.

[1]The results in this thesis were achieved independently of the work of Jansen, who sent us his paper after publication of our work. The results are available as a technical report of the Universiät Trier but they are not published elsewhere.

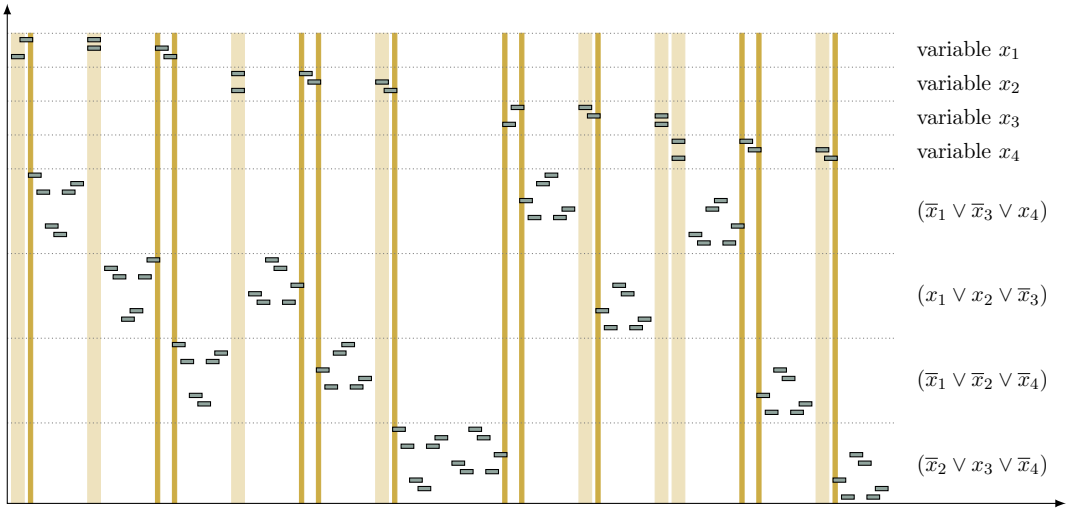


Figure 8.2: Rectangle representation of the graph G constructed from an instance of maximum bounded 3-SAT in the APX-hardness proof of Spieksma [Spi99] for MISP on strip graphs. The yellow bars indicate intersections and non-intersections.

Even though the APX-hardness is stronger than the standard NP-hardness, the graphs that are used in the reduction in [Spi99] are much more complicated, and more importantly for our purposes, their structure differs greatly from the m-composite 2-union graphs we consider. The main difference is that the graphs in the reduction of Spieksma contain several induced cycles while this is usually not the case in our setting. In Figures 8.2 and 8.3, without describing the constructions in detail, we demonstrate at an example how the graphs resulting from Spieksma's and from Jansen's/our[1] reduction look like. However, note that the rectangle representation is not very appropriate for the particular graphs of Spieksma, and hence, in his paper, he considers the actual graphs instead of this representation. Technically, the graphs of both reductions are m-composite 2-union graphs since they are strip graphs. By construction, our graphs have $m = 2\,n$ sections in the x-dimension, additionally observing the property of an $(n + r)$-strip graph with respect to the y-dimension, where n denotes the number of variables of the satisfiabilty instance, and r is the number of clauses. The graphs of Spieksma, on the other hand, allow only for being interpreted as m-composite 2-union graphs, if considering each strip in the y-dimension as a separate section. This leads to $(3\,n + 9\,r)$-composite 2-union graphs.

For the more general class of 2-union graphs, Bar-Yehuda et al. [BYHN+06] show that the unweighted problem is APX-hard when all representing rectangles are squares of side length 2 whose endpoints are integral. It is argued that every graphs whose vertices have degree 3 or less can be represented as this special type of 2-union graph, and for degree 3 graphs MISP is known to be APX-hard. Note that the described 2-union graphs are in fact *proper*, i.e., no projection of a rectangles on one of the axis is strictly contained in the projection of any other rectangle on that axis. However, m-composite 2-union graphs do not necessarily observe this property.

Approximation results. Addressing approximation algorithms, we are only aware of results for either very special or very general classes of multiple-interval graphs, such as strip graphs and proper 2-union graphs on the one hand, and t-interval graphs on the

other hand. However, there seems to be no results especially focusing on intermediate graph classes.

Addressing proper 2-union graphs, which are not our major concern, Berman [Ber00] provides a $\frac{d}{2}$-approximation at runtime $\Omega(n^2)$ for MWISP in d-claw-free graphs which implies an approximation factor of $2.5 + \varepsilon$ for proper 2-union graphs; this holds because proper 2-union graphs are 5-claw free, where a d-claw is a star with d leaves [BNR96]. Berman, DasGupta and Muthukrishnan [BDM02] propose a fast $\mathcal{O}(n \log n)$ time approximation algorithm, however, only achieving an approximation factor of 3.

For unweighted strip graphs, Spieksma [Spi99] showed that a simple greedy algorithm observes an approximation factor of 2. The factor was later improved to 1.582 by Chuzhoy, Ostrovsky and Rabani [COR06]. Their essential idea is to divide the x-axis into certain blocks and to show that one obtains near optimal solutions which contain only rectangles that lie completely in one of these blocks and where the number of rectangles per block is bounded.

In contrast to the above results for very restricted graph classes, Bar-Yehuda et al. [BYHN$^+$06] proposed a $2t$-approximation for MWISP on the general class of t-interval graphs. A standard integer linear programming formulation for MWISP in general graphs uses a constraint for each maximal clique in the graph, requiring that at most one vertex in that clique is chosen. However, the number of cliques may be super-polynomial. Bar-Yehuda et al. use a linear programming relaxation of this formulation, in which they only consider certain *interval cliques*. Fractional solutions of this relaxation are transferred to integer solutions via a rounding algorithm which is based on a factional variant of the local ratio technique.

8.2 Maximum weight independent sets in m-composite 2-union graphs

First, we show that for m being part of the problem input, MWISP is NP-hard in m-composite 2-union graphs, even in the unweighted case. The graphs that are used in the reduction are in fact strip graphs or, if you wish, *m-composite strip graphs*. Afterwards, we proceed to describe an exact dynamic programming approach running in polynomial time for any constant m. Finally in Section 8.3, we describe how both negative and positive results carry over to CSAP in modified form.

8.2.1 Hardness result for uniform weights and m being problem input

In this section, we show that the unweighted variant of the MWISP, i.e., the MISP, is strongly NP-hard in m-composite 2-union graphs. The hardness is shown by a reduction from:

3-Satisfiability (3-SAT)	
Given:	Set of n variables $x_1, \ldots, x_n$ and r clauses $c_1, \ldots, c_r$ on three literals in conjunctive normal form.
Task:	Decide whether there exists a truth assignment to the variables such that in each clause at least one literal is true.

Theorem 8.6 *For m being part of the problem input, the MISP in m-composite 2-union graphs is strongly* NP*-hard.*

Proof. We give a reduction from strongly NP-hard 3-SAT [Coo71]. Let I denote an arbitrary 3-SAT instance with n variables $x_1, \ldots, x_n$ and r clauses $c_1, \ldots, c_r$. W.l.o.g. we may assume no clause to contain two literals of the same variable. With $m := 2\,n$, we now construct an m-composite 2-union graph G, such that G admits an independent set of cardinality $n + r$, if and only if there is a truth assignment for the variables of I such that all clauses are satisfied.

For each of I's $2\,n$ literals, we introduce a vertex in the graph G, which is represented by rectangles, in such a way that rectangles belonging to literals of different variables will always be independent, while the rectangles representing two literals of one variable will intersect in the second dimension. This can be achieved by the following definition of rectangles which is also illustrated in Figure 8.3.

literal	first dimension of rectangle	second dimension of rectangle
x_i	$\big[2\,r\,(i-1)\ ,\ 2\,r\,(i-\frac{1}{2})\big[$	$[i-1;\ i[$
$\overline{x}_i$	$\big[2\,r\,(i-\frac{1}{2})\ ,\ 2\,r\,i\,\big[$	$[i-1;\ i[$

Furthermore, we introduce a vertex for each occurrence of a literal in a clause. The corresponding rectangles are positioned in a separate section in the second dimension which is reserved only for that clause. Regarding the first dimension, a rectangle associate with a literal y is positioned in such a way that it intersects the literal rectangle representing y's negation. Moreover, no two clause rectangles intersect in the first dimension. The formal definition of a rectangle in case of the occurrence of a literal in a clause is provided in the following table, see again also Figure 8.3.

literal	clause	first dimension of rectangle	second dimension of rectangle
x_i	c_j	$\big[2\,r\,(i-\frac{1}{2})+j-1\ ,\ 2\,r\,(i-\frac{1}{2})+j\big[$	$[n+j-1;\ n+j[$
$\overline{x}_i$	c_j	$[2\,r\,(i-1)+j-1\ ,\ 2\,r\,(i-1)+j[$	$[n+j-1;\ n+j[$

Now all occurrences of literals in the same clause intersect in the second dimension, while occurrences of literals in different clauses are always independent.

Now suppose the graph G admits an independent set S of cardinality at least $n + r$. Since the two literals of each variable are adjacent, S may contain at most n vertices corresponding to literals. On the other hand, vertices representing the three occurrences of literals in each clause are also pairwise adjacent, so similarly, S may contain at most one vertex per clause, and hence, in total at most r. Thus, S contains exactly one literal of each variable and one occurrence of a literal for each clause. Let us interpret the choice of literals in S as a truth assignment X for the variables of I. Each occurrence of a literal in a clause is adjacent to its negation in G, so such occurrence being in S implies it is true in X, for S is an independent set. Hence, X fulfills at least one literal in each clause of I, which is consequently a yes-instance.

Conversely, suppose I is a yes-instance and X is a truth assignment fulfilling all clauses. Then an independent set in G can be constructed in the same way, picking n

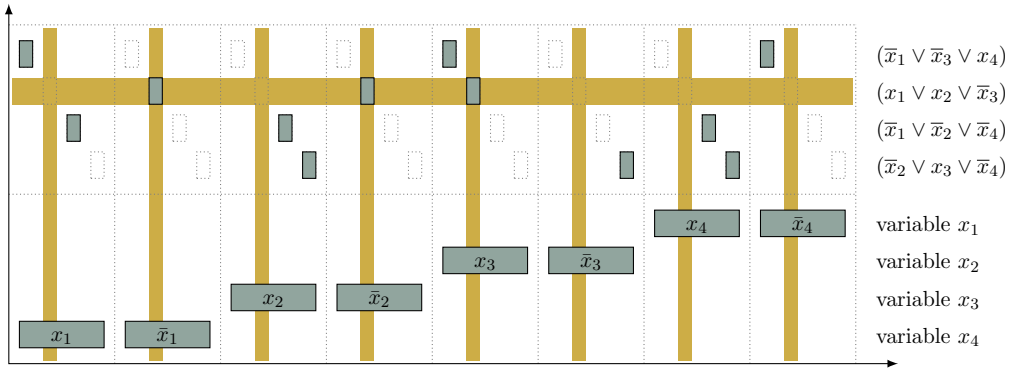


Figure 8.3: Rectangle representation of the graph G constructed from the 3-SAT instance with four variables and clauses. Every clause has its own set of potential positions of rectangles representing it (dotted rectangles). For the second clause, these positions lie in the intersections of the yellow horizontal and vertical bars.

rectangles corresponding to the literals in X and an occurrence of a literal true in X for each clause.

Finally, for $m = 2n$, the graph G is also m-composite: Firstly, no rectangle crosses an integral multiple of r in the first dimension, i.e., the end points of the rectangles corresponding to literals. So the intervals $\{[(i-1)r, ir[\mid i = 1, \ldots, 2n\}$ define m disjoint sections of the first dimension which contain all rectangles. Secondly, rectangles which intersect in the second dimension either belong to the literals of one variable or represent literals of the same clause. In both cases, the rectangles have identical section-relative intervals in the first dimension by our construction. Hence G also satisfies property (8.1). □

Note that this reduction actually demonstrates that MISP is even *NP*-hard in 2-union graphs of a very specific structure: The graph constructed is the edgewise union of a collection of pairwise disjoint 2- and 3-cliques on the one hand, and a collection of disjoint stars on the other. In particular, the graph is a strip graph.

Corollary 8.7 *The MISP on strip graphs is strongly* NP*-hard.*

It is not immediately clear, however, that hardness of this problem entails hardness of CSAP. In Section 8.3, we show that the reduction can be transformed such that the resulting interval representation is the interval model of a CSAP instance.

8.2.2 Dynamic program for constant m

Let us now turn to positive results. We propose the following dynamic programming algorithm to compute a maximum weight independent set in m-composite 2-union graphs, which runs efficiently when m is a constant.

Theorem 8.8 *For any constant m, a maximum weight independent set in m-composite 2-union graphs can be computed in polynomial time by dynamic programming.*

Proof. Let G be an m-composite 2-union graph on n vertices which is represented by a set of rectangles. We define a *state* to be an m-tuple

$$S_v = (v_1, \ldots, v_m),$$

where v_i is either a vertex whose rectangle lies in interval L_i in the first dimension according to Definition 8.4, or v_i is empty. Note that the vertices of one state are thereby independent in the first dimension, and $(n+1)^m$ is a rough upper bound on the number of states. We call a state *feasible*, if its vertices also form an independent set in the second dimension (and hence in G).

We now define a directed acyclic graph $\mathcal{S}$ on the set of feasible states, and prove that a longest path from the empty state, whose components are all empty, to some other state corresponds to a maximum weight independent set in G. Since the number of states is polynomially bounded for constant m, this will yield the result.

We call two states S_u and S_v *compatible*, if $S_u \cup S_v$ forms an independent set in G. For the ith component v_i of S_v, let $s_{L_i}(v_i) := s_{v_i} - s_{L_i}$ and $e_{L_i}(v_i) := e_{v_i} - s_{L_i}$ denote the start and end point of the L_i-relative interval of $v_i^{(1)}$, the projection of v_i in the first dimension. Here, s_{v_i} and e_{v_i} are the absolute start and end points of v_i in the first dimension, and analogously, s_{L_i} denotes the start point of L_i. Recall that the intervals L_i are assumed to have equal length. When v_i is empty, we set $s_{L_i}(v_i)$ and $e_{L_i}(v_i)$ to zero. We define the arc set of graph $\mathcal{S}$ as

$$E(\mathcal{S}) \;:=\; \big\{(S_u, S_v) \,|\, S_u, S_v \text{ compatible and } e_{L_i}(u_i) \le e_{L_j}(v_j) \;\; \forall\, i,j = 1,\dots,m\big\}.$$

So intuitively, when following a path in $\mathcal{S}$, the corresponding rectangles of each section in the first dimension are traversed in a certain monotone fashion.

Now $\mathcal{S}$ is clearly acyclic, and we argue that any path $P = (S_s, \dots, S_t)$ in $\mathcal{S}$ defines an independent set $S_s \cup \dots \cup S_t$ in G: First, the union of two subsequent states in P is independent by definition of $E(\mathcal{S})$. Furthermore, the union of all states in P is independent in the first dimension due to the demanded $e_{L_i}(u_i) \le e_{L_j}(v_j)$ in $E(\mathcal{S})$. It remains to show that the union of any two non-subsequent states in P is independent in the second dimension as well.

For the sake of contradiction, assume there is a state $S_u \in P$ and a rectangle $u_i \in S_u$ which, in the second dimension, intersects with a rectangle v_j contained in a state succeeding S_u on P. Let S_v denote the first such state. Since adjacent states form independent sets by definition of $E(\mathcal{S})$, there must be a state S_a in between S_u and S_v on P which contains neither u_i nor v_j. In particular it holds $a_j \neq v_j$ because otherwise S_v would not be the first state conflicting with S_u.

Since u_i and v_j intersect in the second dimension, property (8.1) implies that the section relatives projections in the first dimension intersect as well, i.e.,

$$s_{L_j}(v_j) \;\le\; e_{L_i}(u_i) \;\le\; e_{L_j}(v_j)\,. \tag{8.2}$$

On the other hand, from the definition of $E(\mathcal{S})$, and from the fact that $a_j \neq v_j$ we know

$$e_{L_i}(u_i) \;\le\; e_{L_j}(a_j) \;<\; s_{L_j}(v_j)\,.$$

This is a contradiction to (8.2). Consequently any path in $\mathcal{S}$ defines an independent set in G.

