Identical Parallel Machine Scheduling Problems: Structural patterns, bounding techniques and solution procedures

Dissertation

zur Erlangung des akademischen Grades
doctor rerum politicarum
(Dr. rer. pol.)

vorgelegt dem
Rat der Wirtschaftswissenschaftlichen Fakultät
der Friedrich-Schiller-Universität Jena

am: 02.11.2016

von: Dipl.-Math. oec. Alexander Lawrinenko

geboren am: 16.07.1986 in: Saalfeld/Saale

Gutachter

1. Professor Dr. Armin Scholl, Lehrstuhl für Allgemeine Betriebswirtschaftslehre und Management Science, FSU Jena

2. Professor Dr. Nils Boysen, Lehrstuhl für Allgemeine Betriebswirtschaftslehre und Operations Management, FSU Jena

Datum der Verteidigung: 24.03.2017

Acknowledgment

First, I would like to express my sincere thanks to my supervisor Prof. Dr. A. Scholl who gave me the unique opportunity to be a part of his team for almost five years. He was always a reliable contact person for important questions and gave me a lot of helpful advices for my work.

I am very grateful to my coauthor Dr. R. Walter who put me on the right path and from whom I have learned so much. His efforts were a great help for the completion of my doctoral thesis.

I am indebted to my second assessor Prof. Dr. N. Boysen who always had an open ear for questions and also gave helpful feedback in doctoral seminars.

My great thanks also go to my other coauthors S. Schwerdfeger and M. Wirth for the excellent collaboration. Especially the research discussions with S. Schwerdfeger were often a really helpful milestone for further steps in my work.

I acknowledge my gratitude also to my other colleagues at the Chair for Management Science for the great working atmosphere and the numerous fruitful discussions.

My very sincere thanks go to my mother, to my father, who sadly passed away, to my grandparents and to my uncles for their caring support.

Finally, I wish to express my deepest gratitude to my beloved wife Nino who was always patient with me and gave the energy to finalize my work. You are the reason why I wake up happy each morning.

Contents

List of Figures

List of Tables

Chapter 1

Introduction and overview

1.1 Introduction

1.1.1 Production process planning

Production planning systems are a core element of modern manufacturing companies. For the future production in a given planning horizon they address the following issues (cf. Domschke et al., 1997):

- production program planning, i.e. decisions on the product mix in line with the general corporate objectives,
- deployment of input factors such as raw materials or staff,
- production process planning, i.e. planning and control of the manufacturing process.

The different components can be subdivided into strategic, tactical, and operational production planning tasks in dependence on their time horizon. Especially long-term planning is a very complex scenario where experience values as well as stochastic forecasts are taken into account and an interdisciplinary collaboration for the process of decision making is required. These decisions might also have a big impact on downstream processes and therefore the sole application of mathematical methods might not be sufficient for satisfactory results. In contrast, operational plans can often be handled very well by the application of exact and heuristic solution procedures as the corresponding problems might be well-defined with a deterministic data base. For the productivity of the manufacturing system, scheduling plays a crucial role, where the efficient distribution of the workload is determined. Trends like mass customization ensure, that often a single-unit and small-series manufacturing of individual products has to be performed in the context of mass production. As the determination of the optimal workload distribution is not trivial in such cases, the application of mathematical methods becomes relevant.

In general, the workload distribution among different entities is a common problem that arises in various practical areas such as machine scheduling, parallel computing, bin packing, or often just as a sub-problem in a more complex environment. Consequently, the workload can be described by a set of items, tasks or jobs which have to be assigned to

entities such as bins, processors or machines. Although these topics do not look similar at first glance, they do all seek for a productive system with a uniform utilization and wear of all entities.

The variety of existing problems differs with respect to the configuration of the workload and the entities as well as the target that is aimed for. In some scenarios the number of required entities might be fixed (e.g. the number of machines in a manufacturing system), while in other ones the number of used entities shall be minimized (e.g. the number of bins in order to transport a set of certain goods). Simultaneously, it might be the goal to distribute the whole workload in a system or to maximize the amount of assignable workload because of certain restrictions, like time-constraints, which do not allow for an assignment of all workload pieces.
From a theoretical point of view, these prerequisites can be modeled in a mathematical sense by defining entities and workload pieces as fixed parameters, or as variables so that the number of entities or workload pieces is minimized or maximized in order to reach a certain goal.

1.1.2 Typical objectives for workload balancing

Turning back to production process planning, a well-known representative for workload distribution is given by the identical parallel machine scheduling problem (cf., e.g. McNaughton, 1959, Blazewicz, 1987, Lawler et al., 1993, Moktotoff, 2001). Despite its relatively simple problem inherent structure it is one of the most studied combinatorial optimization problems in the last 50 years with respect to heuristic solution approaches (cf., e.g. Graham, 1966, Coffman et al., 1978, Frangioni et al., 2004, Paletta and Ruiz-Torres, 2015) which are advantageous in cases with a limited planning time. But also for the time-consuming task of determining an optimal assignment of the workload, some exact solution procedures have been proposed (cf., e.g. Rothkopf, 1966, Dell'Amico and Martello, 1995, Dell'Amico et al., 2008, Walter and Lawrinenko, 2016).

For the identical parallel machine scheduling problem, the number of jobs n and machines m is fixed and one can think of several meaningful workload oriented objective functions that are connected to different areas for a practical applicability. The certainly most famous target is the minimization of the makespan, which is known as the $P||C_{max}$-problem in literature (cf. Graham et al., 1979). The makespan simply denotes the workload of the machine with the latest completion time $C_{max} = \max\{C_1, \ldots, C_i, \ldots C_m\}$ – where C_i denotes the completion time of machine i – and it is directly associated with the productivity of a manufacturing system, as a shorter production period allows for an earlier execution of follow-up tasks.

The counterpart of $P||C_{max}$ is given, when the goal is changed such that the workload of the least loaded machine shall be maximized. The so called machine covering problem $P||C_{min}$ (cf., e.g. Woeginger, 1997, Haouari and Jemmali, 2008a, Walter et al., 2016) is by far less studied but also has a lot of practical relevance. The problem was originally de-

scribed for spare part assignments (cf. Friesen and Deuermeyer, 1981) and another possible application is the fair allocation of investment projects to different geographical regions (cf. Haouari and Jemmali, 2008a). This clearly indicates, that scheduling problems are capable of handling a wide and diversified area of real world problems with a similar problem inherent structure.

For both presented criteria only the value of a single production entity is regarded in the objective function. This might lead to schedules where the workload is in general poorly balanced and therefore, e.g. an unequal wear of machines occurs. To account for this issue, specified workload balancing criteria can be applied instead. A straightforward criterion is given when the difference or the ratio between the C_{max} and C_{min} machines is minimized (cf. Karmarkar and Karp, 1982, Coffman and Langston, 1984) and therefore the completion time of all machines is implicitly considered. A more direct measure is given by the least squares method, where $\sum_{i=1}^{m} (C_i - \mu)^2$ is minimized and μ denotes the mean completion time $\mu = \sum_{i=1}^{m} C_i/m$ of all machines (cf., e.g. Chandra and Wong, 1975, Alon et al., 1998, Walter and Lawrinenko, 2014, Schwerdfeger and Walter, 2016). Although this problem variant typically requires a high computational effort for the determination of good quality solutions, it is certainly the most suited measure for workload balancing.

When the problem definition is changed such that the number of utilized entities with capacity C shall be minimized for the distribution of the entire workload, the well-known bin-packing problem is obtained. A profound literature overview on the topic is given in Delorme et al. (2016). The problem is also a dual of $P||C_{max}$. This basically means that bin packing instances can also be represented as equivalent $P||C_{max}$ instances, by transforming the bin capacity C into an equal maximal time span for the manufacturing process, where the task is to decide if m machines are sufficient for the processing of the entire workload. This connection can reciprocally be used to obtain stronger solution procedures and relaxations for both problems.
In the same manner, the bin covering problem is a dual of the machine covering problem $P||C_{min}$. Here, the goal is to maximize the number of covered entities with capacity C for the distribution of the entire workload. This problem variant typically arises when certain goods with individual weights, e.g. tomatoes on the vine, shall be packed in such a way that the number of packages with a weight of at least C is maximized. Taking all together, the mentioned topics in this chapter indicate for the variety of possible practical applications of rather simple structured problems.

1.1.3 Context of the doctoral thesis

Scheduling problems are one of the most studied fields in Operations Research. There is a multitude of problem variants that can be subdivided in dependence on the machine environment, job characteristics and the objective function. The doctoral thesis on hand mainly focuses on problems with identical parallel machines where no further restrictions, like release or due dates, for a possible assignment of jobs are given. Despite the large amount of publications for these problem types, there is surprisingly still a lot of potential

for further research studies. For some problem variants there is a lack of publications with respect to exact solution procedures. But also for the improvement of existing procedures, e.g. by the addition of further dominance criteria, there is still some potential. The analysis of problem inherent structures is also pretty promising, since the existing knowledge is mostly restricted to simple ideas, e.g. based on symmetrical reflections. Because of these insights, the definition of novel bounding techniques is also a substantial research field.

1.1.4 Contribution of the doctoral thesis

For the just mentioned topics several rather technical results are obtained in the doctoral thesis. We give insights to solution patterns and properties of several combinatorial optimization problems and show that despite the simple structure a lot information can be derived. The main result of the thesis is a characterization of potentially optimal schedules for $P||C_{max}$ as well as $P||C_{min}$ which works independently of the actual processing times of the jobs. The idea is to identify schedules that have the potential to become uniquely optimal when a certain setting of processing times is given. The approach is based on ideas similar to the concept of inverse optimization and also applicable for many other related problems, like workload balancing, bin packing or bin covering. We also show, that the approach can be transformed to dominance criteria that can e.g. be used in branch-and-bound algorithms. The advanced hardware technologies that came out in the recent years are also beneficial for the implementation of such complex structured ideas as an increased amount of information can be handled in a reasonable amount of time. Therefore, many parts of the work have a focus on the performance of enumeration approaches.

Furthermore, we propose several bounding techniques based on different methodological approaches. For example, new upper bounds for $P||C_{min}$ are derived, that are based on the solution of the corresponding bin covering problem. The analysis of solution structures and the incorporation of proven methods like lifting strategies (cf. Haouari et al., 2006) or column generation also lead to several new bounds for $P||C_{min}$, k_i-Partitioning and the minimum cardinality bin covering (MCBCP).

Based on the aforementioned analysis we are able to propose several effective solution methods which is clearly demonstrated by the computational results in the respective chapters. We also define some benchmark data sets that allow for a thorough and diversified analysis of practically relevant instances. Finally, a comparison between solution approaches for machine scheduling in the Operations Research and Artificial Intelligence communities is also a substantial part of the work, where the usage of the composite knowledge of both communities for future research projects is discussed.

As our methodological approaches are designed such that an increased efficiency of exact and heuristic solution methods shall be achieved, we see an impact on the solvability of practically relevant instances for workload distribution. However, as we derived parameter settings that are challenging in general, we think there is still a lot of potential for

further improvements by the definition of tailored solution methods. The given insights into structural patterns of potentially optimal solutions are also a potential prerequisite for further researches with respect to solution properties and a consequent design of novel algorithms. As we revealed some novel bounding techniques and solution methods based on the connection between similarly structured problems, a deepening examination of dualities might also be a fruitful approach for further research projects.

1.1.5 Structure of the thesis

The doctoral thesis consists of six papers which correspond to the six following chapters. In Chapter 2 a literature overview on the most relevant workload related criteria for identical parallel machine scheduling and number partitioning in general is given. In Chapter 3–5 three specific criteria for machine scheduling, namely $P||C_{max}, P||C_{min}, P||$NSSWD, are theoretically analyzed. In Chapter 6 a representative for number partitioning is also treated from a rather theoretical point of view. Finally, the work is concluded with a variant of bin-covering which is a dual of the $P||C_{min}$ problem with respect to bounding techniques (cf. Chapter 7) and a summary of the thesis as well as an outlook on possible future researches in Chapter 8.

Five of the six papers are currently submitted to peer-reviewed journals (see Table 1.1). Three articles have been accepted for publication or have already been published in three different journals, where two of them are rank A (according to the ranking VHB-JOURQUAL3 of "Verband der Hochschullehrer für Betriebswirtschaft e.V.") and one is rank B. The article "Effective solution space limitation for the identical parallel machine scheduling problem" is currently prepared to be resubmitted after an extensive reorganization of the paper content. The two remaining articles are both currently "submitted" and one of them is already "under-review".
The own contribution for each paper is subdivided into single-authored, leading ("federführend") and considerable ("maßgeblich"). For the determination of the own contribution a concrete breakdown for the six papers is displayed in Table 1.2 where information on the participation with respect to the following subjects are assessed:

- research/methodological concept, i.e. the determination of the research scope, which basically includes the definition of research questions and goals as well as basic ideas for their implementation,
- literature research/review, i.e. the classification of the problem and the validation of the currently existing literature on a certain topic,
- theoretical analysis and results, i.e. technical results for bounding, solution structures, worst-case behavior etc.,
- design/generation of solution methods, i.e. the concrete design of solution procedure components,
- programming, i.e. the efficient programmatic implementation,
- experimental studies, i.e. the selection of a suited data set, criteria for the evaluation as well as comparison with results from literature.

Authors	contribution	Journal	VHB
1. A survey on the identical parallel machine scheduling problem – Bounding techniques, approximation results, and solution approaches			
co-authors: none	single-authored	submitted – European Journal of Operational Research	A
2. Effective solution space limitation for the identical parallel machine scheduling problem			
co-authors: R. Walter	considerable	rework – Working Paper	–
3. Improved approaches to the exact solution of the machine covering problem			
co-authors: R. Walter, M. Wirth	considerable	Online first – Journal of Scheduling, DOI: 10.1007/s10951-016-0477-x	A
4. A note on minimizing the normalized sum of squared deviations on m parallel processors			
co-authors: R. Walter	considerable	published – Computers & Industrial Engineering 75, 257-259, 2014	B
5. Reduction criteria, upper bounds, and a dynamic programming based heuristic for the k_i-partitioning problem			
co-authors: S. Schwerdfeger, R. Walter	leading	under review – Journal of Scheduling	A
6. Lower bounds and algorithms for the minimum cardinality bin covering problem			
co-authors: R. Walter	considerable	published – European Journal of Operational Research 256, 392–403, 2017	A

Table 1.1: Overview of the accomplished research projects

paper	res./method. concept	literature res./review	theoretical analysis	solution methods	programming	experimental study
1	3	3	–	–	–	–
2	2	2	2	2	2	2
3	1	1	1	1	2	3
4	1	1	2	–	2	2
5	3	2	2	3	3	3
6	2	1	2	2	3	2

Table 1.2: Contributions to the papers of the doctoral thesis, broken down by topics: 1 – under proportional , 2 – proportional, 3 – over proportional

1.2 Overview on the six papers

1.2.1 The overview article

Title: A survey on the identical parallel machine scheduling problem – Bounding techniques, approximation results, and solution approaches

Motivation and research questions: Although there has been a considerable amount of literature for the identical parallel machine problem in the recent years, no survey on the topic has been given for almost two decades. Therefore, as an up-to-date overview is mandatory, literature for the famous $P||C_{max}$ problem as well as for closely related problems such as $P||C_{min}$, workload balancing or $P||\sum w_j C_j$ is reviewed.

In the Artificial Intelligence community the identical parallel machine problem is typically known as multi-way number partitioning. However, the relationship to literature of the Operations Research community is rather loose as Artificial Intelligence papers are only seldom mentioned there and vice versa. Because of that, we analyze the potential of the composite knowledge for potential future research projects.

Contribution: In the article, an overview of the most relevant literature for the identical parallel machine problem with respect to bounding techniques as well as heuristic and exact solution methods is given. In this context, we distinguish between Operations Research and Artificial Intelligence publications and give a characterization of the main contributions from the respective papers.

Results: Based on the proposed characterization of the literature, it can be seen that the scope and content of research projects changed within last decades. Until the early 90s publications are mostly related to simple structured heuristics and their worst-case behavior. Later, more sophisticated algorithms like meta-heuristics or branching approaches became popular. Also, the amount of researched problem variants increased over time and therefore a larger amount of real-world problems could be modeled. Finally, some research gaps were revealed, as we showed that for some problem variants there is a lack of substantial distributions. Even for well studied problem variants like $P||C_{max}$ the large amount of papers in the last years indicates for further research potentials.

1.2.2 The article on structural patterns of $P||C_{max}$ optimal solutions

Title: Effective solution space limitation for the identical parallel machine scheduling problem

Motivation and research questions: Despite the large amount of literature for the $P||C_{max}$ problem, there is not much knowledge about properties of optimal solutions. Because of that, it is not surprising that the existing exact solution procedures from literature (cf. Dell'Amico and Martello, 1995, Mokotoff, 2004, Dell'Amico et al., 2008, Haouari and

Jemmali, 2008b) are only capable of handling small- and medium-sized instances. The existing knowledge on optimal solutions mostly exploits information about symmetric reflections and the possible number of assignable jobs on each machine. Therefore, it is our goal to give new and enhanced insights into properties of optimal solution and to propose a competitive branch-and-bound algorithm.