We define the length of an arc (S_u, S_v) as the weight gained by a path, i.e., an independent set, through its traversal:

$$\ell\big((S_u, S_v)\big) \;:=\; \sum_{x \in S_u \cup S_v} w_x \;-\; \sum_{x \in S_v} w_x.$$

Since for any independent set in G, its nodes can be ordered by the end points of their section-relative intervals in the first dimension, any maximum weight independent set also has a representation as a path in $\mathcal{S}$, concluding our argument. □

8.3 Concurrent setup allocations

In the previous section we drew a clear picture of the complexity of MWISP in m-composite 2-union graphs. When m is part of the input, the problem is strongly NP-hard, even in the unweighted case, while the problem is in P for constant m. In this section we will investigate the complexity of the closely related CSAP. Even though being closely related, neither the hardness result nor the dynamic program for MWISP allow for a direct translation to CSAP. The hardness result on the one hand, does not carry over since CSAP—as a special case of MWISP in m-composite 2-union graphs—does not match the special structure of the graphs which occur in the reduction of the proof of Theorem 8.6. On the other hand, the dynamic program from Theorem 8.8 may have a super-polynomial runtime due to a potentially super-polynomial number of saving rectangles. However, utilizing Lemma 7.2, we know at least that the dynamic program achieves a polynomial runtime for to a certain class of CSAP instances.

Corollary 8.9 *For an instance I of CSAP, let τ denote the greatest common divisor of all setup durations $s_{ij}^{(k)}$. When the number of cells m is constant, and the ratio $\max_{j \in J} p_j/\tau$ is polynomial in the size of the input I, then CSAP can be solved in polynomial time, even if the number of setup resources r is part of the input.*

Transferring the hardness result from Theorem 8.6 to CSAP is not as straightforward as in the case of the dynamic program. In order to capture CSAP, we require additional transformation steps. Since these transformations involve certain *floating rectangles* which may correspond to a super-polynomial number of normal rectangles, the reduction does not replace Theorem 8.6. Moreover, Theorem 8.6 holds for uniform weights while the reduction in this section requires non-uniform weights. As we will see, the problem is only hard when the number of setup resources is strictly smaller than the number of cells. Otherwise, when there are sufficiently many setup resources to perform setup work simultaneously on all cells, it is not hard to see that the problem is in P.

Lemma 8.10 *In the special case of $r \geq m$, CSAP can be solved in polynomial time.*

Proof. By assigning a different resource to each cell, we can assume that two saving rectangles conflict if and only if they belong to the same cell and their tool intervals intersect. Thus, an optimal solution can be computed independently for each cell, and the problem reduces to finding maximum weight subsets of pairwise disjoint intervals, i.e., maximum weight independent sets in standard interval graphs, which can be done very efficiently [GLL82]. □

The main result of this section is the following theorem.

Theorem 8.11 *For the number of cells m being part of the problem input, CSAP is strongly* NP*-hard for any fixed number $r < m$ of setup resources.*

In this hardness proof, we follow the main ideas from the proof of Theorem 8.6. Recall that in this proof, the reduction from 3-SAT was leading to an instance of the MISP in m-composite 2-union graphs with very few rectangles which does not correspond to an instance of CSAP. Hence, further investigations are necessary to show the hardness of the latter problem. Our proof is divided into four major steps: In Section 8.3.1, we consider the special structure of m-composite 2-union graphs resulting from CSAP instances. In general, these graphs contain a very high number of rectangles. In Section 8.3.2, we investige a special CSAP instance with only one cell, for which we show that it is sufficient to concentrate on a very limited set of rectangles in the corresponding 2-union graph. This graph will be of similar structure as the one in the proof of Theorem 8.6. In the main part of the proof in Section 8.3.3, we reduce One-in-3-SAT to CSAP with one resource. Finally in Section 8.3.4, we generalize our result to an arbitrary number of resources.

Throughout the proof, we speak of an interval being contained in another according to the natural sense of time, i.e., we consider the section-relative intervals in the tool dimension as well as the setup dimension.

8.3.1 Structure of m-composite 2-unions graphs representing instances of the concurrent setup allocation problem

The m-composite 2-union graphs corresponding to instances of CSAP exhibit a special structure: All possible tool intervals for the given sequence π need to be considered for each of the m cells. Moreover, for each tool interval I, we have to take into account multiple setup intervals. According to Lemma 7.2 it suffices to consider setup intervals starting at points from (7.3), and the length of a setup interval is either the full duration of the associated setup or the remaining time until the end of the underlying tool interval. Additionally to those standard setup intervals, we also need to consider the case of an empty setup interval.

Even though all our argumentation holds for the discretized setup start times described above, we assume for simplicity that setup work may start at any point in time in the related tool interval. This way, we can think of all setup intervals belonging to a tool interval I as only one rectangle which can *float* along the setup axis within the interval I. According to the number of setup resources r, we copy this rectangle representation of the m-composite 2-union graph to r different sections along the setup axis to obtain the final graph for the CSAP instance.

As described in Section 7.2.4, by adding a sufficiently large constant to each tool interval, weighted by its length, we can assume that every maximum weight independent set covers every job on every cell. For simplicity, we do not consider these additional weights in the following. Still, we assume that in every maximum weight independent set, all jobs are covered by a tool interval on every cell.

8.3.2 Zero-cost decision variant for a special instance of the concurrent setup allocation problem

We consider a zero-cost decision variant of CSAP for a special instance with odd number of jobs n and $m = r = 1$. Later, we will use our investigations on this simple instance to construct a more elaborate one in our hardness proof. We assume $\pi = id$. For any even

job j, we have processing time $p_j = 1$, and for odd j arbitrary $p_j \geq 1$. Setup costs for the single cell are specified by

$$s_{ij}^{(1)} := \begin{cases} 0 & \text{, if } j = i+1 \text{ and } i \neq 1, n-1 \\ M & \text{, if } j = i+1 \text{ and } i = 1, n-1 \\ 0 & \text{, if } j = i+2 \text{ and } i \text{ is even} \\ 1 & \text{, if } j = i+2 \text{ and } i \text{ is odd} \\ C & \text{, if } i = 1 \text{ and } j = n \\ M & \text{, otherwise} \end{cases}$$

with $C > 0$ and $M > \sum_{j \in J} p_j$. The variable M is chosen sufficiently large, such that in any zero-cost schedule no setup of length M occurs; the length of any tool interval is simply not long enough to perform this setup work concurrently.

In the following, we will determine those rectangles which are sufficient to take into account when searching for a zero–cost schedule. Consider Figure 8.4 alongside the discussion.

- Due to the size of M, we cannot chose any tool interval containing the jobs 1 and 2 or $n-1$ and n. Hence, every zero-cost solution chooses the tool intervals $[1]$ and $[n]$. The setup intervals are empty, since no setup has to be performed at the beginning and at the end.

- The tool interval $[2, n-1]$ can only be chosen if all resulting setup is performed concurrently, i.e., if its setup interval has length C.

- Consider tool intervals $[j]$ for $j = 2, 3, \ldots, n-1$. For odd j, no setup is necessary, and hence, the setup interval is empty. If j is even, a setup interval of length 1 is required. Since in this case $p_j = 1$, there is no flexibility in starting the setup interval.

- All other tool intervals require setup work of length M, and thus, they do not occur in zero-cost solutions.

Summarizing, it suffices to consider the tool interval $[2, n-1]$ and all intervals containing exactly one job, where only $[2, n-1]$ provides some flexibility for concurrent setup work; see again Figure 8.4. This leaves exactly two possible tool assignments: Either, we switch the tool after every job, or we only switch after the first job and before the last job.

Finally note that we can replace an even job j by p_j unit-size jobs $j_1, j_2, \ldots, j_{p_j}$ without changing the set of rectangles necessary to consider. We define the setup cost between consecutive replacement jobs to be 0, and transfer the setup cost to and from j to j_1 and j_{p_j}, respectively. All remaining setup cost is set to M. With the same arguments as above, it is sufficient to assume that the jobs $j_1, j_2, \ldots, j_{p_j}$ form one interval instead of previously the longer job j. The resulting number of jobs is polynomial as long as $\sum_j p_j$ is polynomial. We can thus define a 1-cell instance by specifying a set of pairwise non-adjacent jobs to take the role of the even jobs in the above construction. We call these jobs *unit rectangle jobs* (of a certain cell).

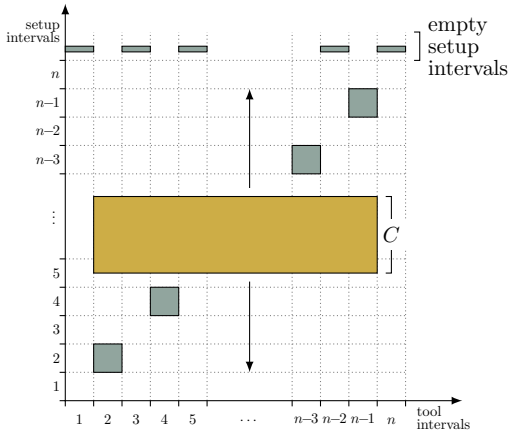


Figure 8.4: Instance of CSAP as discussed in Section 8.3.2. The graphics contains only those saving rectangles which are relevant in the zero-cost decision variant of CSAP. For simplicity, $p_j = 1$ for all j.

8.3.3 Reduction of One-in-3-SAT to the concurrent setup allocation problem with one resource

We will now show the strong NP-hardness of CSAP when he number of cells m is part of the problem input and when there is only one resource. We reduce from one-in-three 3-satisfiability.

One-in-three 3-satisfiability (One-in-3-SAT)	
Given:	Set of n variables $x_1, \ldots, x_n$ and r clauses $c_1, \ldots, c_r$ on three literals in conjunctive normal form.
Task:	Decide whether there exists a truth assignment to the variables such that in each clause exactly one literal is true.

This problem variant is known to be strongly NP-hard [Sch78]. The main ideas of the reduction are taken from the proof of Theorem 8.6.

Consider an instance I' of One-in-3-SAT with n' variables $x_1, \ldots, x_{n'}$ and m' clauses $c_1, \ldots, c_{m'}$. We reduce it to an instance I of CSAP with $n := 2\,(n' + m') + n'\,(3m' + 2n')$ unit-size jobs and $m := 2\,n'$ cells, one for each literal. We refer to the ℓth unit-size job simply by ℓ. Literals x_i and $\overline{x}_i$, $i \in \{1, \ldots, n'\}$, correspond to cell $2\,i - 1$ and $2\,i$, respectively. For each cell, we define the setup cost so as to construct an instance as described in Section 8.3.2. The unit rectangle jobs are chosen as follows: For x_i and $\overline{x}_i$, we choose job $2\,i$ on the cells $2\,i - 1$ and $2\,i$, respectively; see the lower left corners of the sections in Figure 8.5. If literal y is contained in the clause c_j, then we choose job

$$2\,n' + n'\,(3\,m' + 2\,n') + 2\,j - 1$$

on the cell corresponding to the negation of y; see the upper right corners of the sections in Figure 8.5. We refer to the described rectangles as *variable rectangles* and *clause rectangles*, respectively, and set the setup cost C to $3\,m' + 2\,n'$. We will show that there is a zero-cost schedule for I if and only if I' is a yes-instance of One-in-3-SAT.

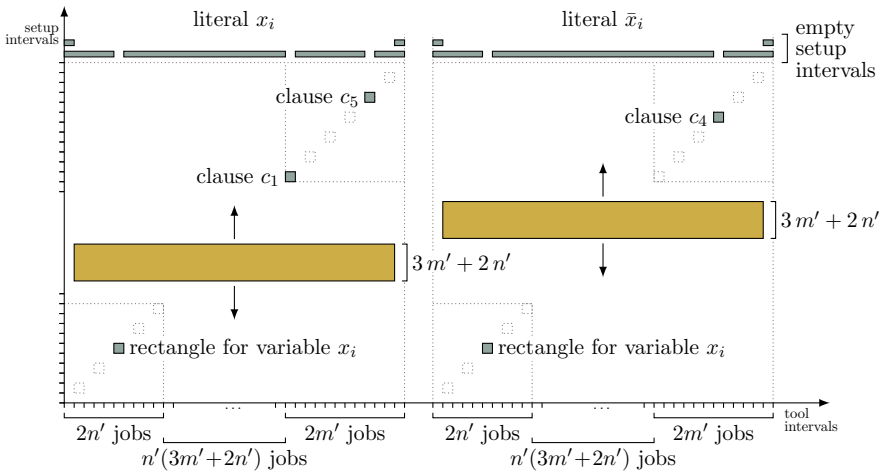


Figure 8.5: Instance of CSAP as descibed in Section 8.3.3. Cells $2i-1$ and $2i$ represent literals x_i and $\overline{x}_i$ from an instance of One-in-3-SAT. The dotted rectangles mark potential positions of variable and clause rectangles on other cells. In this example, the clauses c_1 and c_5 contain literal $\overline{x}_i$, and c_4 contains x_i.

Assume we are given a truth assignment for I'. For any true literal we choose the floating rectangle based on the tool interval $[2, n-1]$ in the corresponding cell, and arrange these n' rectangles without setup conflict in the middle section of the setup dimension which neither contains literal rectangles nor clause rectangles. Technically, for the floating rectangle of variable x_i, we choose the setup interval $[2\,n' + 1 + (i-1)\,C, 2\,n' + 1 + i\,C - 1]$, covering in total the interval ranging from $2\,n'+1$ to $2\,n' + 1 + n'\,(3\,m' + 2\,n') - 1$. On all remaining cells, i.e., on those corresponding to false literals, we choose the variable rectangle and all clause rectangles. This does not lead to a conflict, since all of these variable rectangles belong to different variables, and in every clause exactly one literal is true. For the latter, recall that a clause rectangle representing some contained literal y lies in the cell section corresponding to the negation of y. According to Section 8.3.2, the chosen rectangles represent a zero-cost schedule for I.

Consider the rectangle representation $\mathcal{R}$ of a zero-cost concurrent setup allocation of I. By the size of setup cost C, the representation $\mathcal{R}$ cannot contain more than n' floating rectangles based on the tool interval $[2, n-1]$. Otherwise, not all setup work can be performed concurrently. Assume, that there is a variable for which both such rectangles are chosen. Then, by the above, there is another variable for which no such rectangle is chosen. Since at most one of the corresponding variable rectangles can be contained in $\mathcal{R}$, this leads to schedule with positive cost. Hence, for each variable exactly one rectangle based on $[2, n-1]$ is selected. We set the corresponding literals to true. On the remaining n' cells, all variable and clause rectangles must be chosen to allow a zero-cost schedule. This is possible if and only if for each clause exactly one literal is not covered by a $[2, n-1]$-rectangle.

8.3.4 Generalization for an arbitrary number of resources

We generalize the result of Section 8.3.3 for $r > 1$ resources by adding $r-1$ cells to the above construction. For these cells, we define the setup costs as follows: Between

consecutive jobs, we have cost 0, except after the first and before the last job, where the cost is defined to be M. Between job 1 and n, we set the cost to $n-2$. In all remaining cases, the cost is set to M. The only option to schedule the jobs on such a cell at zero cost is to switch the tank after the first and before the last job and to perform concurrent setup in the meantime, blocking one full resource.

Note that with a similar construction, we can argue that the hardness result of Theorem 8.6 holds even when there is an arbitrary constant number of independent sections in the first dimension.

Chapter 9

Algorithms and experiments for the coil coating problem

In this chapter, we adapt the generic algorithmic approach from Section 6.3 for the scheduling of coil coating lines. This necessitates to design efficient heuristics for the concurrent setup allocation problem as well as to generate a diverse initial population for the genetic sequencing algorithm. The quality of our solutions is evaluated via instance-dependent lower bounds. They stem from an integer program which is based on a combinatorial relaxation of the problem, showing that our solutions are within 10 % of the optimum. Our algorithm is implemented at Salzgitter Flachstahl, a major German steel producer. This has led to an average reduction in makespan by over 13 % and has greatly exceeded the expectations of the practitioners.

Publication remark: As remarked in Chapter 7, the results in this chapter are based on [HKLM11].

In this final chapter on the coil coating problem, we present and evaluate the algorithm we developed for the everyday planning at Salzgitter Flachstahl. It follows the general concept described in Section 6.3. The algorithmic details are described in Section 9.1 in full detail. Thereafter in Section 9.2, we discuss the method we use to compute lower bounds for evaluating our solutions, before finally reporting on our computational experiments in Section 9.3.

9.1 Algorithm

As already discussed in Chapter 6, there are several challenges one is typically faced with when designing algorithms for integrated sequencing and allocation problems. Like for most of these enormously complex problems, also for the coil coating problem optimal solutions are currently out of reach. Moreover, quick computation times are indispensable for the practical usability of our algorithms: Planners at Salzgitter Flachstahl require a plan covering 24–72 hours to be computed within 180 seconds.

In order to satisfy these runtime requirements, we utilize the generic algorithmic framework described in Section 6.3 which was based on a genetic algorithm for the sequencing and a black box cost function to evaluate the generated sequences. In our case, answering the question for the cost of a sequence necessitates the computation of a concurrent setup allocation. Hence, efficient algorithms for this subproblem play a central role in the performance of the overall algorithm. Besides the allocation subroutine,

the initial population was the second component in the approach which needed to be fully adapted to the considered problem. A key to the success of our algorithm lies in constructing this population highly diverse w.r.t. different beneficial aspects as we will see in the following section. Two different heuristics for the *concurrent setup allocation problem* (CSAP) are proposed in Section 9.1.2.

9.1.1 Initial population

Recalling the problem formulation in Section 7.1, the cost of a solution is essentially given by the sum of local and global cost, where the former only depends on the sequence, while the latter is greatly affected by the concurrent setup allocation. In a sense, we would like to eventually avoid global cost by switching tanks smartly, so we focus on transition coils for the initial population.

Transition coils need to be inserted into the sequence in order to bridge differences in a certain criterion r of subsequent coils, such as their weight per meter, thickness, etc. For simplicity, we have omitted these criteria in our problem formulation. This local cost incurred between two coils i and j has the following structure:

$$s_{ij} \;:=\; \max_r\{\delta_r(i,j)\}\,,$$

where δ_r denotes the length of the transition coil necessary to bridge the difference between i and j in criterion r. The exact definitions of the $\delta_r(i,j)$ is very technical. However, it suffices to think of them as functions being non-decreasing in $|r_i - r_j|$.

When considering only one single criterion r, we can minimize the total local cost by sorting the coils according to r. On the other hand, by the definition of the s_{ij}, *one* transition coil can be used to bridge differences in *several* criteria at the same time, making the most of some unavoidable transition coil, in a sense. These two ideas are the essence of the algorithm generating our initial population.

To create an individual $\pi \in \Pi_n$, we first pick a criterion r_1 by which we sort the set of coils to be coated. To break ties, which occur frequently for some criteria, we pick a second criterion r_2. If in the ensuing sequence, call it $\pi_{1,2}$, transition coils become necessary due to some other criterion r_3, we know we could have used them to bridge differences in any other criterion at the same time, in particular differences in r_1 and r_2.

This suggests the following algorithm to build a smarter sequence π from $\pi_{1,2}$: We choose a fixed criterion r_3 different from r_1 and r_2. Then, we add coils to π in the order of $\pi_{1,2}$, skipping those which would cause local cost in π due to differences in r_3. After reaching the end of $\pi_{1,2}$, we add the first unused coil from $\pi_{1,2}$ to the end of π and repeat the process regarding only the coils in $\pi_{1,2}$ which have not been added to π yet.

We can now create different individuals for the initial population by different choices of r_1, r_2, and r_3, and each individual is likely to avoid local cost due to these criteria within certain parts of the sequence, while utilizing unavoidable transition coils to bridge differences in multiple criteria at once. Thereby, we obtain a broad variety of completely different individuals, each composed of sections which are attractive regarding their own part of the objective.

Finally, as proposed in Section 6.3, we utilize Helsgaun's efficient implementation of the Lin-Kernighan Heuristic [Hel00] to compute a sequence which is good (in most cases even optimal) w.r.t. to local cost. This sequence is added to the initial population together with a sufficient number of random sequences.

9.1.2 Heuristics for the allocation problem

Given an individual from our genetic algorithm, i.e., a fixed processing order π of coils, we now solve the resulting instance of CSAP in order to obtain a complete solution to the coil coating problem. Indeed, Corollary 8.9 yields a polynomial-time dynamic program for this subproblem in our application. However, this approach is with a runtime of $\mathcal{O}(n^6)$ obviously not sufficiently fast. Nevertheless, already the change of the perspective from CSAP to the maximum weight independent set problem in m-composite 2-union graphs—as it is done in the dynamic program—helps crucially to design fast heuristic algorithms for the allocation problem.

Independent Set Heuristic. The complexity of our exact algorithm stems from the need to consider interval selections for all coaters simultaneously in order to ensure that savings from all selected setup intervals can be realized by the scarce work resource. Intuitively, the probability that concurrent setup work on different cells can be scheduled feasibly, i.e., one setup at a time, increases with the length of the associated tool interval. This is our heuristic's core idea for computing good tank assignments.

Instead of considering all coaters at once, we consider them separately. Recall that savings from tool intervals for different coaters can be realized independently in any case. Now, instead of explicitly considering all possible saving rectangles belonging to some tool interval I, we assume that during a certain fraction α of I's length setup work can be performed, and for the moment, we do not care when exactly it is performed and even, if it can be performed that way at all. Intuitively, the parameter α is an estimate of the expected fraction of time one of the m coaters will use the work resource for its own concurrent setup work. More precisely, we define the new weight $w'_{\text{tool}}(I)$ of a tool interval I as

$$w'_{\text{tool}}(I) \;:=\; w_{\text{tool}}(I) + \min\{\alpha \cdot |I|\,,\, |I_s|\}\,,$$

where $\alpha \in [0,1]$ is a parameter of the heuristic. When choosing $\alpha = 0$, concurrent setup work is assumed impossible. For $\alpha = 1$, all potential concurrent setup work is assumed to be scheduled for each tool interval. However note that since the coil coating line at Salzgitter Flachstahl has only one team to perform concurrent setup work, the actual amount of performed setup work may well be smaller than it was assumed in the modified weight of the tool interval. With these new modified weights, it suffices to consider tool intervals alone. As a consequence, similar to the case of sufficient work resources mentioned in Lemma 8.10, computing a tank assignment T reduces to finding a maximum weight independent set in an interval graph, which can be dealt with very efficiently; see e.g., [Möh85]. In order to compute a feasible concurrent setup allocation for this tank assignment, we use an earliest-deadline-first strategy as a simple scheduling rule. As defined in (7.2), each setup $w := (i, j, k) \in W(\pi, T)$ can only be performed after completing coil i and before starting j on coater k. Based on this modeling, we define a release time r_w and a deadline d_w for w to be the completion time of i and the start time of j, respectively. Since only one concurrent setup may be performed at a time, we are trying to schedule all tasks on a single work resource. Whenever the work resource becomes available, say at time t, we schedule the setup w with the earliest deadline for which $t \in [r_w, d_w[$.

Online Rule: *First-in First-out* (FIFO). Finally, we consider the tank assignment rule which was previously in use at Salzgitter Flachstahl: Whenever subsequent coils have different colors, switch the tank. If the new tank does not contain the required color,

a color change on that tank becomes necessary. So whenever a third color besides the two in the shuttle tanks is required, the color which was in use earlier is discarded, i.e., we follow a FIFO rule. For scheduling the tasks in the resulting set $W(\pi, T)$, the same earliest-deadline-first rule as above is used. The advantage of this rule is its *online* character (and thus simplicity): the choice of tank depends only on the current and the previous coil.

9.2 Lower bounds from a combinatorial relaxation

Assessing the quality of heuristic solutions is not only a theoretical contribution, but it is an important question in practice to know how much optimization potential is left. In this section we consider three different ways of computing instance-dependent lower bounds on the optimal makespan. Two of them are rather straightforward bounds which hold regardless of the used tank assignment rule, while our more elaborate bound is limited to the use of the online FIFO tank assignment rule described in Section 9.1.2.

Ignoring the need for setups altogether we obtain the trivial lower bound as the sum of processing times of all coils, $\mathrm{LB}_{\mathrm{triv}} := \sum_{j=1}^{n} p_j$. An enhanced idea is to relax the complicating global cost only. This reduces the coil coating problem to determining an optimal sequence with respect to local cost—which can be formulated as a (small) instance of the *asymmetric traveling salesman problem* (ATSP), thus we denote the obtained bound by $\mathrm{LB}_{\mathrm{TSP}}$.

Our third bound takes into account global cost as well. In the extreme case, global cost necessary to be performed before a coil j depends on the entire solution, in particular on the entire tank assignment, prior to running coil j. Our relaxation now limits this dependency in limiting the number of coils which are considered when computing global cost, i.e., we "don't look back too far." More precisely, we concatenate subsequences containing a constant number of coils, say β, for which we exactly compute the global cost. In between subsequences we only consider local cost. By means of an integer linear program we find a cheapest such concatenation. A solution is a sequence of all coils, and the tank assignment is given implicitly by the FIFO rule. As we will discuss at the end of Section 9.2.1, one can also formulate the integer linear program in a most general way, taking into account all possible tank assignments. However, this formulation is currently computational intractable.

9.2.1 An integer program: Concatenating short subsequences

The logic of the model is to assign subsequences of β coils each to $\lceil n/\beta \rceil$ *time slots*, each consisting of β consecutive positions, except for the last slot which may possibly be shorter. Thereby we assume a subsequence to be described not only by the contained jobs and their order, but also by the time slot the sequence is scheduled in. Hence, for each subsequence of β jobs, we introduce $\lfloor n/\beta \rfloor$ separate copies, one for each time slot. Moreover, we introduce a single copy of each sequence of length $n - \lfloor n/\beta \rfloor \cdot \beta$ for the final time slot $\lceil n/\beta \rceil$. For a subsequence σ, we denote by $t(\sigma) \in \{1, \ldots, \lceil n/\beta \rceil\}$ its time slot, and by $q(\sigma, j) \in \{1, \ldots, n\}$ the absolute position of coil j in the overall sequence according to subsequence σ. We subsume all possible subsequences of β coils in the set $\mathcal{S}$. We slightly abuse the set notation $j \in \sigma$ to express that coil j is contained in subsequence σ.

Naturally, the cardinality of $\mathcal{S}$ is exponential in n, and listing $\mathcal{S}$ explicitly in an integer program is out of the question. The general idea is to solve the linear relaxation of our model by dynamically adding sequences, a.k.a. column generation, and embedding this into a branch-and-price framework [BJN+98, DL05].

We have binary variables x_{qj} for deciding whether coil j is assigned to position q or not. Variable $z_\sigma \in \{0,1\}$ indicates whether subsequence $\sigma \in \mathcal{S}$ is selected or not. Each $\sigma \in \mathcal{S}$ is charged for its local and global cost which is computed as if all coaters were entirely clean and empty at the beginning of σ. The sum of these two cost components is denoted by s_σ. In between two subsequences we consider only local cost s_{ij}. In order to model this, we have binary variables y_{ij} which correspond to coil i and j being scheduled last and first in two consecutive subsequences, respectively. Note that due to our restriction to the FIFO rule, global cost within a subsequence can be easily computed.

In total, we formulate the following integer program whose constraints are described below in more detail.

$$\min \quad \sum_{\sigma} s_\sigma \, z_\sigma \; + \; \sum_{i,j=1}^{n} s_{ij} \, y_{ij} \tag{9.1}$$

$$\sum_{q=1}^{n} x_{qj} \;=\; 1 \qquad j = 1, \ldots, n \tag{9.2}$$

$$\sum_{j \in 1}^{n} x_{qj} \;=\; 1 \qquad q = 1, \ldots, n \tag{9.3}$$

$$\sum_{\substack{\sigma \ni j: \\ q(\sigma,j)=q}} z_\sigma \;=\; x_{qj} \qquad q, j = 1, \ldots, n \tag{9.4}$$

$$x_{\mathrm{last}(t),i} + x_{\mathrm{first}(t+1),j} \;\leq\; 1 \,+\, y_{ij} \qquad \begin{array}{l} i, j = 1, \ldots, n, \\ t = 1, \ldots, \lceil \frac{n}{\beta} \rceil - 1 \end{array} \tag{9.5}$$

$$x_{qj} \;\in\; \{0,1\} \qquad q, j = 1, \ldots, n \tag{9.6}$$

$$y_{ij} \;\in\; \{0,1\} \qquad i, j = 1, \ldots, n \tag{9.7}$$

$$z_\sigma \;\in\; \{0,1\} \qquad \sigma \in \mathcal{S} \tag{9.8}$$

The constraints are interpreted as follows. Each coil gets coated exactly once (9.2), each position is filled exactly once (9.3), and a subsequence σ needs to have coil j in position q if and only if the corresponding x_{qj} indicates so (9.4). The precedence constraints (9.5) enforce that in between two consecutive subsequences in time slots t and $t+1$ we incur local cost between coil i in the last position $\mathrm{last}(t)$ in t and coil j in the first position $\mathrm{first}(t+1)$ in $t+1$. The binary variable y_{ij} precisely takes care of that, as its use is penalized in the objective function (9.1) accordingly. The optimal objective value of the integer program is denoted $\mathrm{LB}_{\mathrm{IP}}$.

In order to generalize this lower bound for arbitrary tank assignment rules, one might introduce variables $x_{qjc} \in \{0,1\}$ which additionally decide which tank to use on each coater c in each position q of the sequence. Unfortunately, the resulting model is computationally intractable with today's hardware and optimization techniques.

9.2.2 Pricing

Initially, the integer program (9.1)–(9.8) contains all x_{qj} and y_{ij} variables, but only some z_σ variables. We start with subsequences σ which are derived from a solution produced by our sequencing algorithm, restricted to use the FIFO tank assignment rule. All other z_σ variables are generated only as needed. The coupling constraints (9.4) are the only ones which contain these variables; as a consequence the corresponding dual variables $\mu_{qj} \in \mathbb{R}$ are the only relevant ones for calculating their reduced cost, which must be negative in order to profitably add the variable to the problem. The reduced cost of z_σ is

$$\bar{s}_\sigma \;=\; s_\sigma - \sum_{j \in \sigma} \mu_{q(\sigma,j),j}\,,$$

and the pricing problem is to find a subsequence of minimum, or at least negative, reduced cost. To the best of our understanding, there is no principal alternative to a brute force method since we are able to evaluate the total cost of a subsequence only when we see it in its entirety, thus we have to construct it. This also rules out most of the traditionally used dominance criteria in dynamic programming. Therefore, a straight forward depth first search enumerates the $O(n^\beta)$ subsequences to solve the pricing problem. A pruning criterion based on the dual variable values of coils not yet added to a subsequence helps in mildly reducing the search space without compromising optimality.

9.2.3 Branching

When we determine an integer solution for the x_{qj} variables, all other variables y_{ij} automatically assume integer values as well. That is, a natural candidate for taking branching decisions in the branch-and-bound tree is to branch on

$$\sum_{\sigma \ni j : q(\sigma,j)=q} z_\sigma \notin \{0,1\}$$

for a given coil j at position q. In fact, speaking in terms of branch-and-price methodology, this is branching on the so-called *original variables* x_{qj} of the problem which are explicitly present in this extensive column generation formulation as well. The branching itself can be realized as a modification to the pricing problem instead of adding an explicit branching constraint to the master problem: One simply eliminates the forbidden coil j from position q on one branch, or enforces it by eliminating all other $\{1,\ldots,n\}\setminus\{j\}$ coils from position q on the other branch. Note that the master problem has to be updated according to branching decisions: Variables corresponding to subsequences which do not respect the branching constraint have to be eliminated by setting the corresponding upper bounds to zero.

9.3 Computational study

Since the project was initiated with the explicit goal to develop a tool to go live in an existing production environment, we were provided with realistic data right from the beginning. Planners at Salzgitter Flachstahl repeatedly validated our solutions, and

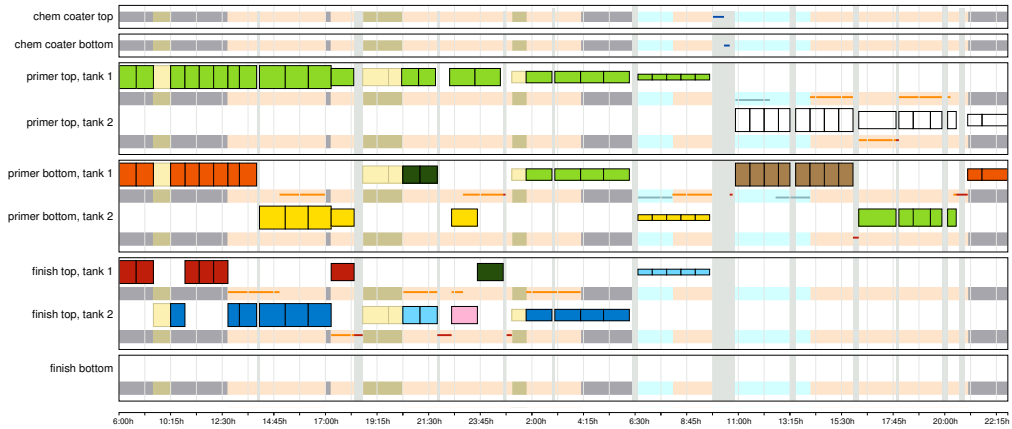


Figure 9.1: Visualization of our solutions as provided for the planners at Salzgitter Flachstahl. Each tank is represented by two lines, the top line shows the actual coils in their color and width together with transition coils while the bottom line is reserved for setup work. When transition coils are run or concurrent setup work is performed on one of the tanks, setup work on the other tanks is blocked. Scrap coils are shown as vertical bars through the whole schedule. For the chem coaters only the setup line is presented since all coils receive the same initial chemical treatment. Note that not all coils require a finish coating. In this case the coils do not appear in the schedule of that coater. In particular, no coil needs to be coated on the finish bottom coater in this example.

addressed various issues regarding the accuracy of cost calculations and the practicability of work schedules, which were eventually solved. This procedure was greatly facilitated by our visualization of the schedules as shown in Figure 9.1.