Problem formulation: For a given set $M = \{M_1, \ldots, M_m\}$ of $m \geq 2$ identical parallel machines and a set $J = \{J_1, \ldots, J_n\}$ of $n > m$ independent jobs with positive processing times $t_1, t_2, \ldots, t_n$, the goal is to assign each job to exactly one machine such that the latest machine completion time $C_{max} = \max\{C_1, \ldots, C_i, \ldots, C_m\}$ – where C_i is the sum of processing times of jobs assigned to M_i – is minimized.

Contribution: Based on an approach similar to inverse optimization, we identify properties of potentially makespan optimal solutions that are also universally valid for other similar scheduling problems. The idea here is to identify schedules with certain structures, that have the potential to become (uniquely) makespan-optimal independently of the actual job processing times. The concept of potentially makespan-optimal schedules is at first described for the case of $m = 2$ machines and afterwards extended for an arbitrary number of machines. Then we show, that our analysis of potentially makespan optimal schedules allows for a definition of an effective dominance rule that can e.g. be used as an addon for existing branch-and-bound algorithms. The so-called "path-related" dominance rule is also extended by incorporating information about the maximum number of jobs on each machine as well as processing times of the jobs.

Results: In the computational study the effectiveness of the dominance rules is analyzed by an incorporation into a straightforward depth-first branch-and-bound algorithm with a strategy for heuristic diving. For a large and diversified set of instances the approach turns out to have a huge benefit in the limitation of the enumeration tree, especially when ratio n to m is not greater than 3. Therefore we gave a considerable contribution on the general solvability of $P||C_{max}$, since problem instances with a higher n to m are mostly easily solvable, even by simple heuristics.

1.2.3 The article on the exact solution of the $P||C_{min}$ problem

Title: Improved approaches to the exact solution of the machine covering problem

Motivation and research questions: In contrast to the substantial body of literature on $P||C_{max}$, the amount of studies on its counterpart $P||C_{min}$ is rather sparse. The few existing papers mostly deal with approximation algorithms and only one of them proposes an exact solution procedure (cf. Haouari and Jemmali, 2008a), which is only capable of solving small- to medium-sized instances. Hence, there is still an obvious need for further research projects on topics such as bounding techniques and exact solution procedures. Similar to $P||C_{max}$, there is also not much knowledge about structural pattern of optimal

solutions. So the question arises, if the concept of "path-related" dominance rules can also be successfully applied to the machine covering problem.

Problem formulation: For a given set $M = \{M_1, \ldots, M_m\}$ of $m \geq 2$ identical parallel machines and a set $J = \{J_1, \ldots, J_n\}$ of $n > m$ independent jobs with positive processing times $t_1, t_2, \ldots, t_n$, the goal is to assign each job to exactly one machine such that the earliest machine completion time $C_{min} = \min\{C_1, \ldots, C_m\}$ is maximized.

Contribution: We propose some novel bounding techniques based on two different ideas. Firstly, an upper bound that exploits the solution structure of $P||C_{min}$ and incorporates the machines with the minimum and maximum number of assignable jobs is proposed. Secondly, several upper bounds based on the closely connected bin covering problem are derived, where the goal is to cover as many bins with capacity C as possible by the assignment of n items. Also a bin packing based constructive heuristic is adapted in order to obtain an additional feasible lower bound for $P||C_{min}$. Furthermore, a branch-and-bound algorithm with a symmetry-breaking branching scheme and enhanced version of the "path-related" dominance rule is introduced. Its effectiveness is analyzed in a comprehensive computational study.

Results: We analyze the performance of our new bounding techniques and of the branch-and-bound algorithm on different benchmark data sets (cf. França et al., 1994, Frangioni et al., 2004). The new upper and lower bounds turn out to be quite successful, especially the bin covering based upper bounds often perform better than the bounds from the previous literature. Analogously to $P||C_{max}$, the performance of branch-and-bound strongly correlates with the effectiveness of the path-related dominance rules and strong results are obtained for rather small n to m ratios. Because of that, we see a huge benefit with respect to the solvability of practically relevant instances.

1.2.4 The note on a wrong result for workload balancing

Title: A note on minimizing the normalized sum of squared workload deviations on m parallel processors

Motivation and research questions: The problem of workload balancing is one of the most common tasks in scheduling theory. As a representative for workload balancing the normalized sum of squared workload deviations (NSSWD) criterion has been introduced by Ho et al. (2009). The authors proposed an algorithm that is based on the coherence of NSSWD to the makespan minimization problem $P||C_{max}$. However, their assumptions are questionable.

Problem formulation: For a given set $M = \{M_1, \ldots, M_m\}$ of $m \geq 2$ identical parallel machines and a set $J = \{J_1, \ldots, J_n\}$ of $n > m$ independent jobs with positive processing times $t_1, t_2, \ldots, t_n$, the goal is to assign each job to exactly one machine such that $\frac{1}{\mu}\sqrt{\sum_{i=1}^{m}(C_i - \mu)^2}$ is minimized with $\mu = \sum_{j=1}^{n} t_j/m$. Consequently, the problem repre-

sents a normalized variant of the least squares method.

Contribution: We provide a counter-example for $m \geq 2$ on a wrong result in Ho et al. (2009) who claimed that a NSSWD optimal schedule is necessarily a $P||C_{max}$ optimal schedule as well. We also explain the incorrectness of their proof. In a computational study we give an overview on $P||C_{max}$ as well as NSSWD optimal schedules and analyze empirically the correlation between both criteria.

Results: Our results reveal that the $P||C_{max}$ criterion is a good approximation for NSSWD when the ratio n to m is rather small. However, there are also some parameter settings where a $P||C_{max}$ optimal solution is not a good candidate solution for workload balancing. Consequently, solution approaches that are based on the $P||C_{max}$ criterion are not necessarily suited for NSSWD as well. However as our computational results reveal, NSSWD optimal solutions have generally a very good performance for the $P||C_{max}$ criterion.

1.2.5 The article on technical results for the k_i-partitioning problem

Title: Reduction criteria, upper bounds, and a dynamic programming based heuristic for the k_i-partitioning problem

Motivation and research questions: Partitioning problems are a widely studied research topic, especially in the Artificial Intelligence community. However, for the k_i-partitioning problem with respect to machine covering there has only been one publication until now (cf. He et al., 2003), where solely an approximation algorithm is proposed. However, as the problem has practical relevance (e.g. for flexible manufacturing systems or the fair allocation of investment projects to different regions) and can be utilized because of its duality to makespan-related partitioning variants, some additional theoretical insights seem to be interesting. Also, no specific techniques for upper bounding have been proposed yet. Finally, as there is no established test bed for the problem we recognize the need for a suited benchmark data set.

Problem formulation: For a given set $M = \{M_1, \ldots, M_m\}$ of $m \geq 2$ identical parallel machines with associated machine-dependent cardinality limits k_i and a set $J = \{J_1, \ldots, J_n\}$ of $n > m$ independent jobs with positive processing times $t_1, t_2, \ldots, t_n$ the goal is to assign each job to exactly one machine such that $C_{min} = \min\{C_1, \ldots, C_m\}$ is maximized and the number of assigned jobs to each machine M_i is not greater than k_i.

Contribution: Several preprocessing methods are introduced, that are capable of reducing the solution space by tightening the cardinality limits and removing jobs as well as machines without changing the value of the optimal solution. Besides the definition of new upper bounds we also extend the concept of lifting which allows for the calculation of upper bounds on a subset of jobs and machines that are also valid for the entire instance. For the approximate solution we define two modified LPT variants and a subset-sum based

improvement heuristic based on an efficient dynamic programming approach.

Results: We analyze the performance of our preprocessing methods, upper bounds and heuristic procedures in a computational study on a novel set of instances for 2106 different parameter settings. The preprocessing methods helped to reduce the solution space for around 12% of the instances, and the enhanced lifting concept was able to tighten the best known upper bound for over 60% of the relevant instances. In general, the results of our computational tests attest for the efficiency of our proposed methods and we were able to simplify most of the problem instances. Overall, we were able to verify an optimal solution for around 70% of the instances with an average computation time of less than one second. Because of the fact, that the machine covering problem is a "dual" of $P||C_{max}$, we see the potential for a possible applicability of the technical results to other similar problems.

1.2.6 The article on solution approaches for the minimum cardinality bin covering problem

Title: Lower bounds and algorithms for the minimum cardinality bin covering problem

Motivation and research questions: The so called minimum cardinality bin covering problem (MCBCP) is a natural variant of the bin covering problem and has received only little attention in literature yet, although it has some connections to real world problems. As a possible application, we think of the transportation or disposal of m different liquids that cannot be mixed, which is known as the liquid loading problem (cf. Christofides et al., 1979). If C volume units of each liquid shall be transported, the goal is to use the fewest possible number of available tanks with an individual size w_j. The MCBCP belongs to the class of mixed integer packing covering problems and can be formulated as a covering integer problem with generalized multiplicity constraints. For this class of problems only a few approximation algorithms have been published which cannot always guarantee a feasible solution. Due to this reason, there is an obvious need for suited heuristic and exact solution procedures. To assess the quality of the solution procedures the definition of strong lower bound techniques is an important task that is considered in the paper.

Problem formulation: For a given set $B = \{B_1, \ldots, B_m\}$ of $m \geq 2$ bins with identical capacity C and a set $J = \{J_1, \ldots, J_n\}$ of $n > m$ independent items with weights $w_1, w_2, \ldots, w_n$, the goal is to cover all m bins by an assignment of the fewest possible number of items.

Contribution: Besides the classification of the problem with respect to the existing literature, also some insights to structural patterns are given. We propose several lower bounds, e.g. based on column generation or the solution structure of MCBCP. For approximate solutions we propose several constructive heuristics as well as an improvement heuristic based on the iterative solution of subset sum problems. Also, a depth-first branch-and-bound algorithm with straightforward dominance criteria and local bounding is developed. Finally, we evaluate the performance of our solution approaches by a comparison with the

results of the commercial solver Gurobi in a comprehensive computational study on a diversified data set.

Results: We derive some results on worst-case performances of our lower bounds. The combined application of our heuristics and the elaborate lower bounding techniques turned out to be quite successful as we were able to verify an optimal solution for about 96% of the instances. In general, our approach significantly outperforms Gurobi for all regarded parameter settings with respect to the computation time and the number of optimally solved instances. Based on our computational study we can conclude, that hard instances mostly occur, when the bin capacity is rather small compared to the average item size and just a few items can be assigned to each bin on average. As we are the first who deal with exact solution approaches for the MCBCP, we see a huge impact on the solvability of practically relevant instances. *Results:* We derive some results on worst-case performances of our lower bounds. The combined application of our heuristics and the elaborate lower bounding techniques turned out to be quite successful as we were able to verify an optimal solution for about 96% of the instances. In general, our approach significantly outperforms Gurobi for all regarded parameter settings with respect to the computation time and the number of optimally solved instances. Based on our computational study we can conclude, that hard instances mostly occur, when the bin capacity is rather small compared to the average item size and just a few items can be assigned to each bin on average. As we are the first who deal with exact solution approaches for the MCBCP, we see a huge impact on the solvability of practically relevant instances.

Chapter 2

A survey on the identical parallel machine scheduling problem – Bounding techniques, approximation results, and solution approaches

Summary

Meanwhile, almost two decades elapsed since the last review dealing with the problem of identical parallel machine scheduling has been published. Therefore we provide an up-to-date survey on the most relevant literature. Besides the famous $P||C_{max}$ problem we also review literature for closely related problems such as $P||C_{min}$, workload balancing or $P||\sum w_j C_j$ and present their main contribution. We also give an insight into the relevant literature contributed by the Artificial Intelligence community, where the problem is typically known as number partitioning.

2.1 Introduction

In this Chapter we consider the identical parallel machine scheduling problem where a given set $J = \{J_1, \ldots, J_n\}$ of n independent jobs with processing times $t_1, t_2, \ldots, t_n$ has to be assigned to a set $M = \{M_1, \ldots, M_m\}$ of $m \geq 2$ identical parallel machines so that a given objective function has to be optimized without further restrictions regarding machine as well as job availabilities. Due to the structure of the identical machines only completion time related measures are meaningful.
The certainly most famous measure in the offline case, is the makespan minimization with the goal of determining a schedule where the maximum completion time $C_{max} = \max\{C_1, \ldots, C_i, \ldots, C_m\}$ of all machines has to be minimized and where C_i denotes the completion time of machine i. Another closely related problem arises when the objective is changed so that the minimal completion time $C_{min} = \min\{C_1, \ldots, C_m\}$ of all machines is maximized. Using the three-field notation of Graham et al. (1979) the problems can

also be denoted by $P||C_{max}$ and $P||C_{min}$ resp. which are well-known abbreviations in literature.

A lot of technical results have been published for the identical parallel machine problem in general, but also connections to real-world problems have been revealed. For $P||C_{max}$ we think of production lines where several machines with the same speed have to perform a certain amount of jobs (cf. Mokotoff, 2001), as well as multiprocessor computers where tasks should be assigned efficiently in order to minimize the overall cpu-time. The machine covering problem $P||C_{min}$ was originally described in the context of spare part assignments (cf. Friesen and Deuermeyer, 1981) and a further possible application is given by the fair regional allocation of investments (cf. Haouari and Jemmali, 2008a).

Since the spread of the machine completion times is limited by the two aforementioned problems, they are already related to workload balancing in a wider sense. However, there are also more specified measures. Two further objective functions consider the machines with the highest and lowest workload and aim for a minimization of $C_\Delta = C_{max} - C_{min}$ and C_{max}/C_{min} resp. To balance the workload between the whole set of machines, a family of criteria is given by the minimization of the p-norm of the completion times, i.e. $\mathcal{L}_p = (\sum_{i=1}^{m} C_i^p)^{1/p}$. As a representative for $p = 2$, Ho et al. (2009) proposed the minimization of the normalized sum of squared workload deviations (NSSWD). The problem of balancing out a schedule occurs when tasks should be distributed among workers in order to avoid unequal treatments (cf. Cossari et al., 2013) and in manufacturing industries where a balanced schedule is important to reduce idle times and work-in-progress (cf. Ouazene et al., 2014).

For all of the above-mentioned objective functions the constraint system is equivalent, since the universal task is to schedule all available jobs among the m machines without regarding their sequence. The resulting constraint system is given by

$$\sum_{j=1}^{n} t_j x_{ij} \geq C_i \qquad i = 1, \ldots, m \tag{2.1}$$

$$\sum_{i=1}^{m} x_{ij} \leq 1 \qquad j = 1, \ldots, n \tag{2.2}$$

$$x_{ij} \in \{0, 1\} \qquad i = 1, \ldots, m;\ j = 1, \ldots, n \tag{2.3}$$

$$C_i \geq 0 \qquad i = 1, \ldots, m \tag{2.4}$$

where (2.1) sets the workload C_i of each machine, (2.2) ensures that each job j is scheduled to exactly one machine i and (2.3), (2.4) set the domains of the variables.

Some other criteria that are based on the sequencing of the jobs are given by the minimization of the total weighted completion time $P||\sum w_j C_j$ as well as the minimization of the job completion times $P||\sum C_j$ and the squared job completion times $P||\sum C_j^2$. Here, the order of the jobs on each machine plays a crucial role for the determination of the solution quality and therefore also a different constraint system is required to obtain the sequencing on the machines. For a practical application of these problems, the optimization of queues can be named.

Although the different criteria already cover a broadly diversified area of real-world applications, they might also occur as a subproblem in more complex structured problems.

For example, Tang et al. (2006) investigated a hybrid flowshop problem where a relaxed problem with Lagrangian multipliers equals identical parallel machine scheduling.
The theoretical foundation for identical parallel machine scheduling is given through the topic of (multi-way) number partitioning, where the goal is to partition a set $\{a_1, a_2, \ldots, a_n\}$ of positive integers into m different sets $\{\mathcal{A}_1, \ldots, \mathcal{A}_m\}$ with respect to a certain objective function. While parallel machine scheduling is the commonly used term in Operations Research (OR) literature, number partitioning is more related to the Artificial Intelligence (AI) community. Nevertheless, as both problems are equivalent, an overview on the combined literature is meaningful.

The remainder of the Chapter is built up as follows. In Section 2.2, an in-depth literature review on the different objective functions for identical parallel machine scheduling is given. In Section 2.3, we also give a literature review on number partitioning. Finally, in Section 2.4, a summary of the Chapter content is given.