The test instances fall in two categories. The first comprises sets of up to 120 coils for a planning horizon of up to 72 hours. These instances are solved well ahead of time for long-term planning. The second category is made up of 20–40 coils for a shorter time horizon of at most 24 hours. These instances usually occur when unplanned changes in demand or availability of coils necessitate short-term replanning.

For many test instances, Salzgitter Flachstahl provided a plan devised by their expert planners for comparison with our solutions. In most cases we were able to obtain unexpected improvements over these plans. We cannot report on the precise numerical characteristics of our data and results due to non-disclosure agreements, hence normalized numbers are presented.

9.3.1 Some implementation details

Parameters for the genetic sequencing algorithm were determined by rigorous testing, with the goal to have one fixed parameter set yielding good performance across all data sets. For most short-term instances, our algorithm finds its best solutions after less than 30 seconds, while solutions keep improving towards the runtime limit of 180 seconds for long-term planning.

In the construction heuristic for the initial population described in Section 9.1.1, we use some coil properties which were omitted in the problem formulation in Section 7.1 for simplicity. All possible three-tuples of width, height, processing speed, and temperature in primer and finish oven are taken as criteria r_1, r_2, and r_3, accounting for about one third of our initial population of 200 individuals. In each generation we create 100

new individuals each from both mutations of random individuals and crossovers of two randomly chosen parents. This results in a total population of 400, of which we keep the 200 with the best makespan for the next generation.

Especially on small instances, the population quickly evolves to a set of individuals with almost identical cost. When the makespan of all individuals is within 5 % of the best individual, we discard the worst 90 % of the population and replace it with a new initial population. This usually leads to further improvement of the best individuals in subsequent generations.

Regarding allocations, we performed extensive testing using the heuristics described in Section 9.1.2. We report on results for the Independent Set Heuristic for different values of the parameter $\alpha \in [0, 1]$ in comparison to the simpler FIFO rule.

For lower bound computations, we implemented a branch-and-price algorithm to solve the integer program (9.1)–(9.8) within the publicly available SCIP framework [Ach07]. Its implementation is not tuned to performance as we used it as a proof-of-concept only, so we do not report computation times for lower bounds, which were huge.

9.3.2 Interpretation of results

Our algorithm produces plans with makespan reductions over 13 % on average, and in the best cases of up to 25 %, compared to reference solutions that had been used in production in both long-term and short-term planning, see Figures 9.2 and 9.4 and the corresponding data in Table 9.1 and 9.2. Expert planners also assert that our solutions "look very different" from theirs while retaining operability. Together with the fact that significant savings are also realized over manual plans which "looked optimal", we take this as evidence that our rigorous mathematical analysis of the savings potential of shuttle coaters fully paid off. It should also be noted again that indeed every detail of production is controlled by our plan, and that these plans were verified on-site at Salzgitter Flachstahl's coil coating line. Hence, cost savings are now realized to their full extent in day-to-day production.

Long-term instances. As expected, our Independent Set Heuristic for setup allocation proves superior to the simpler FIFO online rule. We were unable, however, to find a uniform choice of the parameter α suitable for all instances alike. When determining the best setting for α by trying all values in $\{0.1, 0.2, \ldots, 0.8\}$, the Independent Set Heuristic outperformed FIFO on 12 of the 16 instances, reducing cost by up to 30 % (over FIFO). This translates to makespan savings of up to 6 %, see Figure 9.2. When fixing α to 0.5, the Independent Set Heuristic remains similarly superior to FIFO on 8 instances, while incurring an increase in makespan of at most 1 % in four cases, see Figure 9.3.

Short-term instances. For all short term instances, we succeeded in computing lower bounds by our branch-and-price approach, proving our solutions to be within at most 10 % of makespan optimality, see Figure 9.4. Yet, we did not solve all instances to integer optimality and also used short subsequences only ($\beta \leq 6$), so the lower bound is certainly improvable.

If we concentrate on cost—in contrast to makespan—it can be seen from Figure 9.5 that the integer programming lower bound is able to close much more of the gap to the upper bound than the TSP bound which is optimal w.r.t. local cost only.

Finally, the superiority of the Independent Set Heuristic to FIFO is less significant in short-term planning. While both heuristics were on par for most instances, small improvements over FIFO were observed in three cases.

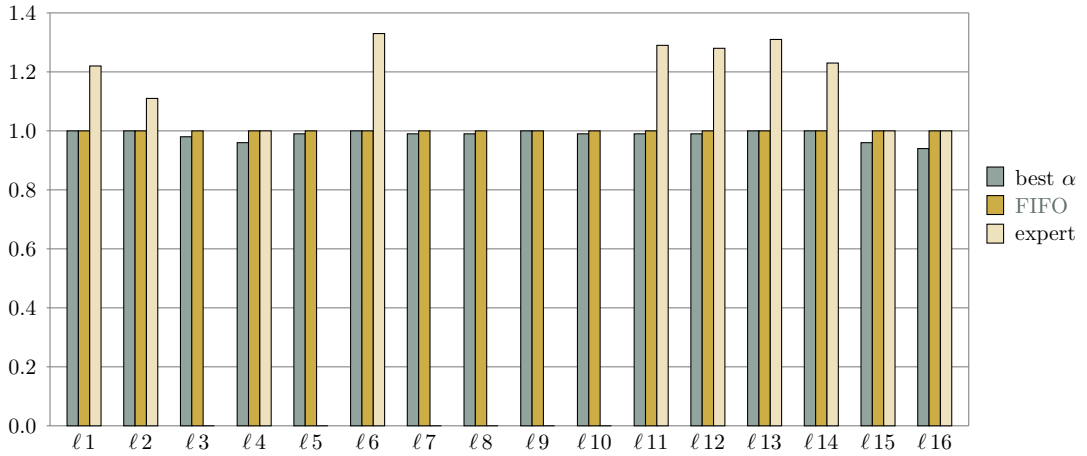


Figure 9.2: Comparison of normalized makespans for representative long-term instances. From left to right, we show the results of our algorithm using the Independent Set Heuristic with best choice of parameter α, of our algorithm using the FIFO online rule, and of the reference solutions devised by expert human planners if available.

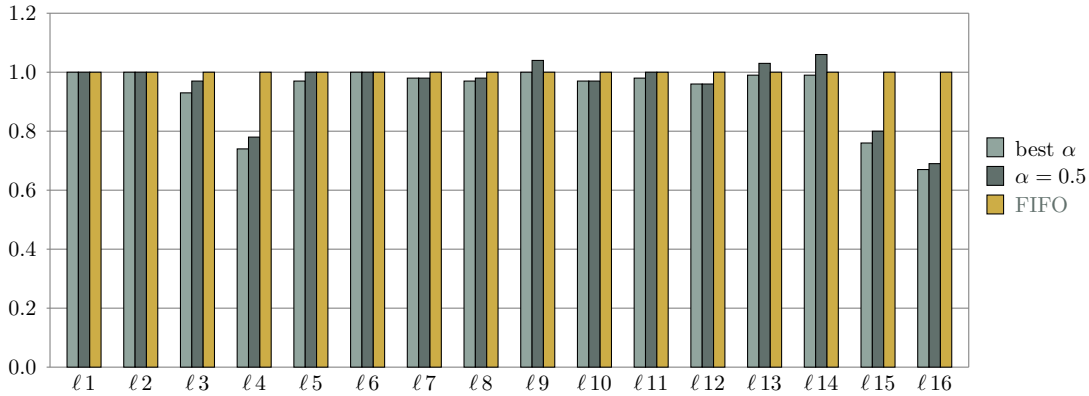


Figure 9.3: Comparison of normalized non-productive time (cost) included in our long-term solutions. From left to right, we show the results of our algorithm using the Independent Set Heuristic with best choice of parameter α, using the same heuristic with fixed choice of $\alpha = 0.5$, and using the FIFO online rule.

	makespan					cost		
instance	LB_{triv}	LB_{TSP}	best α	FIFO	expert	best α	$\alpha = 0.5$	FIFO
$\ell 1$	0.71	0.79	1.00	1.00	1.22	1.00	1.00	1.00
$\ell 2$	0.69	0.78	1.00	1.00	1.11	1.00	1.00	1.00
$\ell 3$	0.71	0.76	0.98	1.00	–	0.93	0.97	1.00
$\ell 4$	0.84	0.89	0.96	1.00	1.00	0.74	0.78	1.00
$\ell 5$	0.73	0.78	0.99	1.00	–	0.97	1.00	1.00
$\ell 6$	0.82	0.88	1.00	1.00	1.33	1.00	1.00	1.00
$\ell 7$	0.58	0.79	0.99	1.00	–	0.98	0.98	1.00
$\ell 8$	0.71	0.77	0.99	1.00	–	0.97	0.98	1.00
$\ell 9$	0.82	0.86	1.00	1.00	–	1.00	1.04	1.00
$\ell 10$	0.73	0.78	0.99	1.00	–	0.97	0.97	1.00
$\ell 11$	0.71	0.76	0.99	1.00	1.29	0.98	1.00	1.00
$\ell 12$	0.75	0.81	0.99	1.00	1.28	0.96	0.96	1.00
$\ell 13$	0.78	0.81	1.00	1.00	1.31	0.99	1.03	1.00
$\ell 14$	0.78	0.82	1.00	1.00	1.23	0.99	1.06	1.00
$\ell 15$	0.83	0.85	0.96	1.00	1.00	0.76	0.80	1.00
$\ell 16$	0.82	0.85	0.94	1.00	1.00	0.67	0.69	1.00

Table 9.1: Normalized results for long-term instances as shown in Figures 9.2 and 9.3.

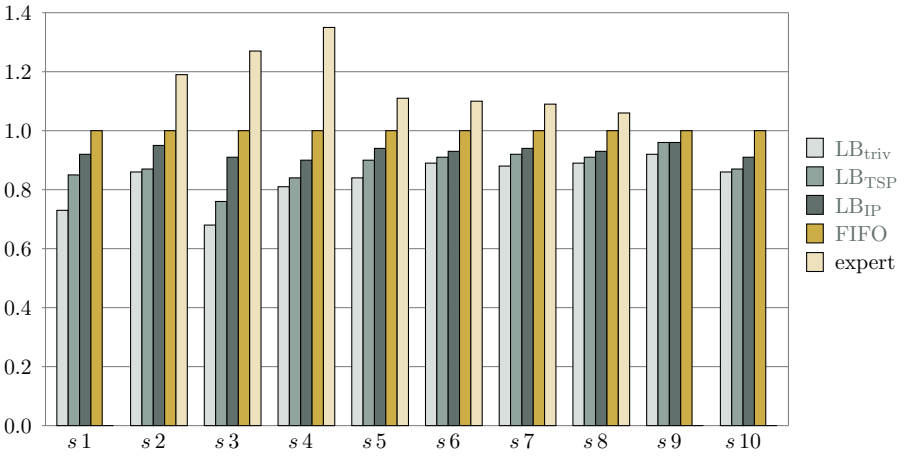


Figure 9.4: Comparison of normalized bounds and makespans for representative short-term instances when restricted to FIFO tank assignment. Bounds displayed from left to right are LB_{triv}, the sum of coil processing times, LB_{TSP}, the sum of processing times plus local cost in an optimal, local cost based ATSP solution, and our IP bound LB_{IP}. Makespans are given for our solutions obtained using the FIFO online rule for scheduling, as well as for a reference solution devised by expert human planners if available.

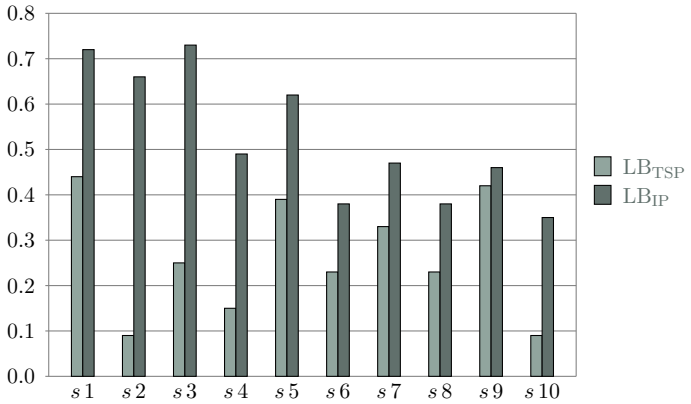


Figure 9.5: Percentage of the closed relative gap between the respective trivial lower bound and the non-productive time included in our short-term FIFO solutions: Our bound LB_{IP} from the branch-and-price approach respects global cost and closes considerably more of the gap than the ATSP based bound LB_{TSP} which only considers local cost.

	makespan					closed gap	
instance	LB_{triv}	LB_{TSP}	LB_{IP}	FIFO	expert	LB_{TSP}	LB_{IP}
$s\,1$	0.73	0.85	0.92	1.00	–	0.44	0.72
$s\,2$	0.86	0.87	0.95	1.00	1.19	0.09	0.66
$s\,3$	0.68	0.76	0.91	1.00	1.27	0.25	0.73
$s\,4$	0.81	0.84	0.90	1.00	1.35	0.15	0.49
$s\,5$	0.84	0.90	0.94	1.00	1.11	0.39	0.62
$s\,6$	0.89	0.91	0.93	1.00	1.10	0.23	0.38
$s\,7$	0.88	0.92	0.94	1.00	1.09	0.33	0.47
$s\,8$	0.89	0.91	0.93	1.00	1.06	0.23	0.38
$s\,9$	0.92	0.96	0.96	1.00	–	0.42	0.46
$s\,10$	0.86	0.87	0.91	1.00	–	0.09	0.35

Table 9.2: Normalized results for short-term instances as shown in Figure 9.4 and 9.5.

Chapter 10

Integrated sequencing and allocation for filling lines in dairy production

In this chapter, we consider an integrated sequencing and allocation problem which arises at filling lines in dairy industry. The underlying allocation problem involves certain packing and covering aspects which not only occur in dairy production but also in other branches of industry. As for the coil coating problem, we concretize our generic approach for integrated sequencing and allocation problems. Based on insights into structural properties of the problem, we propose different allocation subroutines. In cooperation with Sachsenmilch, the algorithm was implemented for their bottleneck filling line, and evaluated in an extensive computational study. For the original data from production, our algorithm computes almost optimal solutions. As a surprising result, our simple greedy algorithms outperform the more elaborate ones in many aspects.

Publication remark: The results in this chapter are based on joint work with Torsten Gellert and Rolf H. Möhring which was presented at the *10th International Symposium on Experimental Algorithms* (SEA 2011) and which is published in *Optimization Letters* in 2011 [GHM11].

In this last chapter, we address a planning problem of filling lines in dairy industry. After the coil coating problem, this is the second integrated sequencing and allocation problem for which we will implement the abstract algorithmic approach described in Section 6.3. The results are not only of interest for providing a practicable planning tool for practice but also as a proof of concept for our generic approach for integrated sequencing and allocation problems.

Dairy production involves various complex optimization problems. Several production planning problems arise from the different processing stages like homogenization or fermentation, and logistical problems need to be solved when delivering the finished products to the customers. However, the bottleneck of the entire process are the filling lines, having a low packaging rate compared to the flow rates of the previous stages [KPG10]. In recent years, this bottleneck role has become more and more crucial due to growing product portfolios, decreasing lot sizes and the resulting increase in the overall setup time. As a consequence, the scheduling of filling lines is one of the most prioritized tasks in the whole planning of dairy factories.

The corresponding optimization problem incorporates sequencing as well as allocation components: For a given set of dairy products—e.g., cream or yogurt—one aims for a filling order with minimum makespan. However, the makespan is not only determined by this order but also by setup tasks and waiting periods which may become

necessary for the chosen sequence. Several options of performing setup work and inserting waiting periods lead to an allocation problem for the fixed filling sequence. Most basic, sequence-dependent setups are necessary in between consecutive products in the sequence. Additionally, to meet hygiene regulations, the filling machinery has to be cleaned and sterilized at certain intervals to which we refer to as *cleanings*. Even though some of the sequence-dependent setups may already involve cleanings, further cleanings are usually required to satisfy the regulation. There are two ways of allocating additional cleanings: By job preemption, they can be inserted at any point in the schedule, or, alternatively, they can replace less exhaustive sequence-dependent setups. This yields a tradeoff between keeping the total number of cleanings small by postponing them as long as possible, and saving additional setup time by replacing sequence-dependent setups, but possibly requiring more additional cleanings in total; see Figure 10.1. Thus, setups are not only determined by their neighboring jobs, but also by scheduling decisions taken in view of the entire sequence.

In addition, there are waiting periods that result from filling constraints of certain products, or from different technical constraints. A typical example for the latter is the production of cream. The pretreated milk is processed in a special tank before the finished cream can be filled into cups. Due to the limited tank size, cream jobs have to be partitioned into different tank fillings. Resulting from the preparation time in the tank and the later cleaning time, the filling machine may be idle, awaiting the first cream job of a new tank filling.

Similar aspects do not only occur in dairy industry but also at packaging lines or in car sequencing. Such constraints often require to partition certain tasks into groups of limited size, while observing minimum time lags between others. E.g., in car sequencing, after a certain number of cars with the same color, the color needs to be changed. In addition, special options like sun roofs can only be installed at every other car, see e.g., [EGN08]. To capture those different application areas, we formulate the side constraints as abstract *generalized capacity constraints* which we will consider alongside the described sequence-dependent setups.

For solving the overall problem, we implement the general algorithmic framework for integrated sequencing and allocation problems which we discussed in Section 6.3. Our work is based on a cooperation with Sachsenmilch, one of Europe's largest and most modern dairy factories. The implemented algorithm is particularly adapted to their bottleneck filling line, and it is evaluated in an extensive computational study utilizing different allocation subroutines. Based on production data from Sachsenmilch, 180 realistic instances were generated, and each of them was tested in 16 different problem scenarios. In our study, we compare the different algorithms, and evaluate their performance with an ATSP lower bound, proving for the data of Sachsenmilch's current production an optimality gap of only 2 %, and of roughly 15 % on average for the generated instances. Due to the presumed weakness of the lower bound, the gap of 2 % makes us believe that the former solution is in fact optimal. In addition to the above framework, we consider a heuristic which may be a natural starting point in (semi-)manual planning.

Outlining this chapter, we give a formal problem definition in Section 10.1 before investigating structural properties of the problem and complexity issues in Section 10.2. Based on these insights, we formulate different algorithm variants in Section 10.3. Finally in Section 10.3.1 we report on the corresponding computational study.

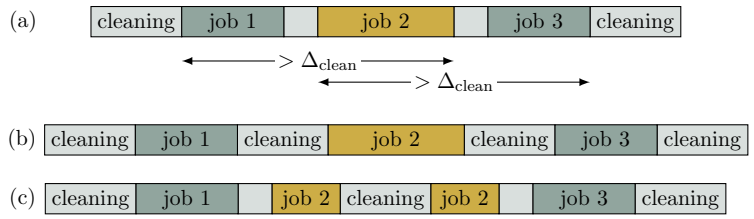


Figure 10.1: The schedule (a) contains only jobs and sequence-dependent setups, two of them being cleanings. The time between the cleanings exceeds the maximum time lag Δ_{clean} between cleanings. The schedules (b) and (c) below are feasible with respect to cleaning time lags. The former replaces two existing sequence-dependent setups while the latter preempts the middle job to require only one additional cleaning.

Related work. In Section 6.2 general aspects of integrated sequencing and allocation problems have been discussed. Here, we focus on results which address side constraints of similar structure as those in our problem from dairy industry. Even though incorporating batching aspects—e.g., resulting from the tank in cream production—our problem differs from classic batch scheduling. There, for single machines, most literature focuses on the sum of weighted completion times or the maximum lateness objective; see [PK00] for a review. In fact, the makespan variant of the single machine case reduces again to the *asymmetric traveling salesman problem* (ATSP).

In car manufacturing, one can observe batching aspects that are similar to ours. E.g., the paint needs to be changed after a certain number of cars with the same color [EGN08]. Still, in contrast to our problem, only consecutive subsequences can form groups in this setting.

Previous work on the planning of filling lines in dairy industry mainly focuses on mixed-integer linear programming (MILP) models to minimize weighted cost functions comprising setup cost, storage cost and others. Most recently, Kopanos et al. [KPG10] proposed a MILP model for parallel filling machines, taking into account machine- and sequence-dependent setups, due dates, and certain tank capacities. Further MILP models were proposed for different problem settings, e.g., by [DS08, LEGvB+05, MNS07]. For the common planning horizon of one week, the models in [KPG10, DS08] could compute optimal solutions at very low computational cost. Based on different relaxations of their model, Marinelli et al. [MNS07] also proposed an algorithm which is heuristically almost optimal, but at a high computational expense.

Different to our problem, regular cleanings and sterilizations of the filling line are performed at the end of the day in all these papers. In combination with their very restricting sequencing constraints, this turns these problems to packing problems rather than to sequencing problems as in our case. The results are not adaptable to our problem. To the best of our knowledge, flexible cleaning scheduling has not been considered in the literature yet.

10.1 Problem formulation

We consider an integrated sequencing and allocation problem as introduced in Section 6.1. The aim is to schedule a set of jobs $J := \{1, 2, \ldots, n\}$ where a job $j \in J$

is specified by the following parameters:

$$\begin{aligned} \text{processing time} &\quad p_j > 0\,, \\ \text{volume} &\quad v_j > 0\,, \\ \text{base (e.g., yogurt or cream)} &\quad b_j\,, \\ \text{(possibly non-existing) flavor} &\quad f_j\,, \\ \text{article number} &\quad a_j\,. \end{aligned} \tag{10.1}$$

The base, the flavor, and the article number are correlated in the following way: If two jobs differ in their base then they also differ in their flavor and article number. And analogously, if the flavors are different then so are the article numbers. We refer to a job with the parameters (10.1) as *dairy job* if the parameters observe also the just described correlation. This correlation plays a major role in the structure of sequence-dependent setups which we will define below.

The overall goal of our planning problem is to compute a *schedule* $S = (\pi, A(\pi))$, subsuming a processing *sequence* π of the jobs J and a feasible (not yet defined) *allocation* $A(\pi)$, which minimizes the makespan, i.e., the completion time of the last job. An allocation $A(\pi)$ for π is a sequence which contains in addition to the actual jobs certain setups and waiting periods, and the jobs appear in the same relative order as in π. We refer to jobs, setups and waiting periods jointly as *tasks*, and denote the duration of a tasks s by p_s. Moreover, we denote by $\mathcal{T}$ the set of all possible tasks

$$\mathcal{T} \;:=\; J \cup \left\{ s \,\middle|\, s \text{ setup with } p_s \in \mathbb{R}_{\geq 0} \right\} \cup \left\{ s \,\middle|\, s \text{ waiting period with } p_s \in \mathbb{R}_{\geq 0} \right\}.$$

Setups and waiting periods result from the following constraints:

Sequence-dependent setups. When scheduling jobs $i, j \in J$ consecutively, then this requires *sequence-dependent setup work* between i and j of duration

$$s_{ij} \;=\; \begin{cases} C_b & \text{, if } b_i \neq b_j & (\textit{product change}) \\ C_f & \text{, if } b_i = b_j \text{ and } f_i \neq f_j & (\textit{sort change}) \\ C_p & \text{, if } b_i = b_j,\ f_i = f_j \text{ and } a_i \neq a_j & (\textit{packaging change}) \\ 0 & \text{, if } b_i = b_j,\ f_i = f_j \text{ and } a_i = a_j & \end{cases} \tag{10.2}$$

for some values $C_b \geq C_f \geq C_p > 0$. Intuitively, the setup duration C_b corresponds to a so called *product change* which includes a change of the base, the flavor, and the packaging material where the latter can be associated with the article number. A *sort change* requires only to change the flavor and the packaging material, and hence, its duration C_f is shorter than C_b. And finally, the duration C_p of a basic *packaging change* is even shorter.

Cleaning constraint. Cleanings have to be performed on a regular basis, i.e., there is a *maximum cleaning time lag* Δ_{clean} not to exceed between the completion and start of consecutive cleanings. Waiting periods directly preceding a cleaning do not account for this time lag. Cleanings can be inserted into a schedule by replacing sequence-dependent setups or by preempting jobs.

Frequency constraints. Considering a set of *identical jobs* $J' \subseteq J$—i.e., for any $i, j \in J'$ it holds $a_i = a_j$, $f_i = f_j$ and $b_i = b_j$—a maximum filling frequency has to be observed. This means that if job $i \in J'$ is processed before some other job $j \in J'$, a time lag of at least $\Delta_{J'} \geq 0$ has to be observed between the completion of i and the start of j. This may require job j to wait.

Capacity constraints. There are $k \in \mathbb{N}$ constraints $R_1, \ldots, R_k$ that require to partition the jobs into *(processing) subgroups* with respect to different properties. A *subgroup* is a contiguous subsequence of tasks in a task sequence σ, i.e., it may contain jobs as well as setups and waiting periods.

Each constraint R_ℓ, $\ell = 1, \ldots, k$, is defined by a set of *job characteristics* C_ℓ (e.g., certain base types), a non-negative *weight function* $w_\ell : \mathcal{T} \to \mathbb{R}_{\geq 0}$ (e.g., reporting the duration or volume of a task), a *maximum subgroup weight* $W_\ell \geq \max_{s \in \mathcal{T}} w_\ell(s)$, and a *minimum time lag function* Δ_ℓ, stating for each subgroup which minimum time lag to respect to the next subgroup. This time lag is independent of the choice of the subgroup processed thereafter.

A subgroup U is an *R_ℓ-subgroup* if it starts and ends with jobs of characteristic C_ℓ and its total weight $\sum_{s \in U} w_\ell(s)$ does not exceed W_ℓ. A task sequence σ and a set of R_ℓ-subgroups $\mathcal{U}$ *satisfy* constraint R_ℓ if every job of characteristic C_ℓ is contained in exactly one subgroup from $U \in \mathcal{U}$, and after every such subgroup the minimum time lag $\Delta_\ell(U)$ is respected.

Based on the above constraints, we can now formulate the allocation problem for the scheduling of filling lines[1].

Filling line allocation problem

Given: Sequence π of n dairy jobs J; durations C_b, C_f and C_p of a product change, sort change and packaging change according to (10.2), respectively; the cleaning time lag Δ_{clean}; set of job sets $\mathcal{J} := \{J' \mid J' \subseteq J\}$ with time lags $\Delta_{J'}$ for any $J' \in \mathcal{J}$ representing the frequency constraints; and capacity constraints $R_1, \ldots, R_k$.

Task: Compute an allocation $A(\pi)$ comprising a set of setups $\mathcal{S}$, a set of waiting tasks $\mathcal{W}$ and a sequence σ of $J \cup \mathcal{S} \cup \mathcal{W}$ such that for any $i, j \in J$ it holds that $\pi(i) < \pi(j)$ if and only if $\sigma(i) < \sigma(j)$, and such that all sequence-dependent setups are performed, and the cleaning constraint, the frequency constraints and the capacity constraints are satisfied, while thereby minimizing

$$\sum_{s \in J \cup \mathcal{S} \cup \mathcal{W}} p_s \,.$$

The jobs may be preempted for cleanings but not for other jobs. In the task sequence σ preempted jobs appear as different subjobs.

Note that the start time of a task s in sequence σ, and hence, in the schedule S, can be computed by summing up the processing times of all tasks which occur before s in σ.

[1] For clarity, some details are omitted; yet they are included in our calculations.

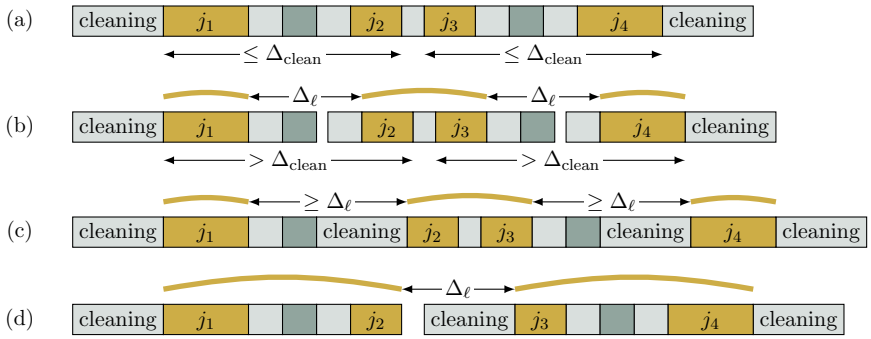


Figure 10.2: Fig (a) shows a schedule containing only jobs (darkgray and yellow) and sequence-dependent setup tasks (lightgray), violating the cleaning constraint and requiring the jobs $J_\ell := \{j_1, \ldots, j_4\}$ to be grouped into subgroups with respect to capacity constraint R_ℓ. By the maximum subgroup size W_ℓ, any R_ℓ-subgroup can contain at most two jobs from J_ℓ, and a constant minimum time lag Δ_ℓ has to be observed after it. If choosing the subgroups (yellow bows) in order to minimize the total waiting time without consideration of the cleaning constraint, we result in (b). However, due to the two waiting periods in (b), the time lag between the cleanings increases such that two additional cleanings become necessary; see (c). Alternatively, (d) requires only one additional cleaning, and more importantly, it has a smaller makespan than (c). Note that (d) respects the minimum cleaning time lag since waiting periods directly preceding cleanings do not account for the cleaning lag.

Considering the sequencing aspect as part of the problem, we refer to the problem of computing an overall schedule $S = (\pi, A(\pi))$ of minimum makespan as *dairy problem*.

The side constraints described above may have global effects on the whole sequence. For instance, if separately satisfying the cleaning constraint before inserting waiting periods, the cleaning constraint may again be violated afterwards. Or contrarily, the choice of subgroups may be suboptimal when not considering the cleaning constraint; see Figure 10.2. Thus, aiming for optimal solutions, the constraints need to be considered jointly.

Finally, note that every sequence possesses a feasible schedule. A very simple schedule considers every characteristic job as a subgroup on its own, and satisfies minimum time lags in some arbitrary (suboptimal) way. When inserting additional cleanings as the last step, all time lags remain unchanged or get even larger.

10.2 Structural properties of the allocation problem

In this section, we will investigate structural properties of our problem in order to gain insights for designing algorithms.

10.2.1 Common structure of the side constraints

The side constraints introduced in Section 10.1 exhibit a very similar structure. In fact, we can formulate all of them as *generalized capacity constraints*. Such a constraint is defined by the same parameters as normal capacity constraints. The only difference

R_ℓ	C_ℓ	w_ℓ	W_ℓ	$c_{\ell,\sigma}(U)$
frequency	identical jobs J'	1	1	$\max\{0,\ \Delta_{J'} - \bar{p}_U\}$
capacity	characteristic C_ℓ	$w_\ell(t)$	W_ℓ	$\max\{0,\ \Delta_\ell(U) - \bar{p}_U\}$
cleaning	all jobs	–	–	$\left\lceil \frac{p_U}{\Delta_{\text{clean}}} - 1 \right\rceil \cdot p_{\text{clean}} + \max\{0,\ p_{\text{clean}} - \bar{p}_U\}$

Table 10.1: Parameters for the generalized capacity constraint, denoting by p_U the total duration of all tasks in the subgroup U of task sequence σ and by $\bar{p}_U$ the total duration of all tasks that occur after U and before the beginning of the next subgroup. Note that $\bar{p}_U$ depends only on U and σ, i.e., it is independent of the subsequent subgroup.

is that instead of assigning time lags to the subgroups and aiming for a selection of subgroups resulting in a minimum total waiting time, we now assign abstract *cost* to subgroups and aim to minimize the total cost.