To evaluate the performance of bounds as well as heuristics, the worst-case performance ratio is a commonly used method. In dependence on the objective function (minimization or maximization) we distinguish between two different cases: For a minimization problem γ_1, let $OPT(I)$ denote the optimal solution for an instance I and $A(I)$ the solution when an arbitrary approximation algorithm A is used. Then, the worst-case performance ratio is defined as the smallest real number $\overline{R}_{\gamma_1}(A) \geq 1$ such that $\overline{R}_{\gamma_1}(A) \leq A(I)/OPT(I)$ holds for all possible instances I. For a lower bound L, analogously $\underline{R}_{\gamma_1}(A) \geq L(I)/OPT(I)$ is defined with $\underline{R}_{\gamma_1}(A) \leq 1$. For a maximization problem γ_2, the definition is basically identical, however $\overline{R}_{\gamma_2} \leq 1$ and $\underline{R}_{\gamma_2} \geq 1$ hold. Formally, we have

$$\overline{R}_{\gamma}(A) = \sup_{I} \{A(I)/OPT(I)\} \tag{2.5}$$

$$\underline{R}_{\gamma}(A) = \inf_{I} \{L(I)/OPT(I)\} \tag{2.6}$$

2.2 Identical parallel machine scheduling

When the case of two machines is considered, the minimization of C_{max} simultaneously leads to a maximization of C_{min} and therefore also to a minimization of the difference between C_{max} and C_{min} as well as the ratio of the workload of both machines. Because of that, these objective functions are equivalent for $m = 2$. Also the subset sum problem (SSP) is an equivalent problem formulation for $m = 2$. Here, the goal is to identify a subset $\mathcal{A}_1$ such that the sum of the processing times is as close to $\sum_{j=1}^{n} t_j/2$ as possible. Therefore, since Martello and Toth (1984) showed that SSP is solvable in pseudo-polynomial time, the same result holds for the aforementioned machine scheduling problems. However, even for $m = 2$, literature for these criteria is well justified, since, e.g. worst-case and probabilistic results obviously still do very well depend on the underlying measure. Based on the two machine case it is readily verified that machine completion time related problems (C_{max}, C_{min} and workload balancing) are NP-hard in the strong sense by a straightforward reduction from partition (cf., Garey and Johnson, 1979). For the generalized case $m > 2$ the above-mentioned criteria are not longer equivalent.

2.2.1 Makespan minimization

In this section an in-depth literature review on the topic of makespan minimization without further restrictive job characteristics is given. As already mentioned, $P||C_{max}$ is by far the most studied among all completion time related criteria. The problem was originally introduced by McNaughton (1959) and since then a substantial body of literature has been released, which is mainly concerned with approximation algorithms, bounding techniques and heuristics but also a few publications with respect to exact solution procedures can be found. Some surveys were also published on the topic of parallel machine scheduling problems (cf. Blazewicz , 1987, Cheng and Sin, 1990, Lawler et al. 1993, and Moktotoff, 2001). However, as the last survey is more than 15 years old and a lot of new solution approaches have been proposed since then, an up-to-date report on the latest developments will be given here.

Some of the presented works utilize the coherence between $P||C_{max}$ and the bin packing problem where each job j is transformed into an item of size t_j and it is checked if the set of items can be packed into at most m bins of size C. If this is the case, a feasible schedule for $P||C_{max}$ with $C_{max} = C$ can be derived. The coherence can also be used for the definition of lower and upper bounds. For a detailed literature overview on bin packing and cutting stock we refer to Delorme et al. (2016).

2.2.1.1 Lower bound strategies

The first simple bounds L_0 and L_1 for the $P||C_{max}$ problem were originally described by McNaughton (1959). The idea behind $L_0 = \sum_{j=1}^{n} t_j/m$ is based on the optimal solution of the LP-relaxation where the indivisibility property of the jobs is neglected. He also proposed an improved version $L_1 = \max\{L_0, \max_j\{t_j\}\}$ regarding the fact that the longest job must be assigned to exactly one machine. Later, Dell'Amico and Martello (1995) proved the worst-case behavior $\underline{R}_{C_{max}}(L_0) = 0$ and $\underline{R}_{C_{max}}(L_1) = \frac{1}{2}$ for the respective bounds and also described an enhanced variant $L_2 = \max\{L_1, t_m + t_{m+1}\}$ with $\underline{R}_{C_{max}}(L_2) = \frac{2}{3}$. The idea here is to consider a partial instance where the $n - m - 1$ smallest jobs are eliminated and it can easily be seen that the makespan is at least as large as $t_m + t_{m+1}$. Dell'Amico and Martello (1995) also developed some bounds based on the solution structure by realizing that there is at least one machine that processes $v = \lceil \frac{n}{m} \rceil$ or more jobs, leading to $L_v = \sum_{j=n-v+1}^{n} t_j$. Furthermore, another bound L_ν is based on the minimum and maximum number of jobs that can be assigned to each machine. Because of its complex structure, we refer to Dell'Amico and Martello (1995) for a detailed description of the bound.

Hochbaum and Shmoys (1987) were the first who used the duality to the bin packing problem for the construction of a lower bound by defining $L_{HS} = \max\{L_1, \max\{L+1 : B_\gamma(L) > m\}\}$ where $B_\gamma(L)$ is the number of bins with capacity $C = L$ used by an approximation algorithm for jobs with $t_j > \frac{L}{5}$. Dell'Amico and Martello (1995) also proposed a bin packing oriented lower bound L_3 with time complexity $\mathcal{O}(n^2 \log U)$, where U is an upper bound and the jobs are characterized with respect to their size in relationship to the bin capacity C and pairwise incompatible jobs can be identified as a consequence.

The most recent publications for lower bounds stem from Haouari et al. (2006) as well as Haouari and Jemmali (2008b). Here, a lifting procedure is presented where bounds are

calculated for specific partial instances J_n^k with the $\lambda_k(n) = k \left\lfloor \frac{n}{m} \right\rfloor + \min\left\{k, n - \left\lfloor \frac{n}{m} \right\rfloor m\right\}$ largest jobs and k machines that are also valid for the entire instance. The determination of the partial instances is based on the idea, that there exists a subset of k machines that have to carry out at least $\lambda_k(n)$ jobs. Haouari and Jemmali (2008b) also introduced a procedure that tries to tighten a lower bound L by solving a specific subset sum-problem.

2.2.1.2 Constructive heuristics

Most of the early publications are related to constructive heuristics and their worst-case behavior. Graham (1966) analyzed the worst-case performance of List Scheduling (LS), an algorithm where for an arbitrarily sorted list of jobs each of it is successively assigned to the machine with the current lowest completion time, and proved that $\overline{R}_{C_{max}}(\text{LS}) = 2 - \frac{1}{m}$ holds. Later, Graham (1969) proved the worst-case performance of $\overline{R}_{C_{max}}(\text{LPT}) = \frac{4}{3} - \frac{1}{3m}$ for the well-known LPT rule which is a specified version of LS, where the jobs are sorted by non-increasing processing times. LS has a time complexity of $\mathcal{O}(n \log m)$, while for the LPT rule the time complexity is $\mathcal{O}(n \log n + n \log m)$ in case of an arbitrarily sorted initial job list. Another worst-case result for LPT was presented by Bruno et al. (1974) with $\overline{R}_{C_{max}}(\text{LPT}) = 1 + \frac{m-1}{mk}$ where k denotes the number of jobs on the makespan machine. Besides the analysis of worst-case results for LS and LPT, the probabilistic study of average-case results gained some attention. Coffman and Gilbert (1985) analyzed the expected competitive ratio $\frac{C_{max}^{LS}}{C_{max}^*}$ of list scheduling for the case, that the processing times are independently drawn from a uniform distribution in $[0, 1]$. With respect to LPT, Frenk and Rinnooy Kan (1986, 1987) proposed results for the asymptotic behavior of the heuristic. They showed that under mild conditions on the probability distribution, the absolute error $C_{max}^{\text{LPT}} - C_{max}^*$ converges to 0 almost surely and also gave results for the speed of convergence for uniformly and exponentially distributed processing times.

Coffman et al. (1978) proposed another constructive heuristic, the so called Multifit (MF) algorithm with time complexity $\mathcal{O}(n \log n + kn \log m)$. Here, through a binary search procedure with k iterations the smallest value for a fixed makespan C is determined such that the application of the bin packing algorithm First Fit Decreasing (FFD) generates a solution not greater than m. They also proved a worst-case performance of $\frac{8}{7}$ for $m = 2$, $\frac{15}{13}$ for $m = 3$ and $\frac{20}{17}$ for $m = 4, \ldots, 7$ and a general performance in dependence on the number of binary-search iterations k with $\overline{R}_{C_{max}}(\text{MF}) = 1.22\text{OPT} + \frac{1}{2^k}$. Friesen (1984) tightened the bound by showing that $\overline{R}_{C_{max}}(\text{MF}) = 1.2\text{OPT} + \frac{1}{2^k}$ holds and also introduced a family of instances for $m \geq 13$ where a ratio of $\frac{13}{11}$ was established. Later, Yue (1990) proved that $\overline{R}_{C_{max}}(\text{MF}) = \frac{13}{11}$ indeed holds for an arbitrary m.

A modified version of Multifit, denoted as MFI, with the same time complexity was introduced by Friesen and Langston (1986) where FFD is refined by splitting the job list, in the case when it is not able to find a feasible solution. They also showed, that an improved worst-case performance of $\overline{R}_{C_{max}}(\text{MFI}) = \frac{72}{61}$ can be achieved, which is tight for $m \geq 12$. Although Multifit got a better worst-case performance compared to LPT, the average behavior of the two algorithms is often vice versa. Motivated by this, Lee and Massey (1988) designed the so called Combine method which takes the LPT schedule as an input for the Multifit binary-search procedure, and still runs in $\mathcal{O}(n \log n + kn \log m)$. For

$m = 2$ they proved a worst-case performance of $\frac{10}{9}$ which is strictly better than the ones of Multifit and LPT. Another similar composite algorithm is due to Gupta and Ruiz-Torres (2001) who proposed the so called Listfit algorithm. Here the Multifit approach is applied to a combination of sub-lists that are gained from different LS schedules. The algorithm has a time complexity of $\mathcal{O}(n^2 \log n + n^2 k \log m)$ and a worst-case ratio equal to Multifit. However, a superior average performance was shown in an empirical evaluation.

Another constructive heuristic based on a different approach is due to Paletta and Pietramala (2007). In their MPS algorithm, at first an initial set of disjoint partial solutions is constructed by a partition of the job list into z families of subsets with certain properties. Afterwards these partial solutions are iteratively combined until a feasible solution is obtained. The algorithm has a time complexity of $\mathcal{O}(n \log n + nm)$ and a worst-case performance of $\overline{R}_{C_{max}}(\text{MPS}) = \mathcal{O}(\frac{z+1}{z} - \frac{1}{mz})$. By changing the procedures that are used to build up the partial solutions as well as to combine them, modified versions of MPS were presented by Gualtieri et al. (2008), Gualtieri et al. (2009) and Chiaselotti et al. (2010), each of them runs in $\mathcal{O}(n \log n)$. The latest publication concerning constructive heuristics stems from Paletta and Ruiz-Torres (2015) and consists of two procedures. At first, a modified version of MPS is applied to construct a feasible solution. Afterwards the so called Many Times Multifit (MTMF) procedure attempts to tighten the initial solution by iteratively using a bin packing based procedure on different job sets. Their Partial Solutions and Multifit (PSMF) algorithm has a time complexity of $\mathcal{O}(tn^2)$ – where t denotes the number of iterations in the MTMF procedure – and a worst-case performance equal to Multifit.

2.2.1.3 Improvement heuristics and local search approaches

In addition to the long list of constructive heuristics there are also many publications related to improvement methods based on local search techniques that mostly differ in the definition of their neighborhoods and the determination of initial solutions. Finn and Horowitz (1979) proposed a simple job exchange algorithm (IC) with time complexity $\mathcal{O}(n \log m)$ and $\overline{R}_{C_{max}}(\text{IC}) = 2 - \frac{2}{m+1}$, where the goal is to even out the machines with the highest and lowest completion time of a randomly generated schedule by swapping jobs, until no further improvements can be achieved. Langston (1982) improved the algorithm by using the LPT rule for the assignment of the $2m$ longest jobs in the initial schedule and also proved that the worst-case performance of his algorithm ICII is equal to the one of LPT. In França et al. (1994) a 3-phase algorithm consisting of a construction phase and two improvement phases with job reassignments and interchanges was introduced. Additionally, the authors designed a set of benchmark instances with uniform distributed processing times and showed, that their approach outperforms the ones of Finn and Horrowitz (1979) as well as Langston (1982).

Ho and Wong (1995) proposed a method, where for several machine pairs the corresponding $P2||C_{max}$ sub-problems are solved by a lexicographic search algorithm on a binary search tree, in order to even out the machine pairs. In Riera et al. (1996) two further algorithms were introduced, where the LPT schedule is improved by application of several job interchange procedures. In their experimental study they show, that their al-

gorithms are in general superior to Multifit. Fatemi-Ghomi and Jolai-Ghazvini (1998) also published a local search algorithm. Here, the goal is to minimize the gap $|C_{i_1} - C_{i_2}|$ for different machine pairs (i_1, i_2) by exchanging jobs as long as improvements are realizable. The approach is applied for four initial schedules that are obtained by the application of different List Scheduling rules.

A more sophisticated local search idea with larger neighborhood definitions was described by Frangioni et al. (2004). Here, job allocations are cyclically changed between r machines on a basis of an improvement graph. The concept is applied for different families of heuristics like label correcting and bottleneck path. To evaluate the potential of their heuristics, the authors also introduced further benchmark instances which are besides the ones of França et al. (1994) the default data sets even today. Alvim and Ribeiro (2004) described a bin packing oriented algorithm, where after a construction phase a binary search procedure is applied, including a redistribution step and an improvement step in order to construct a feasible solution for a prefixed C value. A further iterated local search algorithm where an approximate dynamic programming method is used to find cyclic exchanges between several machines was described by Tang and Luo (2006). The procedure is based on the concept of Frangioni et al. (2004) but differs in the definition of the cyclic exchange neighborhoods.

A method that is based on the exact solution of the pseudo-polynomial two-machine case was proposed by Haouari et al. (2006). More precisely, their Multi-Start Subset Sum (MSS) heuristic generates multiple randomized LPT schedules and tries to improve each of them by solving the $P2||C_{max}$ problem for different machine pairs as long as a reduced makespan is realizable. The procedure is closely related to the one introduced in Ho and Wong (1995), but differs with respect to the time complexity for the solution of $P2||C_{max}$ and the number of initial schedules. Another heuristic that is based on the combination of partial solutions was proposed by Paletta and Vocaturo (2011). After constructing an initial solution in a similar manner to Paletta and Pietramala (2007) they use local search techniques where single jobs or sets of jobs are changed between different machine pairs (i_1, i_2).

2.2.1.4 Meta heuristics

In contrast to the long history of constructive as well as improvement heuristics, the tendency for an application of meta approaches became just popular in the recent two decades. Hübscher and Glover (1994) were the first who proposed a tabu search algorithm with respect to makespan minimization. Their algorithm is based on an efficient candidate list for local moves, a dynamic tabu list to allow for both intensification and diversification, and an influential diversification scheme that reassigns a small amount of large jobs when the solution quality is not sufficient for a certain amount of iterations. A few years later, Thesen (1998) studied the design of tabu search for multiprocessor scheduling by analyzing the composition of different elements such as the tabu list, local search techniques as well as list management strategies. Especially a random blocking of the tabu list and greedy local search methods turned out to be effective. As a representative for evolutionary algorithms Min and Cheng (1999) applied a genetic algorithm to $P||C_{max}$ by defining a fitness function based on a transformed objective function, two-point crossovers

and one-digit mutations. Later, Lee et al. (2006) presented a straightforward simulated annealing approach, where the initial schedule is obtained by application of the LPT rule and solutions are transformed by job interchanges between the makespan machine and the remaining machines. Here, the acceptance probability for new solutions depends on a temperature parameter and on the gap between the previous and current makespan.

The latest publications with respect to meta heuristics for $P||C_{max}$ are based on evolutionary approaches. Dell'Amico et al. (2008) presented a scatter search algorithm where a reference set consisting of high quality and diversified solutions is generated and afterwards subsets of these solutions are iteratively combined in order to generate new solutions. Also several improvement methods based on local search techniques are used to tighten the combined solutions. Kashan and Karimi (2009) proposed a particle swarm optimization algorithm where possible solutions are represented as particles and multiple new operators for their velocities and positions are introduced. They also hybridize their algorithm with a local search algorithm to allow for a faster convergence. A hybrid dynamic harmony search algorithm was presented in Chen et al. (2012) where an encoding scheme based on list scheduling is given that converts the continuous harmonies to integer assignments. The harmony memory is partitioned into sub-harmonies which are capable of exchanging information with each other. Also improvement processes as well as local search techniques are applied to guide the algorithm. Finally, Davidović et al. (2012) described a bee colony optimization algorithm, which is a representative of stochastic swarm optimization and is based on the natural behavior of bees. The algorithm simultaneously builds up several solutions by a stochastic procedure and dismisses bad partial solutions with a certain probability in dependence on their quality.