Generalized capacity constraint. A *generalized capacity constraint* R_ℓ is defined by a set of *task characteristics* C_ℓ (e.g., certain base types, waiting tasks or setups), a non-negative *weight function* $w_\ell : \mathcal{T} \to \mathbb{R}_{\geq 0}$, a *maximum subgroup weight* $W_\ell \geq \max_{s \in \mathcal{T}} w_\ell(s)$, and non-negative *cost functions* $c_{\ell,\sigma}$ on all possible subgroups of a given task sequence σ. An *R_ℓ-subgroup* is defined analogously as for normal capacity constraints, and a set of R_ℓ-subgroups $\mathcal{U}$ *satisfies* constraint R_ℓ if every task of characteristic C_ℓ is contained in exactly one subgroup from $U \in \mathcal{U}$. In this case $\mathcal{U}$ is called *feasible*.

In Table 10.1, we formally define the parameters for modeling our constraints as generalized capacity constraint. The intuition behind these definitions is described in the following. For normal capacity constraints, we can simply interpret the waiting time resulting from a subgroup's time lag as cost. Also for frequency constraints, the modeling is rather trivial (and an overkill, actually). We define the task characteristic to cover exactly the identical jobs J', and we choose the weight function and the maximum subgroup weight such that every subgroup contains exactly one job from J'. A subgroup's cost is the waiting time necessary to observe the minimum time lag $\Delta_{J'}$.

For the cleaning constraint, we define all task types to be contained in the task characteristic, and we interpret a chosen subgroup as follows: Within the subgroup, we insert additional cleanings every Δ_{clean} time units by preempting the job currently run. After the subgroup, we replace the setup and/or reduce the waiting time to insert another cleaning. By this modeling, any subgroup size is feasible, and every combination of replacing and not replacing sequence-dependent cleanings is represented by a choice of subgroups. The subgroup's cost result from the added cleanings, and additional increase in makespan after the subgroup.

Note that a subgroup's cost with respect to a certain constraint strongly depends on the subgroups corresponding to other constraints, since the abstract costs may represent setup or waiting tasks that have to be added to the sequence. E.g., if inserting cleanings into the schedule for the cleaning constraint, necessary waiting times of other constraints may decrease, and so would the cost of the associated subgroups. Thus, by optimally satisfying one constraint after the other, in general, we do not obtain an optimum schedule, see again Figure 10.2.

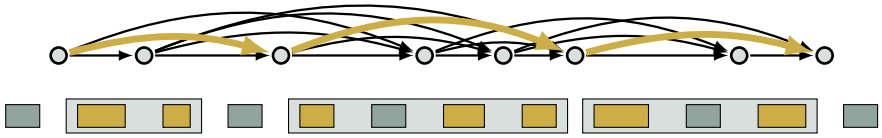


Figure 10.3: A sequence of characteristic (yellow) and non-characteristic (darkgray) tasks (with some intermediate spacing for clarity). Vertices and arcs are chosen as described below. The marked path corresponds to the choice of subgroups as shown in the three boxes.

In the following, we assume an (infeasible) allocation with the corresponding task sequence σ to be given, containing the actual jobs, sequence-dependent setups, and possibly already additional cleanings or waiting periods.

Lemma 10.1 *Considering a fixed task sequence and a generalized capacity constraint R_ℓ, one can compute a minimum cost feasible set of R_ℓ-subgroups in time $\mathcal{O}(n^2)$.*

Proof. We reduce the problem to the shortest path problem. For a given sequence of tasks, we define a graph whose vertices lie on the natural time axis of the sequence. For each characteristic task, we place a vertex at its start time. Additionally, a vertex is placed at the completion time of the last characteristic task. We insert an arc pointing from one vertex to another if and only if it is a forward arc in the natural sense of time, and the tasks *below* that arc, i.e., the tasks that lie in the time interval covered by the arc, form a feasible subgroup with respect to the constraint under consideration. More precisely, the subgroup is formed by the subsequence of tasks, starting with the first characteristic job below that arc, and ending with the last such job, respectively.

Since any feasible subgroup is represented by an arc in the graph, this yields a one-to-one correspondence between feasible sets of subgroups and paths in the graph. By using the subgroup's costs as arc weights, a shortest path in this graph corresponds to a minimum cost choice of subgroups; see Figure 10.3. Note that the number of arcs is quadratic in the number of characteristic tasks C_ℓ. Thus, the lemma follows by Dijkstra's shortest path algorithm [Dij59]. □

10.2.2 Satisfying minimum time lags

To deal with minimum time lags resulting from frequency and capacity constraints, we introduce flexible *waiting tasks* instead of assigning fixed waiting periods right away. Such a task s is defined by two, not necessarily adjacent jobs $\text{pred}(s)$ and $\text{succ}(s) \in J$, the *predecessor* and *successor*, respectively, and a *minimum time lag* $\Delta_s \geq 0$ to keep between the completion of $\text{pred}(s)$ and the start of $\text{succ}(s)$. Its duration p_s, the *waiting time*, results as additional time (currently) necessary to respect the time lag. As we will see below, we can always schedule this waiting time right before the start of $\text{succ}(s)$. However, any changes in the task sequence, e.g., adding setup or waiting tasks, require to recompute the actual waiting times of the waiting tasks.

Given a sequence of tasks σ, one can compute the waiting times of all waiting tasks in σ by a simple greedy method. We sort the waiting tasks according to the order in which their successors appear in σ, breaking ties arbitrarily. Then, we satisfy the waiting tasks in this order by adding the necessary waiting time right before the start of their

successors or rather before the setup task preceding it if there is such. This procedure produces a minimum total waiting time as we will argue in the following.

Observation 10.2 *The greedy method satisfies the waiting tasks in a given task sequence at minimum total waiting time.*

Proof. Consider some task sequence σ. Firstly, note that we can assume w.l.o.g. that among the setup and waiting tasks scheduled between two consecutive jobs, the waiting tasks are scheduled before the setup tasks. This is preferable since waiting periods account for the maximum cleaning time lag if being scheduled after a cleaning, but not if being scheduled right before it. In the following, we only speak of scheduling a waiting task before a job, even though meaning scheduling it before the corresponding setup, if existing.

Now, consider an optimal schedule of waiting times for σ which satisfies all waiting tasks. We will transform this schedule into a schedule as produced by our greedy method without increasing the total waiting time. Let $s_1, s_2, \ldots, s_w$ be the waiting tasks in σ, labeled in the order in which they are considered by the greedy method. Clearly, s_1 can be moved before its successor without changing its waiting time, and without impact on other waiting tasks. We show that we can also schedule all other waiting times right before their successors without increasing the makespan. After these transformations, whenever possible, we transfer waiting time from $s_{\ell-1}$ to s_ℓ, $1 < \ell \leq w$, which leads to the same schedule as produced by our greedy method.

For showing the above claim, assume that for some $1 < \ell \leq w$ all tasks $s_1, \ldots, s_{\ell-1}$ are already scheduled before their successors and for s_ℓ this is not the case. If there is no successor of another waiting task lying in between $\text{pred}(s_\ell)$ and $\text{succ}(s_\ell)$, then we can simply move s_ℓ before $\text{succ}(s_\ell)$ without further impact. Otherwise, if there is a successor $\text{pred}(s_{\ell'})$ lying in between $\text{pred}(s_\ell)$ and $\text{succ}(s_\ell)$ it holds $\ell' < \ell$, and thus, $s_{\ell'}$'s waiting time is scheduled right before $\text{pred}(s_{\ell'})$, and in particular, it lies in between $\text{pred}(s_\ell)$ and $\text{succ}(s_\ell)$. By transferring waiting time from s_ℓ to $s_{\ell'}$, if necessary, all minimum distances are kept when moving s_ℓ's waiting time before $\text{succ}(s_\ell)$. □

10.2.3 Complexity

As mentioned in the related work section, even greatly simplified variants of our problem are MAX SNP-hard. Also if only focusing on sequence-dependent setups and the cleaning constraint the problem remains hard.

Theorem 10.3 *The dairy problem remains strongly NP-hard in the absence of frequency and capacity constraints and for setup parameters $C_b = C_f = C_p =: p$ and cleaning durations p_{clean} for arbitrary constants $p_{clean} > p \geq 0$.*

Proof. The hardness of this restricted problem variant of the dairy problem follows from a reduction from the strongly NP-complete 3-Partition problem [GJ79], which was defined in Section 3.1. Consider a 3-Partition instance with a given set A of $3\,m$ elements from $\mathbb{N}^+$ for some $m \in \mathbb{N}^+$ with $B/4 < a < B/2$ for all $a \in A$, where $B := \frac{1}{m}\sum_{a \in A} a$. The instance A is a yes-instance if it can be partitioned into m disjoint sets $A_1, \ldots, A_m$ with $\sum_{a \in A_i} a = B$ for $i = 1, \ldots, m$. We will construct an instance I of the considered restricted variant of the dairy problem with $p > 0$ arbitrary, whose jobs can be scheduled with makespan $D := (m-1)\,p_{\text{clean}} + m(B + 2\,p)$ if and only if A is a yes-instance.

We define the instance I as follows. For each $a \in A$, we choose a job with duration a, and we define the maximum cleaning time lag Δ_{clean} to be $B + 2p$. If A is a yes-instance, then we schedule jobs that correspond to the same subset A_i of a feasible solution consecutively for each $i = 1, \ldots, m$ and concatenate the triples in an arbitrary order. The total duration of such a job triple, including the two intermediate setup tasks, equals exactly Δ_{clean}. Hence, by replacing every third of the unit setups by a cleaning, we obtain a feasible schedule of makespan D.

Conversely, assume that I exhibits a schedule of makespan at most D. Due to the unit setup duration, this directly implies that there are at most $m - 1$ cleanings in the schedule, and every cleaning replaces one setup. Considering the jobs scheduled between two consecutive cleanings, we obtain at most m subsets of jobs. Assume that in one of these subsets the job durations sum up to strictly less than B. Then, by the total duration of jobs, there is another subset whose job durations sum up to more than B which violates the cleaning constraint. Thus, in every such subset, the job durations sum up to exactly B, and hence, A is a yes-instance. □

10.3 Algorithms and computational study

We utilize our generic framework for integrated sequencing and allocation problems from Section 6.3 which is based on a genetic algorithm for the overall sequencing task. This approach requires to implement two subroutines: A fast algorithm for the fixed-sequence allocation problem, which serves as an evaluation function for the generated sequences, and an algorithm for generating an initial population of sequences. Since the dairy problem is not as complex as the earlier discussed coil coating problem, good results could already be achieved utilizing the standard randomly generated initial population described in Section 6.3. Hence, it remains open to specify algorithms for the allocation problem.

Allocation subroutines. Due to our computation time limit of a few minutes, we can only use allocation subroutines whose runtime does not exceed $\mathcal{O}(n^2)$. Thus, it is unlikely to solve the scheduling problem to optimality. We consider two variants of a generic heuristic, summarized as Algorithm 10.1. In order to investigate effects as shown in Figure 10.2, the two variants differ in the order in which they satisfy the cleaning and the capacity constraints—First indicates that the cleaning constraint is satisfied first. Note that in the first two steps of the algorithm, where we add the sequence-dependent setups and the flexible waiting tasks for the frequency constraints, no optimization potential is lost, since setups can be replaced for cleanings later on, and the durations of the waiting tasks can be flexibly changed.

Both variants of the algorithm are considered with *greedy* and *shortest path* subroutines. The latter solve the subproblems to optimality, utilizing Lemma 10.1, while the former proceed greedily as follows: For the capacity constraints, we add consecutive characteristic jobs to a current subgroup until the maximum capacity is exceeded. In case of the cleaning constraint, cleanings are added before the latest possible job by replacing a setup and/or reducing waiting time; i.e., never by preempting jobs. We denote the four resulting algorithms by GreedyLast, GreedyFirst, SPLast, and SPFirst, respectively, where First and Last refer to the point when the cleaning constraint is satisfied.

Alg. 10.1: Generic algorithm for the filling line allocation problem

```
insert sequence-dependent setups;
insert waiting tasks for frequency constraints as described in Section 10.2.2 ;
if algorithm of type First then
    satisfy cleaning constraint;
    satisfy capacity constraints, and, if necessary, satisfy cleaning constraint again;
else
    satisfy capacity constraints;
    satisfy cleaning constraint;
```

ATSP heuristic. Alternatively to the above framework, we consider a heuristic algorithm which comes close to (semi-)manual planning as typically performed in practice. It computes a solution taking into account only the sequence-dependent setup durations s_{ij} and partially also the time lags from the frequency constraint. Since we aim for a formulation as ATSP, the latter is only regarded when jobs are scheduled consecutively. Technically, this means to consider the following modified sequence-dependent setup durations

$$s'_{ij} \;:=\; \max\left\{s_{ij}\,,\; \Delta_{ij}\right\} \qquad \text{where} \qquad \Delta_{ij} \;:=\; \begin{cases} \max\limits_{J' \ni i,j} \Delta_{J'} & \text{, if } \;\exists\; J' \ni i,j \\ 0 & \text{, otherwise} \end{cases} \tag{10.3}$$

where J' denote job sets that represent a frequency constraint. Based on the values s'_{ij}, we define an ATSP instance on the jobs and compute a solution—our sequence—with Helsgaun's efficient implementation of the Lin-Kernighan Heuristic [Hel00], which can be assumed to be optimal for our instance sizes. The allocation is then determined by GreedyFirst. We refer to the overall heuristic as ATSPH.

Lower bound. To assess the performance of the different algorithms we compute instance-based lower bounds. As for the heuristic ATSPH, we solve the corresponding instance of ATSP with distances s'_{ij} as specified in (10.3). The computed tour length yields a lower bound on the minimum cost of a solution of the dairy instance.

10.3.1 Computational study

Our cooperation with Sachsenmilch was initiated with the goal to examine the optimization potential of the manual planning of their bottleneck filling machine. To this end, we were provided with data from the current production, for which our algorithm computed solutions of equal quality as the current manual plans. Our ATSP lower bound proved an optimality gap of 2 % which shows both side's good performance. Since the lower bound ignores several costly aspects, it is most likely that this solution is in fact optimal.

In the following, our algorithm is examined for various additional realistic instances in several problem scenarios.

Generation of instances and problem parameters. Based on the data from Sachsenmilch, 180 realistic instances were randomly generated. As in the original data, we consider a planning horizon of one week. This results in instances of 15–35 jobs, depending on

the job durations. When generating the instances, we consider different classes of job durations, compositions of base types, and jobs with frequency constraint. The job durations are randomly chosen according to the durations in the *original* instance, or focusing especially on *small* or *large* durations. Concerning the base types, we either consider *all* base types of the original data, only *yogurt products*, or only *cream products* (e.g., creme fraiche or classic cream). Moreover the percentage of jobs with multiple fillings, i.e., with frequency constraint, varies between 20 % and 40 %.

With respect to general problem parameters, we consider different values for Δ_{clean}, for the cream tank size, and the minimum time lags of the frequency constraints. For all these values, we consider the original value and a smaller value, which usually results in a more complicated planning. In addition, we consider different sequence-dependent setup durations, and two capacity constraints: the *cream* constraint, described in the very beginning of this chapter, and a *duration* constraint. The latter requires to group certain jobs into subgroups of limited total duration. We consider the settings *cream* only and *cream & duration*.

We test the four algorithms originating from Algorithm 10.1 on all our instances for all parameter scenarios, resulting in 2880 test cases for each algorithm. Due to a non-disclosure agreement with Sachsenmilch, we can only report on relative data and results.

Implementation details. As for the coil coating problem, the parameters for the genetic sequencing algorithm were determined by rigorous testing. We consider a constant population size of 250, and stop the algorithm, if the makespan has not been improving for 150 iterations, or when the run time limit of 150 seconds is exceeded. The runtime limit is necessary to guarantee practicability in practice, which means to allow a fast replanning. The algorithm is run eight times in parallel in order to compensate fluctuation. For solving the shortest path problem in our algorithm, we use Dijkstra's algorithm from the Boost Graph Library[2].

Interpretation of results. As a surprising result of our computational study, the greedy algorithm variants outperformed the shortest path variants in almost all cases as we will see below and in Table 10.2 and the corresponding Figure 10.4. For our tests, we considered the following parameter settings: The setting *original* comprises all of the 2880 instances in which all parameters were set according to the original instance of Sachsenmilch. The additional settings we considered filter all instances for one particular parameter, i.e., an instance belongs to such a setting if this parameter is set as stated in the table, no matter how the remaining parameters are set.