2.2.1.5 Exact solution procedures

The first publication related to exact solution procedures stems from Rothkopf (1966) who showed that the problem is exactly solvable via dynamic programming in $\mathcal{O}(nU^m)$ where U is an upper bound on the optimal solution. Therefore the algorithm is only capable of solving instances with a rather small number of machines. The first branch-and-bound algorithm for $P||C_{max}$ was introduced by Dell'Amico and Martello (1995) who used a depth-first enumeration tree and also included some novel dominance criteria as well as new lower and upper bounding techniques that are used for global and local bounding. Later, Mokotoff (2004) presented a cutting plane algorithm where valid inequalities are identified and iteratively added to the ILP model until the solution of the corresponding LP-relaxation is integer. However, Dell'Amico and Martello (2005) showed that their branch-and-bound algorithm clearly outperforms the approach of Mokotoff (2004) on a diversified set of instances. More recent publications for the exact solution stem from Haouari and Jemmali (2008b), who embedded lifting based lower bounds (cf. Sect. 2.2.1.1) into a new symmetry-breaking depth-first branching scheme and from Dell'Amico et al. (2008), where a combination of a bin packing oriented binary-search and branch-and-price is used. The latest approach stems from Walter and Lawrinenko (2016), who presented an in-depth analysis about structural patterns of potentially optimal solutions based on a concept that is similar to inverse optimization and derived symmetry-breaking dominance criteria for a depth-first branch-and-bound algorithm. In most of the publications for the exact solution of $P||C_{max}$, especially instances with a ratio $\frac{n}{m} \in [2, 4]$ turned out to be quite

difficult to solve with respect to computation times, while instances with a higher ratio n to m can often already be solved by simple global bounding procedures.

2.2.2 Machine covering

In contrast to the substantial body of publications on $P||C_{max}$ the literature for $P||C_{min}$ is rather sparse. As already mentioned the problem was first described by Friesen and Deuermeyer (1981) and its theoretical aspects were first analyzed by Deuermeyer et al. (1982) who showed that the LPT rule has a worst-case performance of $\underline{R}_{C_{min}}(LPT) = \frac{3}{4}$ that is asymptotically tight. The result was afterwards tightened by Csirik et al. (1992) who proved a worst-case performance of $\underline{R}_{C_{min}}(LPT) = \frac{3m-1}{4m-2}$ in dependence on m. About two decades later Walter (2013) analyzed the performance of a restricted version of LPT that is called RLPT, where iteratively subsets of jobs are assigned to distinct machines, and proved that $C_{min}^{LPT} \geq C_{min}^{RLPT}$ holds.

Besides the analysis of the LPT rule, further publications deal with other approximate procedures as well as the description of branch-and-bound algorithms. Woeginger (1997) presented a polynomial time approximation scheme (PTAS) that guarantees a worst-case error of $1 + \epsilon$ and has a time complexity of $\mathcal{O}(c_\epsilon n \log m)$ where c_ϵ represents a constant in dependence on the error rate ϵ. The first approach towards the exact solution of $P||C_{min}$ stems from Haouari and Jemmali (2008a) who introduced new upper bounds, two heuristic procedures as well as a branch-and-bound scheme with symmetry breaking components. Another branch-and-bound procedure was introduced by Walter et al. (2016) who developed new upper bounds based on solution structures as well as on the duality to the bin covering problem and developed a depth-first enumeration tree with several novel dominance criteria.

Besides the duality of $P||C_{min}$ and bin covering with respect to bounding techniques and solution approaches, there is also an equivalent coherence between $P||C_{min}$ and the minimum cardinality bin covering problem (MCBCP). The goal of MCBCP is to minimize the number of packed items with weights w_j $(j = 1, \ldots, n)$ such that the load of m identical bins is at least equal to their capacity C. To exploit the link between the two problems, the idea is to set $t_j = w_j$ for all j as well as $C = U_{C_{min}}$ where $U_{C_{min}}$ is an upper bound $U_{C_{min}}$ for $P||C_{min}$ and to check if the n existing items are sufficient to cover the m bins. If not, $U_{C_{min}} - 1$ represents a feasible upper bound for $P||C_{min}$. For an overview on MCBCP we refer to Walter and Lawrinenko (2017) who also introduced lower bounds for the problem which might lead to improved upper bounds for $P||C_{min}$.

2.2.3 Workload balancing

Although the two aforementioned criteria are already related to workload balancing in a wider sense, criteria where the workloads of two or more machines are explicitly considered in the objective function, are more appropriate for an assessment of balancing. For the minimization of the difference between the most and least loaded machine $C_\Delta = C_{max} - C_{min}$ there is only one publication available, that proposes solution approaches (cf. Karmarkar and Karp, 1982). However, their approach is more related to the AI community and will therefore be reviewed within Sect. 2.3. A second paper for $P||C_\Delta$ stems from Ouazene et al. (2014) who introduced a new mixed integer linear programming formulation. They also

performed a computational study to show that in general a makespan optimal schedule is not suited when one is seeking for a balanced schedule.

For the closely related problem where the goal is to minimize the ratio $\frac{C_{max}}{C_{min}}$ the literature is also relatively sparse. It was first mentioned by Coffman and Langston (1984) and motivated by the fact that for $P||C_\Delta$ no meaningful worst-case results can be derived. They analyzed the worst-case performance of LPT and showed that $\overline{R}_{C_{max}/C_{min}}(LPT) = \frac{7}{5}$ is tight for all m. For a 3-partitioning variant of the problem, where exactly three jobs have to be assigned to each machine, Kellerer and Woeginger (1993a) analyzed the worst-case performance of LPT with respect to the ratio of the largest to the smallest item $\beta = \frac{t_1}{t_n}$. They showed that $\overline{R}_{C_{max}/C_{min}}(LPT) = \frac{4\beta+5}{2\beta+7}$ is tight for $1 \leq \beta \leq 4$.

The certainly most suited measurements for workload balancing are obtained, when all machines are taken into account in the objective function, instead of just regarding the machines with minimum and maximum completion time. Therefore the minimization of the p-norm of the completion times $\mathcal{L}_p = (\sum_{i=1}^{m} C_i^p)^{1/p}$ is a meaningful approach to seek for a balanced schedule if suited p values are chosen. Note, that for $p = \infty$ the minimization of the makespan is obtained (cf. Sect. 2.2.1) and for $p = 2$ the problem is a non-normalized variant of the least squares method where $\sum_{i=1}^{m} (C_i - \mu)^2$ with $\mu = \sum_{i=1}^{m} C_i/m$ is minimized.

Chandra and Wong (1975) analyzed the worst-case performance of the LPT rule in dependence on p and showed that it is generally bounded by $\frac{3}{2}$. Polynomial time approximation schemes for the $\mathcal{L}_p$-norm were proposed by Alon et al. (1998) and Epstein and Skall (2004). Although the algorithm of Epstein and Skall (2004) was originally designed for uniformly related machines, they showed that it is also capable of handling identical machines. For the case of ideal sets, where $C_i = C_j \; \forall i \neq j$ holds in an optimal solution, Goldberg and Shapiro (2000) analyzed a class of algorithms that are relaxing the LPT rule and derived a worst-case of $\frac{4}{3}$ for $\mathcal{L}_p$.

The problem of minimizing $\sum_{i=1}^{m} C_i^2$ was first treated by Chandra and Wong (1975) who analyzed the worst-case performance of LPT, and derived $\overline{R}_{\sum_{i=1}^{m} C_i^2}(\text{LPT}) = \frac{25}{24}$. Afterwards Leung and Wei (1995) slightly improved the worst-case behavior in cases where $m \mod 5 \neq 0$. For $m = 2$ Koulamas and Kyparisis (2008) presented a delayed LPT heuristic where the five biggest jobs are optimally assigned and afterwards the standard LPT routine is applied to the remaining jobs. The algorithm has a worst-case performance of $\frac{50}{49}$ and a time complexity of $\mathcal{O}(n \log n + c)$ where c is a constant that describes the time needed for the optimal assignment of the first five jobs. Recently, Walter (2015) analyzed the coherence between $P2||C_{max}$ and $P2||\sum_{i=1}^{m} C_i^2$ and derived properties that allow for a transfer of worst-case ratios between both problems. Worst-case bounds for $\mathcal{L}_2$ were obtained by Goldberg and Shapiro (1999), who showed that in the case of ideal sets the LPT-rule has a worst-case performance of $\frac{37}{36}$.

Ho et al. (2009) introduced a new workload balancing criterion, the so called Normalized Sum of Squared Workload Deviations (NSSWD) where $\frac{1}{\mu}(\sum_{i=1}^{m} (C_i - \mu)^2)^{1/2}$ has to be minimized. They also discussed some properties of the problem and introduced a heuristic algorithm that is based on the repeated solution of sub-problems with two machines with respect to the makespan criterion. Later, Ouazene et al. (2014) showed that the NSSWD

criterion is equivalent to $\sum_{i=1}^{m} C_i^2$ as

$$\sum_{i=1}^{m} (C_i - \mu)^2 = \sum_{i=1}^{m} \left(C_i^2 - 2C_i\mu + \mu^2\right) = \sum_{i=1}^{m} C_i^2 - 2\mu \sum_{j=1}^{n} p_j + m\mu^2 = \sum_{i=1}^{m} C_i^2 - m\mu^2 \quad (2.7)$$

holds.

Consequently, NSSWD just represents a normalized variant of $\sum_{i=1}^{m} C_i^2$ and is therefore a representative of the $\mathcal{L}_2$-norm. Cossari et al. (2012) proposed an algorithm for $P||$NSSWD with two phases, where at first partial solutions are merged until a feasible solution is achieved and afterwards an improvement phase is performed where different local search techniques are applied. Later, Walter and Lawrinenko (2014) corrected a wrong result of Ho et al. (2009), who claimed that an optimal $P||$NSSWD optimal schedule is always an optimal $P||C_{max}$ schedule, too. The most recent publication that treats algorithmic methods for $P||$NSSWD stems from Schwerdfeger and Walter (2016), where a subset sum based procedure is proposed that exactly solves the case with $m = 3$. The idea is afterwards extended to a local search approach for $m \geq 4$.

Another workload balancing criterion was proposed by Rajakumar et al. (2004) and denoted as the Relative Percentage of Imbalance (RPI) which is defined as $RPI = \frac{1}{m} \sum_{i=1}^{m} (C_{max} - C_i) / C_{max}$. However, as Ho et al. (2009) showed, the measure is equivalent to the minimization of the makespan in case of identical machines, as it converts to $RPI = 1 - \frac{\mu}{C_{max}}$ and therefore just represents a normalized variant of C_{max} with values in $[0, 1]$.

Cossari et al. (2013) proposed a local search algorithm based on shifts and swaps of one or two jobs between two machines and evaluated its performance for three further workload balancing measures, namely the standard deviation $\sigma = \sqrt{\frac{1}{m} \sum_{i=1}^{m} (C_i - \mu)^2}$ that is equivalent to NSSWD, the mean deviation $\delta_1 = \frac{1}{m} \sum_{i=1}^{m} |C_i - \mu|$, and the mean difference $\delta_2 = \frac{1}{m(m-1)} \sum_{i_1=1}^{m} \sum_{i_2=1}^{m} |C_{i_1} - C_{i_2}|$. The normalized variants of the criteria are given through

$$\sigma^{\text{norm}} = \frac{\sigma}{\mu\sqrt{m-1}} = \frac{NSSWD}{\sqrt{m(m-1)}},\ \delta_1^{\text{norm}} = \frac{m\delta_1}{2(m-1)\mu},\ \delta_2^{\text{norm}} = \frac{\delta_2}{2\mu} \quad (2.8)$$

We refer the interested reader to Kumar and Shanker (2001), who analyzed and compared nine different balancing criteria based on imbalance measures.

2.2.4 Job completion time based criteria

Job completion time based criteria are of concern when not only the assignment of the jobs to the machines plays a role (as it is the case within Sect. 2.2.1 – 2.2.3) but also their sequence on the machines. In this context it is important to note, that for the corresponding optimization problems regarding minimization of the sum of job completion times $P||\sum C_j$, the sum of squared job completion times $P||\sum C_j^2$ as well as the sum of the weighted job completion times $P||\sum w_j C_j$ with $w_j \in \mathbb{R}_{\geq 0}$ the basic notation is different compared to the other objective functions in Sect. 2.2.1–2.2.3. So far, C_i denoted the completion time of machine i, however to remain consistent with the literature, the completion time C_j of job j will now be referenced by its index j.

Conway et al. (1967) showed that $P||\sum C_j$ is solvable in polynomial time by a generalization of the shortest processing time rule (GSPT), where the jobs are sorted and labeled in ascending order $t_1 \leq t_2 \leq \ldots \leq t_n$ and assigned to the first available machine and in a tie-break situation the lowest-indexed machine is chosen. However, the same result does not hold for the other two problems.

The complexity of $P||\sum C_j^2$ remained open quite long. Eventually, Cheng and Liu (2004) settled the complexity and showed that the problem is strongly NP-hard. They also analyzed the asymptotic behavior of the SPT rule and showed that the error rate converges in probability to zero when the processing times are uniformly distributed in $[0, 1]$. Further approximation results with respect to the GSPT rule were presented by Della Croce and Koulamas (2012) who showed that the worst-case ratio is between $\frac{\sqrt{5}+2}{\sqrt{5}+1}$ and $\frac{\sqrt{6}+2}{\sqrt{6}+1}$ for $m = 2$ and not less than $\frac{\sqrt{m+4}+2}{\sqrt{m+4}+1}$ for arbitrary m values.

For the problem of minimizing the weighted completion time $\sum w_j C_j$ a significantly higher number of papers has been published. The first work stems from Eastman et al. (1964) who developed a lower bound for the problem that was later enhanced by Webster (1992). The complexity of $P||\sum w_j C_j$ was finally settled by Bruno et al. (1974) who showed that the problem is strongly NP-hard for $m \geq 2$. Baker and Merten (1973) analyzed the problem from a rather technical point of view by providing properties of (near) optimal solutions. They also illustrated the performance of different heuristics in a computational study. Sahni (1976) proposed a polynomial time approximation scheme for fixed m values with time complexity $\mathcal{O}(n(\frac{n^2}{\epsilon})^{m-1})$ where a schedule not worse than $1 + \epsilon$ times the optimal solution is obtained.

Kawaguchi and Kyan (1986) analyzed the performance of the shortest weighted processing time rule (WSPT), a variant of list scheduling for the case where the items are sorted by non-decreasing $\frac{w_j}{t_j}$ values and derived a worst-case performance of $\frac{1}{2}(1 + \sqrt{2}) \approx 1,207$. Later, Schulz (1996) presented a simple constructive heuristic that uses linear programming relaxations and has a worst-case performance of $(3-\frac{1}{m})/(1-\frac{1}{n+1})$, which is generally worse than the result for WSPT. Skutella and Woeginger (2000) were the first who presented a PTAS independently from m for $P||\sum w_j C_j$. The algorithm is based on the observation that the partitioning of jobs in dependence on their ratios $\frac{t_j}{w_j}$ guarantees a good overall performance.

With respect to the exact solution of the problem there has also been some research work. The first branch-and-bound algorithm was described by Elmaghraby and Park (1974) who gave some insights into properties of optimal solutions and proposed a depth-first search based on the lower bound of Eastman et al. (1964). Barnes and Brennan (1977) enhanced the approach by presenting new structural insights as well as an entrapment rule that allows for an elimination of non-optimal branches.

A different branching approach with an enhanced lower bound was presented by Sarin et al. (1988). The idea behind their branching scheme is to avoid symmetric solutions by predefining solution permutations which allow for a reduced branching tree. A backward dynamic programming algorithm which is polynomial in the sum of the job weights $T = \sum t_j$ was proposed by Lee and Uzsoy (1992). The algorithm exploits the fact, that in an optimal solution the jobs on each machine are always sequenced in the WSPT order. Another branch-and-bound algorithm stems from Belouadah and Potts (1994). They

also proposed a new ILP formulation based on time intervals and derived a Lagrangian relaxation of the machine capacity constraints. Azizoglu and Kirca (1999) introduced a branch-and-bound algorithm based on a new lower bound and dominance criteria based on structural patterns of optimal solutions. The last algorithm for the exact solution of $P||\sum w_jC_j$ stems from van den Akker et al. (1999) who proposed a binary branch-and-bound algorithm. They also introduced a column generation approach on the set covering formulation of the problem and showed in a computational study that it leads to strong lower bounds. Another interesting problem variant was analyzed by Xu and Nagi (2013), who proposed a column generation procedure for a problem formulation that combines C_{max} and $\sum w_jC_j$.

2.2.5 Classification of the scheduling literature

The following Table 2.1 summarizes the relevant literature on identical parallel machine problems. Each row belongs to a specific paper, gives the $\alpha|\beta|\gamma$-notation of the problem(s) under consideration, and its main contribution(s) with respect to the classification below. Although we carefully investigated the literature, we do not claim the table to be complete and to contain every single publication and every single aspect of its contributions.

general:

AN probabilistic analysis
CG column generation approaches
DA benchmark data set generation
LU bound computation (lower (upper) bounds for maximization (min.) problems)
MA mathematical model
SU survey
TC theoretical characteristics of the solution space
WC worst-case analysis of bounds and/or approximation algorithms

exact solution:

BB branch-and-bound
BP branch-and-price
CP cutting plane algorithm
DP dynamic programming
EN enumeration approach
IW iterative weakening

heuristic solution:

BS beam search approach
CH constructive heuristic
HS harmony search
GA genetic algorithm (including scatter search and memetic algorithms)
GR greedy randomized adaptive search procedure
IH improvement heuristic (including local search approaches)
PA population based approach (including bee colony and particle swarm)
PT polynomial time approximation scheme
SA simulated annealing
TS tabu search

publication	notation	main contribution
Alon et al. (1998)	$P\|\|\left(\sum C_i^p\right)^{1/p}$	PT
Alvim and Ribeiro (2004)	$P\|\|C_{max}$	IH
Azizoglu and Kirca (1999)	$P\|\|\sum w_jC_j$	BB, LU, TC
Baker and Merten (1973)	$P\|\|\sum w_jC_j$	TC
Barnes and Brennan (1977)	$P\|\|\sum w_jC_j$	BB, TC
Belouadah and Potts (1994)	$P\|\|\sum w_jC_j$	BB, LU, MA, TC
Blazewicz (1987)	$P\|\|C_{max}$	SU
Bruno et al. (1974)	$P\|\|C_{max}$, $P\|\|\sum w_jC_j$	WC
Chandra and Wong (1975)	$P\|\|\left(\sum C_i^p\right)^{1/p}$, $P\|\|\sum C_i^2$	WC
Chen et al. (2012)	$P\|\|C_{max}$	HS
Cheng and Liu (2004)	$P\|\|\sum C_j^2$	AN, TC
Cheng and Sin (1990)	$P\|\|C_{max}$	SU
Chiaselotti et al. (2010)	$P\|\|C_{max}$	CH
Coffman and Gilbert (1985)	$P\|\|C_{max}/C_{min}$	AN
Coffman and Langston (1984)	$P\|\|C_{max}/C_{min}$	WC
Coffman et al. (1978)	$P\|\|C_{max}$	IH, WC
Conway et al. (1967)	$P\|\|\sum C_j$	TC
Cossari et al. (2012)	$P\|\|$NSSWD	IH
Cossari et al. (2013)	$P\|\|\sigma^{\text{norm}}$, $P\|\|\delta_1^{\text{norm}}$, $P\|\|\delta_2^{\text{norm}}$	IH
Csirik et al. (1992)	$P\|\|C_{min}$	WC
Davidović et al. (2012)	$P\|\|C_{max}$	PA
Dell'Amico and Martello (1995)	$P\|\|C_{max}$	BB, LU, TC, WC
Dell'Amico and Martello (2005)	$P\|\|C_{max}$	BB
Dell'Amico et al. (2008)	$P\|\|C_{max}$	BP, CG, GA
Della Croce and Koulamas (2012)	$P\|\|\sum C_j^2$	WC
Deuermeyer et al. (1982)	$P\|\|C_{min}$	WC
Eastman et al. (1964)	$P\|\|\sum w_jC_j$	LU
Elmaghraby and Park (1974)	$P\|\|\sum w_jC_j$	BB, TC
Epstein and Skall (2004)	$P\|\|\left(\sum C_i^p\right)^{1/p}$	PT
França et al. (1994)	$P\|\|C_{max}$	DA, IH
F.-Ghomi and J.-Ghazvini (1998)	$P\|\|C_{max}$	IH
Finn and Horowitz (1979)	$P\|\|C_{max}$	IH, WC
Frangioni et al. (2004)	$P\|\|C_{max}$	DA, IH
Frenk and Rinnooy Kan (1986)	$P\|\|C_{max}$	AN
Frenk and Rinnooy Kan (1987)	$P\|\|C_{max}$	AN
Friesen (1984)	$P\|\|C_{max}$	WC
Friesen and Langston (1986)	$P\|\|C_{max}$	CH, WC
Goldberg and Shapiro (1999)	$P\|\|\left(\sum C_i^2\right)^{1/2}$	WC
Goldberg and Shapiro (2000)	$P\|\|\left(\sum C_i^p\right)^{1/p}$	CH, WC
Graham (1966)	$P\|\|C_{max}$	WC
Graham (1969)	$P\|\|C_{max}$	HC, WC
Gualtieri et al. (2008)	$P\|\|C_{max}$	CH
Gualtieri et al. (2009)	$P\|\|C_{max}$	CH

Gupta and Ruiz-Torres (2001)	$P\|\|C_{max}$	CH, WC
Haouari and Jemmali (2008a)	$P\|\|C_{min}$	BB, IH, LU, TC
Haouari and Jemmali (2008b)	$P\|\|C_{max}$	BB, IH, TC
Haouari et al. (2006)	$P\|\|C_{max}$	IH, LU, TC
Ho and Wong (1995)	$P\|\|C_{max}$	IH
Ho et al. (2009)	$P\|\|$NSSWD	IH
Hochbaum and Shmoys (1987)	$P\|\|C_{max}$	LU
Hübscher and Glover (1994)	$P\|\|C_{max}$	TS
Kashan and Karimi (2009)	$P\|\|C_{max}$	PA
Kawaguchi and Kyan (1986)	$P\|\|\sum w_jC_j$	WC
Kellerer and Woeginger (1993a)	$P\|k=3\|C_{max}/C_{min}$	WC
Koulamas and Kyparisis (2008)	$P2\|\|\sum C_i^2$	CH, WC
Langston (1982)	$P\|\|C_{max}$	IH, WC
Lawler et al. (1993)	$P\|\|C_{max}$	SU
Lee and Massey (1988)	$P\|\|C_{max}$	CH, WC
Lee and Uzsoy (1992)	$P\|\|\sum w_jC_j$	DP
Lee et al. (2006)	$P\|\|C_{max}$	SA
Leung and Wei (1995)	$P\|\|\sum C_i^2$	WC
McNaughton (1959)	$P\|\|C_{max}$	LU, TC
Min and Cheng (1999)	$P\|\|C_{max}$	GA
Mokotoff (2001)	$P\|\|C_{max}$	SU
Mokotoff (2004)	$P\|\|C_{max}$	CP, MA
Ouazene et al. (2014)	$P\|\|$NSSWD, $P\|\|C_\Delta$	MA, TC
Paletta and Pietramala (2007)	$P\|\|C_{max}$	CH, WC
Paletta and Ruiz-Torres (2015)	$P\|\|C_{max}$	CH, WC
Paletta and Vocaturo (2011)	$P\|\|C_{max}$	IH
Riera et al. (1996)	$P\|\|C_{max}$	IH
Rothkopf (1966)	$P\|\|C_{max}$	DP
Sahni (1976)	$P\|\|\sum w_jC_j$	PT
Sarin et al. (1988)	$P\|\|\sum w_jC_j$	BB, LU
Schulz (1996)	$P\|\|\sum w_jC_j$	CH, WC
Schwerdfeger and Walter (2016)	$P\|\|$NSSWD	IH
Skutella and Woeginger (2000)	$P\|\|\sum w_jC_j$	PT
Tang and Luo (2006)	$P\|\|C_{max}$	IH
Thesen (1998)	$P\|\|C_{max}$	TS
van den Akker et al. (1999)	$P\|\|\sum w_jC_j$	BB, CG, LU
Walter (2013)	$P\|\|C_{min}$	TC
Walter (2015)	$P2\|\|\sum C_i^2$	WC
Walter and Lawrinenko (2014)	$P\|\|$NSSWD	TC
Walter and Lawrinenko (2016)	$P\|\|C_{max}$	BB, TC
Walter et al. (2016)	$P\|\|C_{min}$	BB, LU, TC
Webster (1992)	$P\|\|\sum w_jC_j$	LU
Woeginger (1997)	$P\|\|C_{min}$	PT
Yue (1990)	$P\|\|C_{max}$	WC

Table 2.1: Literature on identical machine scheduling problems

2.3 Number partitioning

In the AI community the task of assigning n integers a_j of a set $\mathcal{S}$ to m subsets $S_1, \ldots, S_m$ with $\mathcal{S} = \dot{\cup}_{i=1,\ldots,m} S_i$ in order to minimize the largest subset sum or to balance the subset sums are usually known as multi-way number partitioning and balanced multi-way number partitioning resp. Therefore, number partitioning is a kind of theoretical foundation, while the equivalent problem of parallel machine scheduling represents a possible practical application. However, as both communities have an individual research scope, a thorough review of the literature in the AI community for number partitioning will be given now.

2.3.1 Two-way number partitioning

Two-way number number partitioning describes the case of parallel machine scheduling with $m = 2$ which is also equivalent to the subset sum problem. The first noticeable algorithm for two-way number partitioning stems from Horowitz and Sahni (1974) who introduced a heuristics procedure where the integers are divided into two randomly generated subsets of size $\frac{n}{2}$ each and the goal is to find combined subset sums of the two halves that are as close to $\sum_{j=1}^{n} a_j/2$ as possible. The runtime of their algorithm is $\mathcal{O}(n2^{n/2})$ since it requires a sorted list of subsets. Schroeppel and Shamir (1981) improved the algorithm with respect to the memory usage by handling the subsets in a sorted order by the usage of a min-heap.

Karmarkar and Karp (1982) proposed the so called Differencing Method that laid the foundation for many further researches on the topics of two-way, multi-way and balanced multi-way number partitioning. Their algorithm is based on the idea, that a solution can be obtained by an iterative replacement of the two largest remaining integers, say a_i and a_j, by their difference $|a_i - a_j|$ in the original list of integers. They also extended this idea for an arbitrary number of subsets by combining partial solutions. The time complexity of the Karmarkar-Karp algorithm is $\mathcal{O}(n \log n)$.

Their approach was later extended by Korf (1998) to an exact solution procedure for two-way number partitioning. Here, a binary search tree is used to allow for the condition that the two largest remaining integers can also be assigned to different subsets. Korf (1998) also extended a greedy-based heuristic to an exact solution procedure (called Complete Greedy Algorithm) by applying a binary search tree and introducing several dominance criteria. Both algorithms have a time complexity of $\mathcal{O}(2^n)$ due to the structure of the binary search tree.

Since the early 90s several meta heuristic approaches have been published for two-way number partitioning. Johnson et al. (1991) proposed a simulated annealing algorithm for two-way number partitioning and described the difficulty of finding a suitable neighborhood definition. Based on these results, Ruml et al. (1996) analyzed the impact of encoding schemes and search techniques such as simulated annealing or genetic algorithms and concluded, that the choice of a suited encoding scheme is crucial for the search efficacy. Argüello et al. (1996) introduced two randomized versions of the Differencing Method of

Karmarkar and Karp (1982) by applying the Greedy Randomized Adaptive Search Procedure (GRASP) methodology. The first algorithm performs the differencing operation on random integers, while the second algorithm applies differencing on a subset of $\mathcal{S}$ first and afterwards on the remaining integers.

Later, Berretta et al. (2004) developed recombination approaches that aim for weight-matching and are used as operators within a memetic algorithm. Alidaee et al. (2005) showed that two-way number partitioning as well as balanced multi-way number partitioning can also be modeled as unconstrained quadratic binary programs and applied a tabu search algorithm for the modified formulation. The latest paper stems from Pedroso and Kubo (2010) who proposed a depth-first branch-and-bound algorithm and a breadth-first beam search heuristic, which uses a diving method based on the Differencing Method, enabling a quick identification of feasible solutions.

For the case that the a_i values are real numbers drawn from a uniform distribution instead of being integer, several probabilistic analyses with respect to the optimal difference between both sets were made, e.g. by Karmarkar et al. (1986), Lueker (1987), Tsai (1992), and Yakir (1996). For the case that the integers are distributed in the range $\left\{1, \ldots, 2^{kn}\right\}$, the analysis of phase transition received a lot of attention. Phase transition basically describes the phenomenon that there is a drastic change with respect to the required time for the determination of an optimal solution for a certain k value, as the probability of perfect partitions (i. e. $C_1 = C_2$) tends to zero. For the analysis of the phase transition we refer, e.g. to Korf (1998), Gent and Walsh (1998), and Mertens (1998, 2006).

2.3.1.1 Multi-way number partitioning

For multi-way number partitioning with the goal of minimizing the largest subset sum, Michiels et al. (2007) analyzed the worst-cast performance of the Differencing Method of Karmarkar and Karp (1982) and showed that for $m \geq 3$ it is bounded between $\frac{4}{3} - \frac{1}{3(m-1)}$ and $\frac{4}{3} - \frac{1}{3m}$. Korf (2009) proposed an exact recursive binary tree algorithm, where at each node of the tree the inclusion or exclusion of an integer to a certain number of subsets is performed. Subproblems with two remaining subsets are solved with the extended Karmarkar and Karp algorithm. In Korf (2011) the approach was improved by replacing the binary tree with an extended version of the Schroeppel and Shamir algorithm that allows for a more effective search of the solution space. A different branch-and-bound approach was presented in Moffitt (2013). Here, all possible sets of integers are sequentially generated that may be assigned to S_1 while the remaining integers are assigned by a binary search tree to the remaining $m-1$ subsets. The algorithm uses the principle of weakest-link optimally, what basically means that a suboptimal solution for $m-1$ sets may still lead to a global optimal solution if $C_1 \geq \max\{C_2, \ldots, C_m\}$ holds. The most recent publication on the exact solution of multi-way number partitioning stems from Schreiber and Korf (2014). Their cached iterative weakening algorithm iteratively increases a given lower bound L until an optimal solution is found with $C^* = L$. To allow for an efficient search of the solution space, possible sets of integers in a certain range $[L, U]$ are only generated once in a preprocessing step. For the generation of feasible solutions, these sets of integers are consecutively considered in an increasing cardinality order.

2.3.2 Balanced multi-way number partitioning

For the balanced multi-way number partitioning problem the relevant literature distinguishes between two different objectives, (i) minimizing the difference between the maximum and minimum subset and (ii) minimizing the largest subset where the cardinality of each subset either is $\lceil \frac{n}{m} \rceil$ or $\lfloor \frac{n}{m} \rfloor$ (cf. Zhang et al, 2011). In case that $\frac{n}{m}$ is integer, the problem variant is equivalent to k-partitioning where exactly $k = \frac{n}{m}$ numbers have to be assigned to each subset. While the practical applicability of (i) was already discussed in the context of workload balancing (cf. Sect. 2.2.3), problems of type (ii) occur, e.g. in the field of flexible manufacturing systems (cf. Dell'Amico et al., 2006) where each unit of the system is only able to perform a certain amount of different operations, because of specific tool magazines with a limited capacity. Especially in literature about problems with cardinality constraints, there are not always clear borders between Operations Research and Artificial Intelligence.

Tasi (1995) proposed a modified version of the Karmarkar and Karp heuristic where it is ensured that each subset receives either $\lceil \frac{n}{m} \rceil$ or $\lfloor \frac{n}{m} \rfloor$ integers. With respect to (ii), Michiels et al. (2012) analyzed the worst-case behavior of the Karmarkar and Karp heuristic in dependence on k and m. For any fixed k they showed that the worst-case ratio is between $2 - \sum_{i=0}^{k-1} \frac{i!}{k!}$ and $2 - \frac{1}{k-1}$. In dependence on m they proved a worst-case ratio of $2 - \frac{1}{m}$.

For the k-partitioning variant, Babel et al. (1998) introduced several lower bound arguments as well as approximation algorithms and analyzed their worst-case performance. Further publications on this problem stem from Dell'Amico and Martello (2001), where lower bounds and corresponding worst-case analyses are regarded and from Dell'Amico et al. (2004), where among others a scatter search approach and a branch-and-bound algorithm are introduced. The latest paper for balanced multi-way number partitioning stems from Zhang et al. (2011) who developed two heuristic algorithms which are able to solve instances with an odd cardinality limit and a skewed data set respectively. For $k = 3$ Kellerer and Kotov (1999) developed a $\frac{7}{6}$-approximation algorithm, where selected large integers are assigned to different subsets and a binary search procedure is applied for the assignment of the remaining integers.

Another similar problem variant is given, when each subset has an individual cardinality limit k_i. The first contribution to this problem stems from Dell'Amico et al. (2006) who developed several lower bounds, e.g. based on column generation, as well as reduction criteria and heuristic solution procedures such as scatter search. Further publications are due to Zhang et al. (2009) and Kellerer and Kotov (2011) who both dealt with approximation algorithms.

When analogous to $P||C_{min}$ the objective function is changed such that the smallest subset shall be maximized, two further articles are available. He et al. (2003) introduced constructive heuristics for k-partitioning as well as k_i-partitioning and analyzed their worst-case performances. For k_i-partitioning, Lawrinenko et al. (2016) introduced several lower bounds, preprocessing and lifting strategies as well as heuristic solution methods such as a dynamic programming based improvement procedure.