We compared the better result of the two greedy algorithms GreedyLast and GreedyFirst (BestGreedy) with the better result of the two shortest path variants SPLast and SPFirst (BestSP). BestGreedy performed always better than BestSP, up to 4 % in makespan average, except when considering the cream and duration constraint in combination; see Table 10.2. However, in this case the gap is negligible. In fact, for the orginial test setting, BestSP is never better than BestGreedy. If utilizing BestSP, the worst case gap compared to BestGreedy is 6 %, whereas conversely this gap may grow up to 210 %. Also the runtimes differ greatly, even though this is not very surprising; see again Table 10.2. While BestSP produces better solutions for

[2] http://www.boost.org/

	BestSP/BestGr. [%]				GreedyLast/ LB [%]				average runtime [s]			
parameter	∅	σ	min	max	∅	σ	min	max	GL	GF	SPL	SPF
original	1.007	0.015	1.000	1.050	1.105	0.057	1.007	1.203	5	6	39	55
all prod.	1.014	0.079	0.985	1.876	1.117	0.085	1.000	1.894	6	6	49	71
yogurt prod.	1.017	0.025	0.948	1.120	1.109	0.065	1.010	1.390	5	6	44	67
cream prod.	1.003	0.058	0.946	2.120	1.201	0.278	1.031	3.943	7	8	53	75
original dur.	1.010	0.020	0.974	1.093	1.146	0.152	1.007	2.769	7	7	53	77
small dur.	1.009	0.018	0.956	1.120	1.148	0.151	1.007	2.862	10	11	81	118
large dur.	1.014	0.098	0.946	2.120	1.133	0.220	1.000	3.943	2	2	13	18
original tank	1.007	0.017	0.946	1.120	1.117	0.081	1.000	1.716	6	7	49	72
small tank	1.043	0.155	0.949	2.120	1.321	0.411	1.000	3.943	6	7	49	65
cream con.	1.015	0.073	0.946	2.120	1.160	0.213	1.000	3.943	6	6	47	67
cream & dur.	0.999	0.003	0.960	1.016	1.097	0.078	1.000	1.716	5	6	41	63
original Δ_{clean}	1.011	0.045	0.956	1.596	1.121	0.108	1.000	2.671	6	7	49	69
small Δ_{clean}	1.012	0.069	0.946	2.120	1.164	0.224	1.000	3.943	6	7	49	71

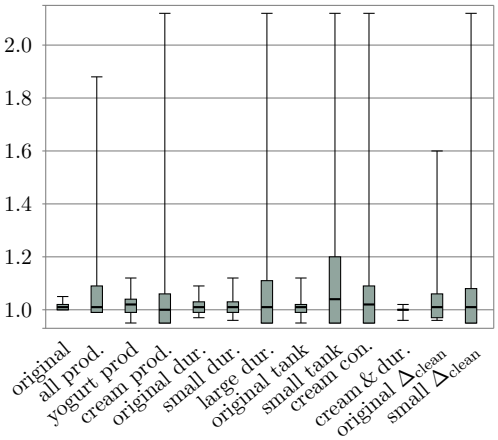
Table 10.2: Results of the computational study. Figure 10.4 shows related plots. By ∅ and σ, we denote the average and the standard deviation, respectively. The values min and max refer to the smallest and largest value attained. To the algorithms in Algorithm 10.1, we refer by their initials GL,GF, SPL and SPF.

the allocation subproblem, the faster running time of BestGreedy allows for more iterations of the genetic algorithm and hence better overall solutions. Moreover, BestSP inserts cleanings by preemption, which makes it harder to incorporate these cleanings into later waiting times.

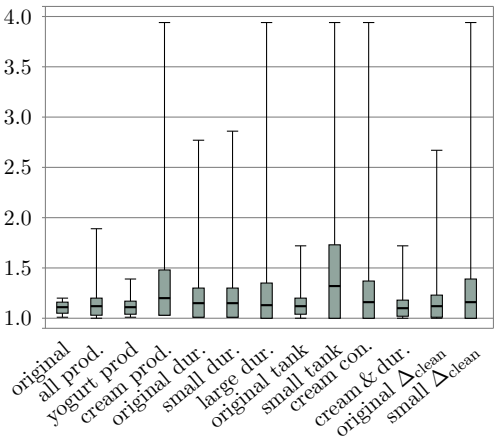
If comparing the two greedy and the two shortest path approaches with each other, the difference of the performance is not that striking. The average performance is almost identical. However, considering the worst case behavior, in both cases the Last-algorithm is only about 6 % worse than the First-algorithm, where conversely, the gap can attain 15 %. This may be due to the advantage of scheduling the inflexible cleanings as late as possible.

The optimality gap computed with the ATSP lower bound is roughly 15 %, and it is slightly better for the greedy algorithms than for the shortest path algorithms. Exemplary for our best algorithm GreedyLast, the results are given in Table 10.2. For all algorithms, the optimality gap is much worse for a small tank size. However, in this case the lower bound performs very poorly, so that one may assume this gap to be far away from being tight.

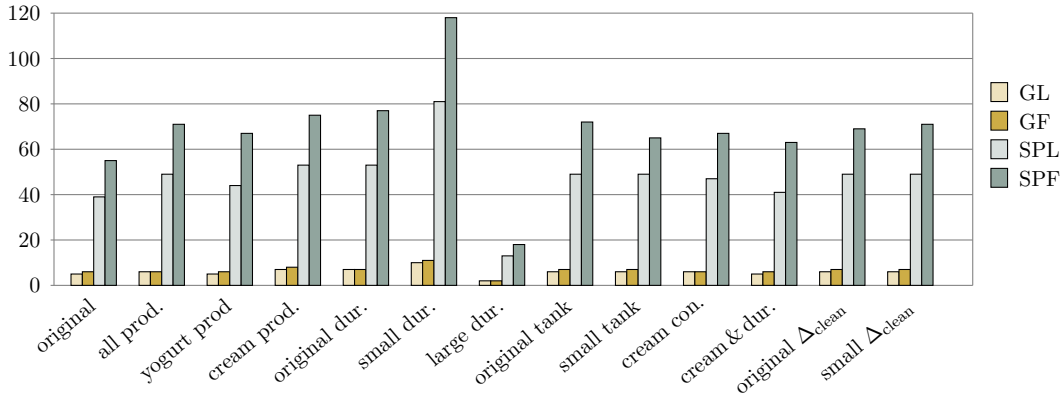
The heuristic ATSPH computes schedules with on average twice the makespan of the solutions of the other algorithms. While expert planners can still improve these solutions, our genetic algorithmic framework provides a considerably improved base for semi-manual planning and also enables automatic planning.



(a) Ratio BestSP/BestGreedy.



(b) Ratio GreedyLast/ LB.



(c) Average runtime.

Figure 10.4: Results of the computational study as shown in Table 10.4. To the algorithms in Algorithm 10.1, we refer by their initials GL,GF, SPL and SPF. The boxes in (a) and (b) show an interval of $-/+$ the standard deviation around the average value.

Chapter 11

Conclusions

In contrast to the first part of the thesis, where the focus was mainly on theoretical insights, in the second part, the major goal was to develop efficient algorithm to be used in every-day planning in industry—which of course also necessitated theoretical investigations on the way. We considered two problems that stemmed from projects with industry, a scheduling problem for coil coating lines and a planning problem for filling lines in dairy industry. Both problems are typical examples of a class of complex sequencing problems as they can be found in virtually all branches of industry.

Capturing the essential characteristics of such problems, we defined the class of so called *integrated sequencing and allocation problems*. It accentuates two components of the problem, the choice of the actual processing sequence and the additional decisions to be taken for this sequence, both of which largely impact the makespan of the solution. These components are mirrored in our generic algorithmic framework for such kind of problems. The general concept of this approach is based on a standard and and easy-to-implement genetic algorithm while its strength lies in the idea of algorithmically separating the generic sequencing aspect from the very problem-specific decisions for a chosen sequence. In the end, the success of the approach crucially depends on the quality of the algorithms for this latter *allocation subproblem*.

For the coil coating problem, the allocation problem led us to the maximum weight independent set problem in m-composite 2-union graphs. Considering exact algorithms, we almost closed the complexity gap in terms of the variable m, with the only remaining question being whether the problem allows for a fixed-parameter algorithm—which requires the algorithm to guarantee a runtime of $\mathcal{O}(2^m \cdot \text{poly}(n))$ in contrast to the $\mathcal{O}(n^{2m})$ runtime of our dynamic program. However, for a parameter based on the number of maximal cliques, van Bevern et al. [vBMNW12] showed fixed-parameter tractability.

Turning to a different topic, even though not being the major focus of our work, there are interesting open questions concerning the approximability of the maximum weight independent set problem. On the one hand, there is a $2t$-approximation algorithm for the very general class of t-interval graphs [BYHN$^+$06], while on the other hand, for the very restricted class of strip graphs, we know that the unweighted problem variant allows for a 1.582-approximation algorithm [COR06]. Surprisingly, it is not clear whether any of the graph classes in between, i.e., t-union graphs or (m-composite) 2-union graphs, allow for better approximation guarantees than $2t$.

Concerning our general model for the concurrent setup allocation problem, we point out that the interpretation of allocations as independent sets in 2-union graphs is strongly based on the number of tanks at each coater being two. Whenever a chosen tool interval

ends, we switch the tank on that particular coater. Since there are only two tanks, it is clear to which tank to switch to. Assuming more than two tanks, a selection of tool intervals does not properly define a tank assignment anymore. Hence, addressing a setting with three or more tanks would require a fundamentally new approach.

Turning towards the dairy problem, already the complexity of the underlying allocation problem is a first open question. From a practical point of view, however, it will be most likely inevitable to use heuristic allocation algorithms. Even if the problem was allowing for polynomial-time algorithms, we expect them to be rather of dynamic programming nature with too large runtimes than to be suitable subroutines for the overall sequencing approach.

The heuristics we considered for the filling line allocation problem satisfy the different generalized capacity constraints one by one, differing in the order of constraints and the way in which feasibility is ensured. Our computational study showed that, at least when satisfying the constraints separately, it does not pay off to solve the allocation subproblem in each step to optimality, since this requires rather extreme scheduling decisions—such as preempting jobs for cleanings—which may incur conflicts in later steps, leading to costly overall solutions. From the practical perspective, this is a very interesting observation since solutions produced with greedy subroutines involve less actions for the workers, and are preferred by practitioners for this reason. Still, from the example in Figure 10.1 we know that, at least for the general problem formulation, optimal solutions may necessitate preemption. This leads to the interesting question of identifying classes of instances which allow for optimal non-preemptive solutions, or alternatively, to determine the price of non-preemption.

Stepping back from the allocation subproblems, and focusing on the actual goal of developing efficient algorithms for day-to-day planning in practice, we could achieve very satisfying results for both problems, the coil coating problem and the dairy problem. After all, the careful investigations on the structure of the problems paid off. For the coil coating problem, the 2-union graph model led to a tank assignment rule which turned out to be significantly superior to the simpler greedy FIFO rule. Compared to previous plans, the optimized solutions yield double digit percentage reductions in makespan, greatly exceeding what was deemed possible before. Our optimization module has been in use for the planning of the coil coating line at Salzgitter Flachstahl since December 2009.

For the dairy problem the success of the implemented algorithm was more of analytic nature, approving the high quality of our partner Sachsenmilch's production plans. Their manually planned schedules and the solutions produced by our algorithm were equally good, both achieving an optimality gap of only 2 %—which we believe implies optimality due to the weak lower bounds. Moreover, our extensive computational study suggested, that the planners at Sachsenmilch do not give away optimization potential by strictly avoiding preemption of jobs for cleaning operations. Overall, our analysis reassured them in their way of planning. One remaining open point is the development of better lower bounds in order to assess the results also for more complicated instances more realistically.

Eventually, the algorithmic success of our approach brought back into our mind a discussion we had with PSI Metals some years ago. They asked us for a general framework which allowed for modeling general global setup cost as easily as this was the case for sequence-dependent setup durations in an asymmetric traveling salesman framework, since the latter was mastered very well by their software engineers. At that time, we believed that it was impossible to provide such a tool, and we still believe

that it is not possible to design *one* ready-to-use algorithm that produces acceptable solutions to *all* general sequencing problems. However, from our current point of view, our general framework might be as close as we can get to answering their question —an adequate compromise between a ready-to-use tool and an algorithm developed newly from scratch.

Bibliography

[ABC+99] F. N. Afrati, E. Bampis, C. Chekuri, D. R. Karger, C. Kenyon, S. Khanna, I. Milis, M. Queyranne, M. Skutella, C. Stein, and M. Sviridenko, *Approximation schemes for minimizing average weighted completion time with release dates*, Proceedings of the 40th Annual Symposium on Foundations of Computer Science (FOCS), IEEE, 1999, pp. 32–44. 15, 16

[ABCC06] D. L. Applegate, R. E. Bixby, V. Chvátal, and W. J. Cook, *The traveling salesman problem: A computational study*, Princeton University Press, 2006. 77

[ABMP10] S. Albers, S. K. Baruah, R. H. Möhring, and K. Pruhs (eds.), *Open problems – Scheduling*, vol. 10071, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Germany, 2010. 11, 45

[Ach07] T. Achterberg, *Constraint integer programming*, Ph.D. thesis, Technische Universität Berlin, 2007. 118

[AL97] E. Aarts and J. K. Lenstra (eds.), *Local search in combinatorial optimization*, John Wiley & Sons, 1997. 78

[Ali93] B. Alidaee, *Numerical methods for single machine scheduling with nonlinear cost functions to minimize total cost*, Journal of the Operational Research Society **44** (1993), 125–132. 45

[ANCK08] A. Allahverdi, C. T. Ng, T. C. E. Cheng, and Mikhail Y. Kovalyov, *A survey of scheduling problems with setup times or costs*, European Journal of Operational Research **187** (2008), 985–1032. 77

[AvD09] R. Akkerman and D. P. van Donk, *Analyzing scheduling in the food-processing industry: Structure and tasks*, Cognition, Technology & Work **11** (2009), 215–226. 74

[Bak74] J. K. Baker, *Introduction to sequencing and scheduling*, John Wiley & Sons, 1974. 15

[BCK+07] N. Bansal, H.-L. Chan, R. Khandekar, K. Pruhs, C. Stein, and B. Schieber, *Non-preemptive min-sum scheduling with resource augmentation*, Proceedings of the 48th Annual Symposium on Foundations of Computer Science (FOCS), IEEE, 2007, pp. 614–624. 15

[BDM02] P. Berman, B. DasGupta, and S. Muthukrishnan, *Simple approximation algorithm for nonoverlapping local alignments*, Proceedings of the 13th Annual Symposium on Discrete Algorithms (SODA), SIAM, 2002, pp. 677–678. 99, 101

[Ber00] P. Berman, *A $d/2$ approximation for maximum weight independent set in d-claw free graphs*, Proceedings of the 7th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT), LNCS, vol. 1851, Springer, 2000, pp. 214–219. 101

[BGT97] E. Bampis, F. Guinand, and D. Trystram, *Some models for scheduling parallel programs with communication delays*, Discrete Applied Mathematics **72** (1997), 5–24. 77

[BHLR10] A. Butman, D. Hermelin, M. Lewenstein, and D. Rawitz, *Optimization problems in multiple-interval graphs*, ACM Transactions on Algorithms **6** (2010), Article 40. 96

[BJN+98] C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. P. Savelsbergh, and P. H. Vance, *Branch-and-price: Column generation for solving huge integer programs*, Operations Research **46** (1998), 316–329. 115

[BK80] P. C. Bagga and K. R. Kalra, *A node elimination procedure for Townsend's algorithm for solving the single machine quadratic penalty function scheduling problem*, Management Science **26** (1980), 633–636. 18, 45, 55

[BLMSP06] L. Becchetti, S. Leonardia, A. Marchetti-Spaccamelaa, and K. Pruhs, *Online weighted flow time and deadline scheduling*, Journal of Discrete Algorithms **4** (2006), 339–352. 15

[BNR96] V. Bafna, B. Narayanan, and R. Ravi, *Nonoverlapping local alignments (weighted independent sets of axis-parallel rectangles)*, Discrete Applied Mathematics **71** (1996), 41–53. 101

[BP10a] N. Bansal and K. Pruhs, *The geometry of scheduling*, Proceedings of the 51st Annual Symposium on Foundations of Computer Science (FOCS), IEEE, 2010, pp. 407–414. See also `http://www.win.tue.nl/~nikhil/pubs/wflow-journ3.pdf`. 9, 17

[BP10b] ———, *Server scheduling to balance priorities, fairness, and average quality of service*, SIAM Journal on Computing **39** (2010), 3311–3335. 9, 16, 19

[BSV08] E. Balas, N. Simonetti, and A. Vazacopoulos, *Job shop scheduling with setup times, deadlines and precedence constraints*, Journal of Scheduling **11** (2008), 253–262. 77