2.3.3 Classification of the number partitioning literature

In the same vein as the overview on the most relevant OR papers in Sect. 2.2.5 a classification for the number partitioning literature is given in Table 2.2. As there is no unified classification scheme available, like the $\alpha|\beta|\gamma$-notation for scheduling problems, we use the following abbreviations for the corresponding problem formulations.

2WNP/MWNP	two-way/multi-way number partitioning problem
BMNP	balanced multi-way number partitioning problem
k-PP/k_i-PP	k/k_i-partitioning problem

publication	**notation**	**main contribution**
Alidaee et al. (2005)	2WNP, BMNP	MA, TS
Argüello et al. (1996)	2WNP	GR
Babel et al. (1998)	k-PP	CH, LU
Berretta et al. (2004)	2WNP	GA
Dell'Amico and Martello (2001)	k-PP	LU, WC
Dell'Amico et al. (2004)	k-PP	BB, CH, GA
Dell'Amico et al. (2006)	k_i-PP	CG, CH, GA, LU
Gent and Walsh (1998)	2WNP	TC
He et al. (2003)	k/k_i-PP	CH, WC
Horowitz and Sahni (1974)	2WNP	CH
Johnson et al. (1991)	2WNP	SA
Karmarkar and Karp (1982)	2WNP, BMNP	CH
Karmarkar et al. (1986)	2WNP	AN
Kellerer and Kotov (1999)	k-PP	CH, WC
Kellerer and Kotov (2011)	k_i-PP	CH, WC
Korf (1998)	2WNP	EN, TC
Korf (2009)	MWNP	BB
Korf (2011)	MWNP	BB
Lawrinenko et al. (2016)	k_i-PP	CH, IH, LU, TC
Lueker (1987)	2WNP	AN
Mertens (1998)	2WNP	TC
Mertens (2006)	2WNP	TC
Michiels et al. (2007)	MWNP	WC
Michiels et al. (2012)	BMNP	WC
Moffitt (2013)	MWNP	BB
Pedroso and Kubo (2010)	2WNP	BB, BS
Ruml et al. (1996)	2WNP	TC
Schreiber and Korf (2014)	MWNP	IW
Schroeppel and Shamir (1981)	2WNP	CH
Tasi (1995)	BMNP	CH
Tsai (1992)	2WNP	AN
Yakir (1996)	2WNP	AN
Zhang et al. (2009)	k_i-PP	CH, WC
Zhang et al. (2011)	k-P	CH

Table 2.2: Literature on number partitioning problems

2.4 Conclusion

In this Chapter an overview on the most relevant literature with respect to bounding techniques, approximation ratios, as well as solution procedures for identical parallel machine scheduling and number partitioning has been provided. The last survey on the topic was published over 15 years ago, so a complete overview of the latest developments is mandatory because a lot of progress has been made since then, especially on the field of meta heuristics. In this context, we also reviewed the literature for less studied objective functions such as the machine covering problem $P||C_{min}$ or k_i-partitioning and gave a sketch of their practical applicability and their correlation to other similarly structured optimization problems. Also an overview on the literature for number partitioning has been provided, which is the common term for identical parallel machine scheduling in the AI community.

Although identical parallel machine scheduling and number partitioning are two terms for the same topic, the relationship between both respective communities is surprisingly rather loose, as AI literature is only seldom mentioned in OR papers and vice versa. Also the research questions of both communities are not always identical. In OR, solution procedures are typically theoretically analyzed with respect to their worst-case performance, while the probabilistic behavior is more often investigated in the AI community. For the identification of hard instances, OR papers typically analyze parameter settings with respect to the number of jobs and machines, while AI papers have a stronger focus on the influence of the size of input data and the number of bits required to store them. Bridging the gap between both communities and using the mutual knowledge appears to be a promising approach in future research projects, since it allows for a wider scope on the topic.

Chapter 3

Effective solution space limitation for the identical parallel machine scheduling problem

Summary

For the classical makespan minimization problem on identical parallel machines, we identify universally valid characteristics of optimal schedules. Based on these novel structural insights we derive several strong dominance criteria.
Implemented in a branch-and-bound algorithm these criteria have proved to be effective in limiting the solution space, particularly in the case of small ratios of the number of jobs to the number of machines.

3.1 Introduction

In this Chapter we are concerned with the following fundamental scheduling problem. Given a set $M = \{M_1, \ldots, M_m\}$ of $m \geq 2$ identical parallel machines and a set $J = \{J_1, \ldots, J_n\}$ of $n > m$ independent jobs with positive processing times $t_1, t_2, \ldots, t_n$, the task consists in a non-preemptive assignment of the jobs to the machines so that the latest machine completion time (also called *makespan*) $C_{max} = \max\{C_1, \ldots, C_m\}$ – where C_i is the sum of processing times of jobs assigned to M_i – is minimized. Using the three-field notation of Graham et al. (1979) this problem is abbreviated as $P||C_{max}$.

Problem $P||C_{max}$ is well-known to be $\mathcal{NP}$-hard in the strong sense (see Garey and Johnson, 1979) and it constitutes one of the very basic problems in scheduling theory which has received (and still receives) a lot of attention from researchers as well as practitioners. As a result, numerous publications – primarily on approximation algorithms and (meta-)heuristic approaches – appeared. Here, substantial contributions during the last years stem from Alvim and Ribeiro (2004), Frangioni et al. (2004), Dell'Amico et al. (2008), Paletta and Vocaturo (2011), and Davidović et al. (2012), to name but a few. However, contributions to exact solution procedures (cf. Dell'Amico and Martello, 1995; Mokotoff,

2004; Dell'Amico et al., 2008; Haouari and Jemmali, 2008b) are quite rare and, so far, only little is known about structural patterns of optimal solutions (cf. Dell'Amico and Martello, 1995).

Both of the last-mentioned issues are addressed by the present Chapter which is composed of a theoretical part (Sections 3.2 and 3.3) and an algorithmic part (Sections 3.4 and 3.5). In the theoretical part we (i) provide a thorough investigation of the underlying solution space resulting in the identification of novel structural patterns of makespan-optimal schedules (cf. Section 3.2) and (ii) derive new dominance criteria based on the previous results (cf. Section 3.3). In the subsequent algorithmic part we (i) elaborate on how the new insights can be efficiently implemented in a branch-and-bound algorithm (cf. Section 3.4) and (ii) present results of a comprehensive experimental study on a set of difficult benchmark instances (cf. Section 3.5). Finally, Section 3.6 concludes the Chapter with some ideas for future research.

In the remainder of the Chapter we presuppose the jobs to be labeled so that $t_1 \geq t_2 \geq \ldots \geq t_n$.

3.2 Theoretical study of the solution space

This section contains a profound theoretical study of the underlying solution space and focuses on the derivation of necessary conditions for makespan-optimal solutions. For this purpose, we developed an approach that resembles the idea of inverse optimization (cf., e.g., Ahuja and Orlin, 2001).

3.2.1 A symmetry-breaking solution representation

In order to avoid symmetric solutions resulting from a simple renumbering of the machines, we use a schedule representation that we call *non-permuted*, meaning that a schedule $S \in \{1, 2, \ldots, m\}^n$ – where $S(j) = i$ indicates that job j is assigned to machine i – fulfills the following two conditions:

1. $S(1) = 1$
2. $S(j) \in \left\{1, \ldots, \min\{m, 1 + \max_{1 \leq k \leq j-1} S(k)\}\right\}$ for all $j = 2, \ldots, n$.

In other words, in a non-permuted schedule it never occurs that a job is assigned to a machine i_2 as long as there exists an empty machine i_1 with $i_1 < i_2$. Obviously, any schedule that does not fulfill the two conditions can be transformed into a non-permuted one by simply renumbering the machines. As a consequence, only non-permuted schedules will be considered in the remainder of the Chapter.

3.2.2 Potential optimality

In order to obtain insights into structural patterns of optimal schedules, we developed the following approach. Unlike the usual approach where one is seeking an optimal schedule to a given $P||C_{max}$-instance, we consider arbitrary schedules and ask whether we can select processing times so that the respective schedule becomes (uniquely) makespan-optimal.

If possible, the respective schedule is said to be *potentially* (unique) makespan-optimal, otherwise the schedule can never become (uniquely) makespan-optimal and is therefore said to be *non-potentially* (unique) makespan-optimal. Obviously, in a mathematical sense, the set of potentially unique optimal schedules $\mathcal{S}^{**}$ is a subset of the set of potentially optimal schedules $\mathcal{S}^{*}$ which is itself a subset of the set of all solutions $\mathcal{S}$, i.e., in general we have $\mathcal{S}^{**} \subset \mathcal{S}^{*} \subset \mathcal{S}$. Furthermore, the set of non-potentially unique makespan-optimal schedules $\bar{\mathcal{S}}^{**}$ is the relative complement of $\mathcal{S}^{**}$ in $\mathcal{S}$, i.e., $\bar{\mathcal{S}}^{**} = \mathcal{S} \setminus \mathcal{S}^{**}$. Thus, a complete characterization of $\bar{\mathcal{S}}^{**}$ directly implies a complete characterization of $\mathcal{S}^{**}$. As will be seen next, we focus on characterizing $\bar{\mathcal{S}}^{**}$ instead of directly characterizing $\mathcal{S}^{**}$ as this turned out to be more viable. We start with an analysis of the two machine case upon which, later on, the analysis of the generalized case of $m \geq 3$ machines will build substantially.

3.2.2.1 The two machine case

In a first step, we introduce an alternative method to illustrate schedules. Usually, Gantt charts are the means of choice to display which job is assigned to which machine and at which time the processing of a job starts and ends. However, as the concept of potential optimality does not base on processing times but rather on information concerning the current number of jobs assigned to each machine, we propose the following simple illustration of schedules called *path-representation*. Associated with this kind of representation is the definition of the *corresponding path* P_S to a schedule S. To put it simply, the corresponding path P_S to a schedule S is a string of length $n+1$ where the j-th entry $(j = 1, \ldots, n)$ represents the difference between the number of jobs assigned to machine 1 and machine 2 in S after the assignment of the first j jobs. Note that these are the j longest ones because we presuppose the jobs to be sorted in non-increasing order of their processing times. Further, we set $P_S(0) = 0$ representing initially empty machines. In a graphical illustration of a path, the entries are linearly connected (see Example 3.2.1).

Example 3.2.1
Consider the following three schedules $S_1 = (1, 2, 2, 1, 2, 1)$, $S_2 = (1, 1, 2, 2, 1, 2)$, $S_3 = (1, 1, 1, 1, 2, 2)$ *and their corresponding paths presented below.*

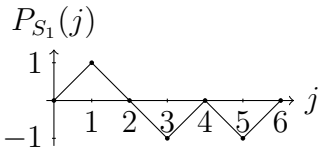


Figure 3.1: Path P_{S_1}

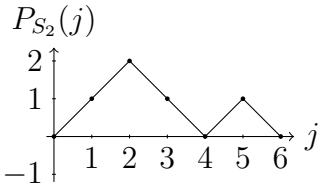


Figure 3.2: Path P_{S_2}

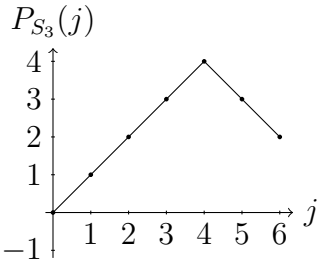


Figure 3.3: Path P_{S_3}

It is readily verified that S_1 *constitutes a potentially unique makespan-optimal schedule, e.g., for the vector of processing times* $T_1 = (9, 6, 5, 3, 2, 1)$, *while* S_2 *is indeed potentially makespan-optimal but not potentially unique makespan-optimal (consider* $T_2 = (3, 3, 3, 3, 2, 2)$ *for which, among others,* S_1 *and* S_2 *are optimal). For reasons of comparison,* S_3 *serves as an example for a non-potentially makespan-optimal solution.*

Before we proceed, we want to remark that the path representation is neither based on explicit values of the processing times nor does it provide information about them. Requiring the jobs to be sorted in non-increasing order of their processing times, it is just a catchy way to illustrate non-permuted schedules.

For the theoretical study of structural patterns of optimal $P2||C_{max}$-schedules, the set

$$S_{NEG}(n) = \Big\{ S \,|\, P_S(j) < 0 \;\text{ for some }\; j \in \{3, \ldots, n\} \Big\}$$

plays a crucial role. Schedules belonging to $S_{NEG}(n)$ are said to fulfill the (two-machine) *path-condition*. If no further specification is needed, we use S_{NEG} instead of $S_{NEG}(n)$.

Theorem 3.2.2
Let S be a schedule which is no element of $S_{NEG}(n)$. Then, S is a non-potentially unique makespan-optimal schedule.

Proof
Consider a schedule $S \notin S_{NEG}$ and let $J_1(S) = \{a_1, \ldots, a_r\}$ and $J_2(S) = \{b_1, \ldots, b_s\}$ denote the set of job-indices that are assigned to machine 1 and 2, respectively. Note that $r \geq s$ and $r + s = n$ holds. Furthermore, w.l.o.g. assume that $1 = a_1 < a_2 < \ldots < a_r$ and $b_1 < \ldots < b_s$. Since $S \notin S_{NEG}$, we can conclude

$$a_1 < b_1, a_2 < b_2, \ldots, a_s < b_s.$$

Thus, the makespan of S is $C_{max}(S) = \sum_{k=1}^{r} t_{a_k}$.

In order to prove that S is a non-potentially unique makespan-optimal solution, we construct a schedule $\bar{S}$ that is "not longer" than S, i.e., $C_{max}(\bar{S}) \leq C_{max}(S)$, for any feasible instance. We distinguish two cases according to the number of jobs assigned to machine 2 in S.

1. $s < 2$.

   Consider a schedule $\bar{S}$ which is obtained from S by shifting job a_2 to machine 2. Obviously, this cannot increase the makespan.

2. $s \geq 2$.

   Consider a schedule $\bar{S}$ which is obtained from S by swapping the jobs a_s and b_s, i.e., $J_1(\bar{S}) = \{a_1, \ldots, a_{s-1}, a_{s+1}, \ldots, a_r, b_s\}$ and $J_2(\bar{S}) = \{b_1, \ldots, b_{s-1}, a_s\}$. Then, for the makespan of schedule $\bar{S}$ we get

   $$C_{max}(\bar{S}) = \max \Big\{ \sum_{k=1}^{s-1} t_{a_k} + t_{b_s} + \sum_{k=s+1}^{r} t_{a_k}, \sum_{k=1}^{s-1} t_{b_k} + t_{a_s} \Big\}.$$

   Provided that $C_{max}(\bar{S}) = \sum_{k=1}^{s-1} t_{a_k} + t_{b_s} + \sum_{k=s+1}^{r} t_{a_k}$, we can conclude

   $$\sum_{k=1}^{s-1} t_{a_k} + t_{b_s} + \sum_{k=s+1}^{r} t_{a_k} \leq \sum_{k=1}^{r} t_{a_k} = C_{max}(S).$$

In the other case, i.e., $C_{max}(\bar{S}) = \sum_{k=1}^{s-1} t_{b_k} + t_{a_s}$, we can conclude

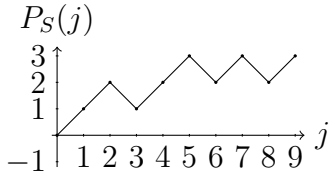
$$\sum_{k=1}^{s-1} t_{b_k} + t_{a_s} \leq \sum_{k=1}^{s-1} t_{a_k} + t_{a_s} = \sum_{k=1}^{s} t_{a_k} \leq \sum_{k=1}^{r} t_{a_k} = C_{max}(S).$$

Thus, we have that $C_{max}(\bar{S}) \leq C_{max}(S)$.

This completes the proof of the theorem as in either case a schedule $\bar{S}$ exists which is not longer than S (for any feasible instance of $P2||C_{max}$). ■

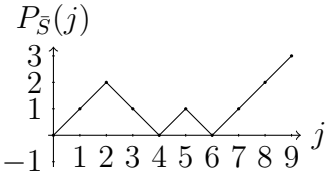
Note that the constructed schedule $\bar{S}$ itself is not required to be an element of S_{NEG}. However, it is quite obvious that an iterative application of the shifting- (cf. Case 1) and/or the swapping-operation (cf. Case 2) will turn a given schedule $S \notin S_{NEG}$ into a schedule S' that is actually an element of S_{NEG}. We call the whole process the *two-machine path-modification*. Clearly, during this process – which is illustrated in Example 3.2.3– the makespan cannot increase.

Example 3.2.3
Assume $n = 8$, $m = 2$ and consider the following vector of processing times $T = (20, 18, 15, 12, 10, 10, 8, 5, 2)$. The initial schedule $S = (1, 1, 2, 1, 1, 2, 1, 2, 1)$ is transformed to $\bar{S} = (1, 1, 2, 2, 1, 2, 1, 1, 1)$ in a first step and then into a potentially unique makespan-optimal schedule $S' = (1, 1, 2, 2, 2, 1, 1, 1, 1)$.