[BT06] A. Bellabdaoui and J. Teghem, *A mixed-integer linear programming model for the continuous casting planning*, International Journal of Production Economics, Theoretical Issues in Production Scheduling and Control & Planning and Control of Supply Chains and Production **104** (2006), 260–270. 75

[BYHN+06] R. Bar-Yehuda, M. M. Halldórsson, J. Naor, H. Shachnai, and I. Shapira, *Scheduling split intervals*, SIAM Journal on Computing **36** (2006), 1–15. 96, 99, 100, 101, 135

[CK02] C. Chekuri and S. Khanna, *Approximation schemes for preemptive weighted flow time*, Proceedings of the 34th Annual Symposium on Theory of Computing (STOC), ACM, 2002, pp. 297–305. 15

[Coo71] S. A. Cook, *The complexity of theorem-proving procedures*, Proceedings of the 3rd Annual Symposium on Theory of Computing (STOC), ACM, 1971, pp. 151–158. 102

[COR06] J. Chuzhoy, R. Ostrovsky, and Y. Rabani, *Approximation algorithms for the job interval selection problem and related scheduling problems*, Mathematics of Operations Research **31** (2006), 730–738. 99, 101, 135

[Cow03] P. Cowling, *A flexible decision support system for steel hot rolling mill scheduling*, Computers & Industrial Engineering **45** (2003), 307–321. 75

[CS11] M. Cheung and D. B. Shmoys, *A primal-dual approximation algorithm for min-sum single-machine scheduling problems*, Proceedings of the 14th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems and the 15th International Workshop on Randomization and Computation (APPROX-RANDOM), LNCS, vol. 6845, Springer, 2011, pp. 135–146. 9, 17

[CvB93] G. D. H. Claassen and P. van Beek, *Planning and scheduling*, European Journal of Operational Research **70** (1993), 150–158. 74

[DBC99] M. Delucchi, A. Barbucci, and G. Cerisola, *Optimization of coil coating systems by means of electrochemical impedance spectroscopy*, Electrochimica Acta **44** (1999), 4297–4305. 81, 82

[DBCF05] R. G. Duarte, A. C. Bastos, A. S. Castela, and M. G. S. Ferreira, *A comparative study between Cr(VI)-containing and Cr-free films for coil coating systems*, Progress in Organic Coatings **52** (2005), 320–327. 81

[DCST+95] F. Della Croce, W. Szwarc, R. Tadei, P. Baracco, and R. di Tullio, *Minimizing the weighted sum of quadratic completion times on a single machine*, Naval Research Logistics **42** (1995), 1263–1270. 18, 45

[DFR00] F. Deflorian, L. Fedrizzi, and S. Rossi, *Effects of mechanical deformation on the protection properties of coil coating products*, Corrosion Science **42** (2000), 1283–1301. 81

[Dij59] E. W. Dijkstra, *A note on two problems in connexion with graphs*, Numerische Mathematik **1** (1959), 269–271. 128

[DL05] J. Desrosiers and M. E. Lübbecke, *Selected topics in column generation*, Operations Research **53** (2005), 1007–1023. 115

[DS08] P. Doganis and H. Sarimveis, *Optimal production scheduling for the dairy industry*, Annals of Operations Research **159** (2008), 315–331. 123

[DV14] C. Dürr and O. C. Vasquez, *Order constraints for single machine scheduling with non-linear cost*, Proceedings of the 16th Meeting on Algorithm Engineering & Experiments (ALENEX), SIAM, 2014, pp. 98–111. 46

[EGN08] B. Estellon, F. Gardi, and K. Nouioua, *Two local search approaches for solving real-life car sequencing problems*, European Journal of Operational Research **191** (2008), 928–944. 122, 123

[ELMS+12] L. Epstein, A. Levin, A. Marchetti-Spaccamela, N. Megow, J. Mestre, M. Skutella, and L. Stougie, *Universal sequencing on an unreliable machine*, SIAM Journal on Computing **41** (2012), 565–586. 9, 16

[FK84] M. L. Fisher and A. M. Krieger, *Analysis of a linearization heuristic for single-machine scheduling to maximize profit*, Mathematical Programming **28** (1984), 218–225. 16

[Fra75] A. Frank, *Some polynomial algorithms for certain graphs and hypergraphs*, Proceedings of the 5th British Combinatorial Conference, 1975, pp. 211–226. 98, 99

[GHM11] T. Gellert, W. Höhn, and R. H. Möhring, *Sequencing and scheduling for filling lines in dairy production*, Optimization Letters **5** (2011), 491–504, Special issue of SEA 2011. 121

[GJ79] M. R. Garey and D. S. Johnson, *Computer and intractability: A guide to the theory of NP-completeness*, Freeman, 1979. 11, 69, 129

[GLL82] U. I. Gupta, D. T. Lee, and J. Y.-T. Leung, *Efficient algorithms for interval graphs and circular arc graphs*, Networks **12** (1982), 459–467. 105

[GLLRK79] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan, *Optimization and approximation in deterministic sequencing and scheduling: A survey*, Annals of Discrete Mathematics **5** (1979), 287–326. 11, 12

[GP02] G. Gutin and A. P. Punnen, *The traveling salesman problem and its variations*, Springer, 2002. 77

[GS84] S. K. Gupta and T. Sen, *On the single machine scheduling problem with quadratic penalty function of completion times: An improved branching procedure*, Management Science **30** (1984), 644–647. 18, 45, 48

[GW95] A. Gyárfás and D. B. West, *Multitrack interval graphs*, Congressus Numerantium **109** (1995), 109–116. 96

[Hel00] K. Helsgaun, *An effective implementation of the Lin-Kernighan traveling salesman heuristic*, European Journal of Operational Research **126** (2000), 106–130. 77, 112, 131

[HJ12a] W. Höhn and T. Jacobs, *An experimental and analytical study of order constraints for single machine scheduling with quadratic cost*, Proceedings of the 14th Meeting on Algorithm Engineering & Experiments (ALENEX), SIAM, 2012, pp. 103–117. 45, 59

[HJ12b] ———, *On the performance of Smith's rule in single-machine scheduling with nonlinear cost*, Proceedings of the 10th Latin American Symposium on Theoretical Informatics (LATIN), LNCS, vol. 7256, Springer, 2012, pp. 482–493. 19, 45

[HK06] M. M. Halldórsson and R. K. Karlsson, *Strip graphs: Recognition and scheduling*, Proceedings of the 32th International Workshop on Graph Theoretic Concepts in Computer Science (WG), LNCS, vol. 4271, Springer, 2006, pp. 137–146. 99

[HKLM11] W. Höhn, F. G. König, M. E. Lübbecke, and R. H. Möhring, *Integrated sequencing and scheduling in coil coating*, Management Science **57** (2011), 647–666. 81, 95, 111

[HNR68] P. E. Hart, N. J. Nilsson, and B. Raphael, *A formal basis for the heuristic determination of minimum cost paths*, IEEE Transactions on Systems Science and Cybernetics **4** (1968), 100–107. 55, 56

[HPS00] N. G. Hall, C. N. Potts, and C. Sriskandarajah, *Parallel machine scheduling with a common server*, Discrete Applied Mathematics **102** (2000), 223–243. 77

[IMP12] S. Im, B. Moseley, and K. Pruhs, *Online scheduling with general cost function*, Proceedings of the 23rd Annual Symposium on Discrete Algorithms (SODA), SIAM, 2012, pp. 1254–1265. 9, 17, 19

[Jac12] T. Jacobs, *Analytical aspects of tie breaking*, Theoretical Computer Science **465** (2012), 1–9. 23

[Jan91] K. Jansen, *Generalizations of assignments of tasks with interval times*, Tech. report, Universität Trier, 1991. 99

[KD94] N. Kumar and N. Deo, *Multidimensional interval graphs*, Congressus Numerantium **102** (1994), 45–46. 96

[KK08] C. Koulamas and G. J. Kyparisis, *Single-machine scheduling problems with past-sequence-dependent setup times*, European Journal of Operational Research **187** (2008), 1045–1049. 77

[KKR01] H. Kaindl, G. Kainz, and K. Radda, *Asymmetry in search*, IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics **31** (2001), 791–796. 18, 55, 56, 60, 61, 62

[KP00] B. Kalyanasundaram and K. Pruhs, *Speed is as powerful as clairvoyance*, Journal of the ACM **47** (2000), 617–643. 12

[KPG10] G. M. Kopanos, L. Puigjaner, and M. C. Georgiadis, *Optimal production scheduling and lot-sizing in dairy plants: The yogurt production line*, Industrial & Engineering Chemical Research **49** (2010), 701–718. 74, 121, 123

[KPG12] ———, *Efficient mathematical frameworks for detailed production scheduling in food processing industries*, Computers & Chemical Engineering **42** (2012), 206–216. 74

[KTW99] H. Kellerer, T. Tautenhahn, and G. Woeginger, *Approximability and nonapproximability results for minimizing total flow time on a single machine*, SIAM Journal on Computing **28** (1999), 1155–1166. 15

[Law77] E. L. Lawler, *A "pseudopolynomial" algorithm for sequencing jobs to minimize total tardiness*, Annals of Discrete Mathematics **1** (1977), 331–342. 17

[LEGvB+05] M. Lütke Entrup, H.-O. Günther, P. van Beek, M. Grunow, and T. Seiler, *Mixed-integer linear programming approaches to shelf-life-integrated planning and scheduling in yoghurt production*, International Journal of Production Research **43** (2005), 5071–5100. 123

[LLLRK84] J. Labetoulle, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan, *Preemptive scheduling of uniform machines subject to release dates*, Progress in combinatorial optimization, Academic Press, Toronto, 1984, pp. 245–261. 15

[LRKB77] J. K. Lenstra, A. H. G. Rinnooy Kan, and P. Brucker, *Complexity of machine scheduling problems*, Annals of Discrete Mathematics **1** (1977), 343–362. 15

[MGSK88] H. Mühlenbein, M. Gorges-Schleuter, and O. Krämer, *Evolution algorithms in combinatorial optimization*, Parallel Computing **7** (1988), 65–85. 79

[MJ05] B. Meuthen and A.-S. Jandel, *Coil coating*, Vieweg, 2005. 82

[MNS07] F. Marinelli, M. Nenni, and A. Sforza, *Capacitated lot sizing and scheduling with parallel machines and shared buffers: A case study in a packaging company*, Annals of Operations Research **150** (2007), 177–192. 123

[MNT03] C. Meloni, D. Naso, and B. Turchiano, *Multi-objective evolutionary algorithms for a class of sequencing problems in manufacturing environments*, Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, 2003, pp. 8–13. 78

[Möh85] R. H. Möhring, *Algorithmic aspects of comparability graphs and interval graphs*, Graphs and Order, NATO ASI Series, vol. 147, Springer, 1985, pp. 41–101. 98, 113

[MPS13] B. Moseley, K. Pruhs, and C. Stein, *The complexity of scheduling for p-norms of flow and stretch*, Proceedings of the 16th Conference on Integer Programming and Combinatorial Optimization (IPCO), LNCS, vol. 7801, Springer, 2013, pp. 278–289. 16

[MS00] S. A. Mondal and A. K. Sen, *An improved precedence rule for single machine sequencing problems with quadratic penalty*, European Journal of Operational Research **125** (2000), 425–428. 2, 3, 10, 18, 45, 48, 49, 50, 55, 60, 62

[MV13] N. Megow and J. Verschae, *Dual techniques for scheduling on a machine with varying speed*, Proceedings of the 40th International Colloquium on Automata, Languages and Programming (ICALP), 2013, to appear. 9, 16, 69

[MV14] J. Mestre and J. Verschae, *A 4-approximation for scheduling on a single machine with general cost function*, 2014, http://www.sydney.edu.au/engineering/it/~mestre/papers/gen-scheduling-4-approx.pdf. 17

[PK86] B. D. Perrello and W. C. Kabat, *Job-shop scheduling using automated reasoning: A case study of the car-sequencing problem*, Journal of Automated Reasoning **2** (1986), 1–42. 74

[PK00] C. N. Potts and M. Y. Kovalyov, *Scheduling with batching: A review*, European Journal of Operational Research **120** (2000), 228–249. 123

[PY93] C. H. Papadimitriou and M. Yannakakis, *The traveling salesman problem with distances one and two*, Mathematics of Operations Research **18** (1993), 1–11. 77

[RDLD00] B. Rekieck, P. De Lit, and A. Delchambre, *Designing mixed-product assembly lines*, IEEE Transactions On Robotics And Automation **16** (2000), 268–280. 77

[Rot66] M. H. Rothkopf, *Scheduling independent tasks on parallel processors*, Management Science **12** (1966), 437–447. 14, 22

[RS84] M. H. Rothkopf and S. A. Smith, *There are no undiscovered priority index sequencing rules for minimizing total delay costs*, Operations Research **32** (1984), 451–456. 15, 45, 69

[SBR96] A. K. Sen, A. Bagchi, and R. Ramaswamy, *Searching graphs with A*: Applications to job sequencing*, IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans **26** (1996), 168–173. 18, 55, 56, 60

[Sch78] T. J. Schaefer, *The complexity of satisfiability problems*, Proceedings of the 10th Annual Symposium on Theory of Computing (STOC), ACM, 1978, pp. 216–226. 108

[Sch03] A. Schrijver, *Combinatorial optimization*, Springer, 2003. 11

[SCNA08] C. Solnon, V. D. Cung, A. Nguyen, and C. Artigues, *The car sequencing problem: Overview of state-of-the-art methods and industrial case-study of the ROADEF'2005 challenge problem*, European Journal of Operational Research **191** (2008), 912–927. 74

[SDR90] T. Sen, P. Dileepan, and B. Ruparel, *Minimizing a generalized quadratic penalty function of job completion times: An improved branch-and-bound approach*, Engineering Costs and Production Economics **18** (1990), 197–202. 18, 48, 49

[SF62] A. Schild and I. J. Fredman, *Scheduling tasks with deadlines and non-linear loss functions*, Management Science **9** (1962), 73–81. 17, 45, 48

[SG76] S. Sahni and T. F. Gonzalez, *P-complete approximation problems*, Journal of the ACM **23** (1976), 555–565. 77

[Smi56] W. E. Smith, *Various optimizers for single-stage production*, Naval Research Logistics Quarterly **3** (1956), 59–66. 9, 14, 19, 45

[Spi99] F. C. R. Spieksma, *On the approximabilty of an interval scheduling problem*, Journal of Scheduling **2** (1999), 215–227. 98, 99, 100, 101

[SW10] S. Stiller and A. Wiese, *Increasing speed scheduling and flow scheduling*, Proceedings of the 21st International Symposium on Algorithms and Computation (ISAAC), LNCS, vol. 6507, Springer, 2010, pp. 279–290. 9, 16, 19, 22, 45

[Szw98] W. Szwarc, *Decomposition in single-machine scheduling*, Annals of Operations Research **86** (1998), 271–287. 18, 49

[TJH79] W. T. Trotter Jr. and F. Harary, *On double and multiple interval graphs*, Journal of Graph Theory **3** (1979), 205–211. 96

[TLRY01] L. Tang, J. Liu, A. Rong, and Z. Yang, *A review of planning and scheduling systems and methods for integrated steel production*, European Journal of Operational Research **133** (2001), 1–20. 75

[Tow78] W. Townsend, *The single machine problem with quadratic penalty function of completion times: A branch-and-bound solution*, Management Science **24** (1978), 530–534. 17, 45, 55, 56

[TW09] L. Tang and X. Wang, *Simultaneously scheduling multiple turns for steel color-coating production*, European Journal of Operational Research **198** (2009), 715–725. 82

[vBMNW12] R. van Bevern, M. Mnich, R. Niedermeier, and M. Weller, *Interval scheduling and colorful independent sets*, Proceedings of the 23rd International Symposium on Algorithms and Computation (ISAAC), LNCS, vol. 7676, Springer, 2012, pp. 247–256. 98, 99, 135

[vDGS93] P. van Dam, G. Gaalman, and G. Sierksma, *Scheduling of packaging lines in the process industry: An empirical investigation*, International Journal of Production Economics **30–31** (1993), 579–589. 74

[Yua92] J. Yuan, *The NP-hardness of the single machine common due date weighted tardiness problem*, System Science and Mathematical Sciences **5** (1992), 328–333. 16, 45, 69

[ZSL09] W. Zhang, R. Smith, and C. Lowe, *Confocal raman microscopy study of the melamine distribution in polyester-melamine coil coating*, Journal of Coatings Technology and Research **6** (2009), 315–328. 81