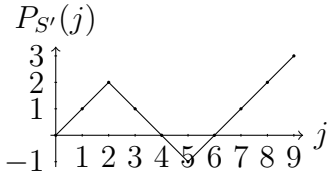
$J_1(S) =$	$J_2(S) =$
$\{1, 2, \mathbf{4}, 5, 7, 9\}$	$\{3, 6, \mathbf{8}\}$
$C_1(S) = 70$	$C_2(S) = 30$

Figure 3.4: Schedule S



$J_1(\bar{S}) =$	$J_2(\bar{S}) =$
$\{1, 2, \mathbf{5}, 7, 8, 9\}$	$\{3, 4, \mathbf{6}\}$
$C_1(\bar{S}) = 63$	$C_2(\bar{S}) = 37$

Figure 3.5: Schedule $\bar{S}$



$J_1(S') =$	$J_2(S') =$
$\{1, 2, 6, 7, 8, 9\}$	$\{3, 4, 5\}$
$C_1(S') = 63$	$C_2(S') = 37$

Figure 3.6: Schedule S'

For the sequel it is useful to record two important properties of the swapping-operation:

- The $s - 1$ longest jobs on machine 1 are not affected by any swap.
- After each swap – turning a schedule $S \notin S_{NEG}$ into another schedule $\bar{S}$ – the entries in the corresponding paths interrelate as follows:

$$P_{\bar{S}}(j) = \begin{cases} P_S(j), & \text{if } j = 1, \ldots, a_s - 1, \\ P_S(j) - 2, & \text{if } j = a_s, \ldots, b_s - 1, \\ P_S(j), & \text{if } j = b_s, \ldots, n. \end{cases}$$

As a consequence, a modified schedule S' fulfills the path-condition at position $2s - 1$ for which exactly $(P_S(2s - 1) + 1)/2$ swaps are required.

3.2.2.2 The generalized case

At first, we briefly explain how we make use of the path-representation in the generalized case. As we now have more than two machines, it seems natural to represent a schedule S via a set of $\binom{m}{2}$ paths. This set contains for each pair (i_1, i_2) of machines $(1 \leq i_1 < i_2 \leq m)$ the corresponding path $P_S^{(i_1,i_2)}$. The paths are defined as in the previous subsection, i.e., $P_S^{(i_1,i_2)}(j)$ represents the difference between the number of jobs assigned to i_1 and i_2 in S after the assignment of the j longest of the n jobs.

Example 3.2.4
Assume $n = 9$, $m = 3$ and consider the schedule $S = (1, 2, 1, 2, 2, 3, 1, 1, 3)$. The corresponding paths to the three machine-pairs are illustrated below.

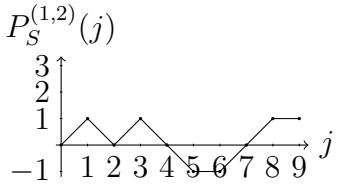


Figure 3.7: Path $P_S^{(1,2)}$

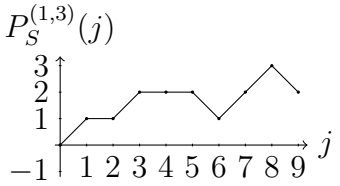


Figure 3.8: Path $P_S^{(1,3)}$

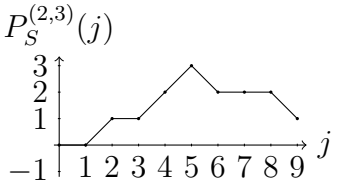


Figure 3.9: Path $P_S^{(2,3)}$

Note that besides up- and downward moves, now horizontal moves appear as well in the graphical illustration of the corresponding paths.

Next, the definition of the set $S_{NEG}(n)$ is extended to the set $S_{NEGX}(n, m)$ as follows:

$$S_{NEGX}(n,m) = \Big\{ S \,|\, \text{for each pair } (i_1,i_2) \text{ there exists either a } j \in \{3,\dots,n\} \text{ so that } P_S^{(i_1,i_2)}(j) < 0,$$
$$\text{or } 0 < j_1 \leq j_2 < n \text{ so that } P_S^{(i_1,i_2)}(j) = 1 \text{ for } j = j_1,\dots,j_2 \text{ and } P_S^{(i_1,i_2)}(j) = 0 \text{ else } \Big\}.$$

Note that $S_{NEGX}(n, m)$ contains all non-permuted schedules of n jobs on m machines where (i) each machine processes at least one job and (ii) each corresponding path has at least one negative entry if the graphical illustration of the path has less than $n - 2$ horizontal moves. We say that such schedules fulfill the *path-conditions*. Again, we write S_{NEGX} for short. Then, our main theorem reads as follows.

Theorem 3.2.5
Let S be a schedule which is no element of $S_{NEGX}(n, m)$ $(m \geq 3)$. Then, S is a non-potentially unique makespan-optimal schedule. Moreover, every schedule $S \notin S_{NEGX}(n, m)$ can be turned into a schedule that is an element of $S_{NEGX}(n, m)$ by successive application of the two-machine path-modification.

Lemma 3.2.6
Let $1 \leq i_1 < i_2 < i_3 \leq m$ and consider a schedule S where $P_S^{(i_1,i_2)}$ fulfills its path-condition while $P_S^{(i_1,i_3)}$ does not. Then, application of the two-machine path-modification to $P_S^{(i_1,i_3)}$ preserves the fulfillment of the path-condition of (i_1, i_2).

Lemma 3.2.7
Let $1 \leq i_1 < i_2 < i_3 \leq m$ and consider a schedule S where both $P_S^{(i_1,i_2)}$ and $P_S^{(i_1,i_3)}$ fulfill their path-condition while $P_S^{(i_2,i_3)}$ does not. Then, application of the two-machine path-modification to $P_S^{(i_2,i_3)}$ preserves the fulfillment of the path-condition of (i_1, i_2) and (i_1, i_3).

The respective proofs of the two lemmata are to be found in the Appendix as they are rather technical.

Proof of Theorem 3.2.5
The first part of the proof is trivial. Since S is no element of $S_{NEGX}(n, m)$, there exists a pair of machines – say (i_1, i_2) – whose corresponding path does not fulfill the path-condition. According to Theorem 3.2.2, after application of the two-machine path-modification to the pair (i_1, i_2) the respective path very well fulfills its path-condition without increasing the maximum completion time of i_1 and i_2. Moreover, as the path-modification does not affect any jobs on the remaining $m - 2$ machines, the makespan of the transformed schedule cannot be greater than the makespan of S. Hence, S cannot be potentially unique makespan-optimal.

The second part can be proved with the help of the two lemmata 3.2.6 and 3.2.7. At first, we consider the path $P_S^{(1,2)}$. In case that $P_S^{(1,2)}$ does not fulfill its path-condition, we apply the two-machine path-modification (cf. end of Section 3.2.2.1). Afterwards, we iteratively apply the two-machine path-modification to the paths $P_S^{(1,3)}, \ldots, P_S^{(1,m)}$. According to Lemma 3.2.6, this yields that each path $P_S^{(1,i)}$ $(i = 2, \ldots, m)$ fulfills its path-condition. Now, with regards to non-permuted schedule, a renumbering of the machines might be necessary. Then, we apply the two-machine path-modification to $P_S^{(2,3)}$. Since both $P_S^{(1,2)}$ and $P_S^{(1,3)}$ already fulfill their path-condition, Lemma 3.2.7 ensures that the fulfillment is preserved during the modification of $P_S^{(2,3)}$. Next, we iteratively apply the two-machine path-modification to the paths $P_S^{(2,4)}, \ldots, P_S^{(2,m)}$. According to the two lemmata, we now obtain that not only the paths $P_S^{(1,i)}$ $(i = 2, \ldots, m)$ fulfill their path-condition but also the paths $P_S^{(2,i)}$ $(i = 3, \ldots, m)$. Again, a renumbering of the machines might be required. The whole process can be repeated until finally the path $P_S^{(m-1,m)}$ also fulfills its path-condition. As a consequence of the specific iterative application of the two-machine path-modifications, the initial schedule $S \notin S_{NEGX}(n, m)$ is transformed into a schedule that is very well an element of $S_{NEGX}(n, m)$. ■

To bring it back to the beginning of Section 3.2.2 where we aimed at a complete characterization of $\mathcal{S}^{**}$, we can record that $\mathcal{S}^{**} \subseteq S_{NEGX}$. Though we do not have a mathematical proof of the identity of the two sets yet, we strongly conjecture that identity indeed holds as indicated by the results of preliminary experimental studies on small (n, m)-constellations.

3.3 Dominance criteria derived from potential optimality

In this section we deal with how to make use of the new structural insights in a branching based solution procedure. As will be seen next, our findings of the previous section admit a straightforward translation into new dominance criteria which we subsume under the term *path-related* criteria. By means of these new criteria we want to guide the search towards solutions in S_{NEGX} since this set is known to contain at least one optimal solution (cf. Theorem 3.2.5).

3.3.1 Basic dominance criterion

The purpose of the basic criterion is to decide at early stages whether it is possible to complete a partial solution such that it is an element of S_{NEGX}. To answer this question, we identify all machine-pairs which do currently not fulfill their path-condition and determine the *minimum number of required jobs* on the respective machines.

More formally, assume the k longest jobs to be already assigned representing a partial solution $\tilde{S}$ and let $PF_{\tilde{S}}^{(i_1,i_2)}(k) \in \{0,1\}$ indicate for each pair of machines (i_1,i_2) whether its path-condition is currently fulfilled ($PF_{\tilde{S}}^{(i_1,i_2)}(k) = 1$) or not ($PF_{\tilde{S}}^{(i_1,i_2)}(k) = 0$). In accordance with the definition of S_{NEGX}, a pair (i_1,i_2) *currently* fulfills its path-condition if either the corresponding partial path has actually reached the negative sector so far (i.e., the path-condition is fulfilled for sure) or each of the two machines processes exactly one of the first k jobs (i.e., the path-condition is fulfilled up to this point).
Concentrating on the pairs where $PF_{\tilde{S}}^{(i_1,i_2)}(k) = 0$, two cases have to be distinguished according to the current number of jobs assigned to i_1.

1. M_{i_1} processes at most one of the first k jobs.

   In this case it is sufficient to assign one job to i_2 in order to fulfill the respective path-condition.

2. M_{i_1} processes at least two of the first k jobs.

   In this case at least $P_{\tilde{S}}^{(i_1,i_2)}(k) + 1$ jobs still have to be assigned to i_2 to ensure that the (i_1,i_2)-path contains a negative entry.

As each pair (i_1,i_2) with fixed $i_2 \in \{2,\ldots,m\}$ has to fulfill the path-condition,

$$v_{i_2}(k) = \max_{\substack{i_1=1,\ldots,i_2-1 \\ PF_{\tilde{S}}^{(i_1,i_2)}(k)=0}} \left\{P_{\tilde{S}}^{(i_1,i_2)}(k)\right\} + \delta_{i_2}(k) \tag{3.1}$$

gives the minimum number of jobs that still have to be assigned to machine i_2. Clearly, if $PF_{\tilde{S}}^{(i_1,i_2)}(k) = 1$ holds for all $i_1 = 1,\ldots,i_2-1$, then $v_{i_2}(k) = 0$. The additional $\delta_{i_2}(k)$-term corresponds to the aforementioned two cases. More detailed, $\delta_{i_2}(k) = 0$ iff Case 1 holds for all machines $1 \leq i_1 < i_2$. Otherwise, if at least one machine $i_1 < i_2$ processes more than one job (cf. Case 2) then $\delta_{i_2}(k) = 1$.

Then, by assigning the next $v_m(k)$ jobs to machine m, the following $v_{m-1}(k)$ jobs to machine $m-1$ and so on until machine 2 finally receives its $v_2(k)$ required jobs, all $\binom{m}{2}$ path-conditions are satisfied. In case that the minimum number of required jobs exceeds the number of remaining jobs, i.e.,

$$\sum_{i_2=2}^{m} v_{i_2}(k) > n - k \tag{3.2}$$

the current partial solution can be excluded from further search. An example is provided at the end of Section 3.3.

3.3.2 Further improvements

In a first step, we try to increase the minimum number of required jobs. For this purpose, we take a deeper look at pairs of machines where at most one job is currently assigned to each machine. Let (i_1, i_2) be such a pair. Then, according to Section 3.3.1, $v_{i_2}(k)$ is either zero (as the pair currently fulfills its path-condition) or one (since it suffices to assign one job to i_2). As will be shown next, in some cases it is possible to increase $v_{i_2}(k)$ by 1. The increase depends on the minimum number m' of machines that have to process at least two jobs so that the makespan of the corresponding schedule is not greater than a given upper bound U. To compute m', consider the ratios

$$q_i = \frac{T_{Sum} - iU}{m - i} \qquad (i = 0, 1, \ldots, m-1) \tag{3.3}$$

where $T_{Sum} = \sum_{i=1}^{n} t_i$. Starting with $i = 0$, q_0 represents the average machine completion time. Hence, if $q_0 > t_1$ it is quite obvious that at least one machine has to process more than one job. Then, assuming that this machine's completion time equals U – which is a valid simplification – we consider q_1 that represents the minimum average load of the remaining $m-1$ machines. Clearly, in case that $q_1 > t_1$, we can conclude that one of the remaining $m-1$ machines has to process at least two jobs as well. This procedure is repeated as long as $q_i > t_1$ for the first time. In case $U > t_1$, a straightforward calculation yields

$$q_i > t_1 \quad \Longleftrightarrow \quad i < \frac{T_{Sum} - mt_1}{U - t_1} \tag{3.4}$$

by which

$$m' = \left\lceil \frac{T_{Sum} - mt_1}{U - t_1} \right\rceil \tag{3.5}$$

follows. Let $i'(k)$ denote the lowest index of a machine to which currently at least two jobs are assigned, then we can immediately conclude that each machine $i > i'(k)$ has to process at least two jobs as well to fulfill the path-conditions. These are in total $m - i'(k) + 1$ machines. Clearly, in case that $m' > m - i'(k) + 1$, the machines $i = m - m' + 1, \ldots, i'(k) - 1$ also have to process at least two jobs so that the respective $v_i(k)$-values can be increased by one.

Secondly, by incorporation of the processing times we intend to decide whether it is possible to assign the required number of jobs $\sum_{i=1}^{m} v_i(k)$ to the respective machines so

that no machine runs longer than $U-1$. As this problem is $\mathcal{NP}$-hard in the strong sense (by reduction from 3-PARTITION (cf. Garey and Johnson, 1979)), we rather solve a relaxed version (where jobs may be assigned to more than one machine) as described next. Assume that $1 \leq p \leq m$ machines still require at least one job and let $I = \{i_1, i_2, \ldots, i_p\}$ be the set of the corresponding machines, i.e., $v_i(k) > 0$ for all $i \in I$. Then, for each $i \in I$ we determine the longest job j_i that can be assigned to i in combination with the $v_i(k) - 1$ shortest jobs so that i finishes not later $U-1$. More formally,

$$j_i = \min \left\{ j \in \{k+1, \ldots, n\} \mid C_i^k + t_j + \sum_{l=0}^{v_i(k)-2} t_{n-l} \leq U-1 \right\} \tag{3.6}$$

where C_i^k is the current completion time of i after the assignment of the first k jobs. Note that (i) an assignment of a job $j \in \{k+1, \ldots, j_i - 1\}$ to machine i cannot improve U and thus will not lead to a new incumbent solution and (ii) the current partial solution can be fathomed if $C_i^k + \sum_{l=0}^{v_i(k)-2} t_{n-l}$ already exceeds $U-1$.
Further, let π be a permutation of the machines in I which sorts the jobs j_i in order of non-increasing indices. Clearly, in case that $n - j_{\pi(1)} + 1 < v_{\pi(1)}(k)$, the current solution cannot be completed in such a way that all path-conditions are satisfied and the makespan is less than U. In the other case, i.e., $n - j_{\pi(1)} + 1 \geq v_{\pi(1)}(k)$, we check whether $n - j_{\pi(2)} + 1$ is smaller than $v_{\pi(1)}(k) + v_{\pi(2)}(k)$. If this is the case, the partial solution can be fathomed using the same argument as before. Otherwise, the iterative procedure is repeated by successively adding the next machine according to π and checking for $n - j_{\pi(3)} + 1 < \sum_{b=1}^{3} v_{\pi(b)}(k)$, $n - j_{\pi(4)} + 1 < \sum_{b=1}^{4} v_{\pi(b)}(k)$ and so on. In case that one of the inequalities $n - j_{\pi(p')} + 1 < \sum_{b=1}^{p'} v_{\pi(b)}(k)$ $(p' = 1, \ldots, p)$ is fulfilled, the current solution can be fathomed.

Example 3.3.1
Assume $n = 11$, $m = 5$ and consider the following vector of processing times $T = (187, 162, 140, 127, 119, 108, 101, 71, 62, 50, 25)$ for which $U = 237$ constitutes a valid upper bound (by application of the LPT-rule). Furthermore, let $k = 6$ and assume the partial schedule $\tilde{S} = (1, 2, 3, 4, 5, 4)$. Table 3.1 provides the entries of all paths at position 6. The superscript * indicates that the path-condition of the respective pair of machines is currently not fulfilled.

$P_{\tilde{S}}^{(i_1,i_2)}(6)$	2	3	4	5
1	0	0	-1	0
2	-	0	-1	0
3	-	-	-1	0
4	-	-	-	$1^\star$

Table 3.1: Entries of all paths at position $k = 6$

Then, we have $v_i(6) = 0$ for $i = 1, \ldots, 4$ while $v_5(6) = 1 + \delta_5(6) = 1 + 1 = 2$. Since $n - k = 5 > 2 = \sum_{i=1}^{5} v_i(6)$, the current partial schedule cannot be fathomed according to the basic criterion. However, we can increase $v_1(6)$, $v_2(6)$, and $v_3(6)$ by one because $m' = \lceil \frac{1152 - 5 \cdot 187}{237 - 187} \rceil = 5$ and $i'(6) = 4$. Nevertheless, $\tilde{S}$ still cannot be fathomed. At last, we

incorporate the processing times of the five unassigned jobs. Here, we have $I = \{1, 2, 3, 5\}$, $j_1 = 11$, $j_2 = j_3 = j_5 = 8$, *and the fourth iteration yields* $n - j_5 + 1 = 4 < 5 = \sum_{i \in I} v_i(6)$. *Thus, it is not realizable to complete* $\tilde{S}$ *in such a way that all path-conditions are fulfilled and the makespan is less than* U.

3.4 A branch-and-bound algorithm

This section contains a concise description of the proposed branch-and-bound algorithm. The computation of bounds is similar to the one in Haouari and Jemmali (2008b) and the branching scheme is adopted from Dell'Amico and Martello (1995). The most distinctive and innovative component of our algorithm is certainly represented by the path-related dominance criteria.

3.4.1 Lower bounds

We implemented two lower bounds referred to as L_2 and L_3 in Dell'Amico and Martello (1995). The first one, $L_2 = \max\{\lceil \sum_{i=1}^{n} t_i/m \rceil, t_1, t_m + t_{m+1}\}$, is a rather simple one obtained from relaxations of $P||C_{max}$. The second one, $L_3 = \max\{C+1 : \exists t \leq C/2$ for which $B_\alpha(C,t) > m$ or $B_\beta(C,t) > m\}$, utilizes the strong coherence between $P||C_{max}$ and the bin packing problem (BPP). In its core, L_3 consists of two lower bounds $B_\alpha(C,t)$ and $B_\beta(C,t)$ for BPP. For any further details we refer to Dell'Amico and Martello (1995).

In order to enhance lower bounds, we implemented a lifting procedure as described by Haouari et al. (2006). Roughly speaking, with this procedure lower bounds for specific partial instances are determined which are also valid for the entire instance. The lifted version of a bound L is denoted by $\tilde{L}$. Finally, we also implemented a procedure proposed by Haouari and Jemmali (2008b) that tries to tighten a lower bound L by solving a specific subset-sum-problem (SSP). The tightened version is denoted by L_{SSP}. Again, we refer to the literature.

We shall remark that existing literature provides a few more lower bounds (cf., e.g., Fekete and Schepers, 2001, and Webster, 1996). However, we abstain from implementing them here because our focus is on an effective limitation of the solution space (i.e., an effective exploration of the branching tree) rather than on bounding techniques.

3.4.2 Upper bounds

We implemented three procedures, namely the LPT-rule (cf. Graham, 1969), the Multifit-algorithm (cf. Coffman et al., 1978), and an iterated local search improvement heuristic based on Haouari et al. (2006) and Haouari and Jemmali (2008b), to obtain upper bounds U_1, U_2, and U_3, respectively. For further details we refer to the literature.

3.4.3 Application of the bounds

At the root node, the bounds are computed in the following order. At first, we determine L_2 as well as U_1. In case $L_2 = U_1$, an optimal solution is obtained. Otherwise, we compute L_3. If $L_3 < U_1$, we compute U_2 and if $L_3 < U_2$, we additionally determine U_3. If there is

still a gap between L_3 and U_3, the lifted bound $\tilde{L}_3$ is computed. L_{SSP} is only determined in case that $\tilde{L}_3 < U_3$.

To obtain local bounds without consuming too much time, only L_3 is computed at every branched node. For details concerning this computation we refer to Dell'Amico and Martello (1995). Furthermore, we determine U_1 for partial solutions in which at least m jobs have already been assigned and no more than $2m$ jobs remain unassigned.

3.4.4 The branching scheme

In case that the global lower bound $L_G = L_{SSP}$ is smaller than the global upper bound $U_G = U_3$, the branching process starts. We perform a depth-first search where at each level of the branching-tree the job with the longest processing time amongst all unassigned jobs is chosen. So, at level k, the current node generates at most m son-nodes by assigning job k to those machines M_i that fulfill $C_i^{k-1} + t_k < U_G$ where C_i^{k-1} denotes the current completion time of machine i after the assignment of the $k-1$ longest jobs. The corresponding machines are selected according to increasing current completion times. Note that selecting the job with the longest remaining processing time at each level of the tree is well-suited for a straightforward application of the path-related dominance criteria.

A new incumbent solution is determined whenever job n is assigned during the branching process or the application of the LPT-rule at level $k \geq \max\{m+1, n-2m+1\}$ yields a better solution. In this case U_G is updated. Finally, identifying an optimal solution immediately stops the branching process.

3.4.5 Dominance criteria

In case that a current partial solution at level k cannot be fathomed due to the computation of the local lower bound and not all machine-pairs fulfill their path-condition for sure, we apply the path-related dominance criteria derived in Section 3.3. At first, the basic criterion (cf. Section 3.3.1) is applied which consumes $\mathcal{O}(m)$ provided that all relevant information on the paths at level $k-1$ is available. If the number of remaining jobs is sufficiently large, we compute m' (cf. Section 3.3.2) and update the respective $v_i(k)$-values. If necessary, we incorporate the processing times as explained at the end of Section 3.3.2) which can be implemented to run in $\mathcal{O}(mn)$ time. If the current solution still cannot be fathomed, the corresponding node is branched.

The branching process itself can be limited according to four dominance criteria that have originally been introduced by Dell'Amico and Martello (1995). However, in order to apply these criteria along with the new path-related ones, they require some modifications. For sake of readability, all technical details on the issue of compatibility are deferred to the Appendix A.3.

3.5 Computational study

We have coded the algorithm described within Section 3.4 in Java 7.2 language and experimentally tested its performance on various difficult sets of $P||C_{max}$-instances as reported

by Haouari and Jemmali (2008b) and Dell'Amico et al. (2008). Our computational experiments were performed on an Intel Core i7-2600 and 8GB RAM while running Windows 7 Professional SP 1 (64-bit). The maximal computation time per instance was set to 900 seconds and our algorithm was run as a single process/thread.

3.5.1 Performance on Dell'Amico and Martello's instances

In the first experiment, we have run our algorithm (denoted by WL) on the following five problem classes originally proposed by Dell'Amico and Martello (1995):

- Class 1: discrete uniform distribution in $[1, 100]$
- Class 2: discrete uniform distribution in $[20, 100]$
- Class 3: discrete uniform distribution in $[50, 100]$
- Class 4: cut-off normal distribution with $\mu = 100$ and $\sigma = 50$
- Class 5: cut-off normal distribution with $\mu = 100$ and $\sigma = 20$

According to the results stated in the paper by Haouari and Jemmali (2008b), with their algorithm (denoted by HJ) branching was required for only 77 out of 1900 instances. Since we basically use the same bounds, there is no need to reconsider all of their investigated constellations. Instead, we restricted our study to the difficult (n, m)-constellations $(10, 3), (10, 5), (25, 10), (25, 15), (50, 15)$ and the respective classes where branching was required by HJ. Like Haouari and Jemmali (2008b), for each constellation we randomly generated 10 independent instances resulting in a total of 160 instances. A summary of the results is provided in Table 3.2. In this table we document the mean CPU time in seconds (labeled as "Time") as well as the mean number of explored nodes ("NN"). Moreover, the number of unsolved instances if greater than 0 is given in brackets. The results of algorithm HJ are taken from Haouari and Jemmali (2008b). Their algorithm was coded in Microsoft Visual C++ and ran on a Pentium IV 3.2GHz with 1GB RAM. The time limit for HJ was set to 1200 seconds.

Before we proceed, we shall remark that a fair comparison of computation times presented within the sections 3.5.1–3.5.3 appears to be impossible for the following two reasons. Firstly, the algorithms have been coded in different programming languages and secondly, the computer experiments have been carried out on different hardware. Nevertheless, for sake of completeness, we still present computation times.

As can be seen from the entries in Table 3.2, except for the $(50, 15)$-constellation where algorithm HJ performed better than WL on instances of class 4 and 5 (5 unsolved instances on the side of HJ compared to 9 on the side of WL), the performance of WL is strictly superior to HJ's one in terms of both NN and Time. The dominance of WL is particularly impressive for the constellations $(25, 10)$ and $(25, 15)$ where we were able to remarkably reduce the number of explored nodes (fractions of less than $1/150,000$ were achieved) accompanied by a considerable reduction in the mean CPU time. However, noticeable benefit of the path-related dominance criteria seems to be limited to constellations where the fraction of n to m is not greater than about 2.5.

			HJ		WL	
n	m	Class	Time	NN	Time	NN
10	3	1	0.094	150	0.000	6
		2	0.093	127	0.000	7
		3	0.097	142	0.001	15
		4	0.093	131	0.001	22
		5	0.101	136	0.000	12
10	5	2	0.156	174	0.000	3
		4	0.172	225	0.001	2
25	10	1	36.828	4,750,793	0.034	172
		2	121.328	14,814,146	0.076	646
		3	3.648	383,810	0.088	658
		4	12.816	1,434,262	0.068	760
		5	11.005	1,124,031	0.094	837
25	15	2	12.177	1,082,607	0.001	7
50	15	1	0.891	149	0.339	35,549
		4	0.891 (3)	218	40.535 (4)	2,756,219
		5	0.277 (2)	1	0.034 (5)	1

Table 3.2: Results on difficult instances from Dell'Amico and Martello (1995)

3.5.2 Performance on benchmark instances

In a second experiment, we investigated WL's performance on difficult instances included in the benchmark sets proposed by França et al. (1994) as well as Frangioni et al. (2004). The former set consists of uniform instances while the latter one consists of non-uniform instances. For a detailed description, we refer to the literature. All instances can be downloaded from http://www.or.deis.unibo.it/research.html.

Table 3.3 reports on the results obtained by WL in comparison to the ones documented by Haouari and Jemmali (2008b) for algorithm HJ. The experiments were carried out on a total of 70 instances that belong to 7 combinations (n, m, Interval, Uniform/Non-Uniform) for which at least one of the 10 instances per combination could not be solved at the root node by HJ. In addition to the mean CPU time and the mean number of nodes, we also document the mean relative deviation in % (labeled as "Gap") between the makespan of the best solution found by WL and the optimal makespan. Averages are taken over all unsolved instances per combination. Numbers in brackets give the corresponding maximum relative deviation. An optimal solution for each instance of the benchmark set is provided also on http://www.or.deis.unibo.it/research.html.

				HJ		WL		
	n	m	Interval	Time	NN	Time	NN	Gap
Uniform	50	25	$[1, 10^2]$	0.295 (2)	1	0.020	29	– (–)
	50	25	$[1, 10^3]$	0.669 (2)	1	0.187	3,483	– (–)
	100	25	$[1, 10^3]$	1.278	14,364	0.057 (9)	1	0.079 (0.158)
	50	10	$[1, 10^4]$	146.411 (3)	23,607,013	– (10)	–	0.005 (0.008)
	50	25	$[1, 10^4]$	0.369 (1)	1	0.816	66,306	– (–)
	100	25	$[1, 10^4]$	– (10)	–	– (10)	–	0.036 (0.055)
Non-Uniform	100	25	$[1, 10^4]$	246.803 (3)	10,399,656	– (10)	–	0.011 (0.013)

Table 3.3: Results on difficult benchmark instances

Obviously, the performance of our algorithm WL depends strongly on the ratio n/m. While WL easily solved all 30 instances where $n/m = 2$, only 1 of the remaining 40 instances with ratio 4 and 5 has been solved. Thus, WL failed to solve 39 of 70 instances though relative deviations are negligible (mean values are less than 0.08% each).

In contrast, due to tighter global bounds, HJ failed to solve only 21 instances. However, while there are only 16 unsolved instances with ratio 4 and 5, there are also 5 unsolved instances with ratio 2. These are exactly the five $(50, 25)$-instances that required branching in Haouari and Jemmali (2008b). To sum up, the conclusions drawn from the study presented in Section 3.5.1 also apply to the study on difficult benchmark instances.

In addition to Table 3.3, we compared the performances of WL and another algorithm (denoted by DIMM) developed by Dell'Amico et al. (2008). So far, DIMM is considered to be state-of-the-art because this algorithm solved all 780 instances of the benchmark set. Coded in C language, DIMM requires at most 750 seconds per instance on a Pentium IV 3GHz. Profiting from very tight upper bounds generated by a scatter search procedure, DIMM already solved 758 out of the 780 instances at the root node. For the remaining 22 instances (19 uniform and 3 non-uniform ones), we explicitly evaluate the performance of our algorithm WL. Among these 22 instances, there are 4, 14, and 4 instances where the ratio of n to m is 2, 4, and 5, respectively. All but one of them belong to the 7 combinations studied in Table 3.3. As can be seen from Table 3.4, WL is superior to DIMM in solving 3 of the 4 instances with ratio 2 and performs equally well on the fourth one, i.e., the constellation $(10, 5)$ which was not included in Table 3.3. Unsurprisingly, WL was not able to solve the remaining 18 instances within the time limit. However, again, relative deviations are negligible (about 0.026% on average over the 18 instances and a maximum of about 0.055%).

					DIMM	WL
	n	m	Interval	No.	Time	Time
Uniform	50	25	$[1, 10^2]$	3	30.08	0.14
	50	25	$[1, 10^3]$	3	30.02	1.06
	10	5	$[1, 10^4]$	6	0.02	0.03
	50	25	$[1, 10^4]$	1	30.03	8.16

Table 3.4: Detailed results on difficult benchmark instances where n/m=2

3.5.3 Performance on Haouari and Jemmali's instances

In a third experiment, we have run our algorithm on a problem class where the ratio of n to m is 2.5 and the processing times are drawn from the discrete uniform distribution on $[n/5, n/2]$. This class has been proposed by Haouari and Jemmali (2008b) with the aim of generating difficult instances. We selected six representative values of n and randomly generated 20 instances for each leading to a total number of 120 instances. Table 3.5 provides a comparison of our results obtained for algorithm WL and the ones reported in Haouari and Jemmali (2008b) for algorithm HJ.

Obviously, in terms of the number of solved instances, WL is clearly superior to HJ. While our algorithm succeeded solving 109 out of 120 instances, HJ only solved 91 instances. Interestingly, except for the $(20, 8)$-constellation, HJ failed to solve at least four instances for each of the larger constellations. Moreover, whenever branching was required for an instance with 30, 40, or 50 jobs, HJ was not successful in finding an optimal solution. In contrast, WL easily solved all instances where $n \leq 50$ and 19 out of 20 instances for constellation $(60, 24)$ although several of these 100 instances had to be branched.

		HJ		WL	
n	m	Time	NN	Time	NN
20	8	2.445	456,470	0.001	21
30	12	0.551 (5)	1	0.038	25,593
40	16	0.867 (4)	1	0.060	98,602
50	20	1.200 (8)	1	4.516	8,086,028
60	24	76.031 (5)	9,609,214	34.734 (1)	41,241,961
80	32	5.000 (7)	403,164	118.113 (10)	46,367,371

Table 3.5: Results on Haouari and Jemmali's instances

3.5.4 Performance on further instances

Finally, in a fourth experiment, we explicitly studied the contribution of the path-related dominance criteria. For this purpose, we implemented a close variant of WL (denoted by WL'). Except for not using the path-related dominance criteria in WL', both algorithms are identical. We have run them on a large set of randomly generated instances according to the 5 classes as described within Section 3.5.1. This time, we generated 100 instances for each class and each of the five investigated (n, m)-constellations resulting in a total of 2500 instances. Due to results of the previous subsections, we restricted the fourth experiment to rather small-sized instances in terms of both n and n/m. The results are summarized in Table 3.6.

			WL'		WL	
Class	n	m	Time	NN	Time	NN
1	20	8	0.002	64	0.002	48
	20	10	0.001	6	0.001	6
	25	10	0.008	1,830	0.007	583
	30	12	0.038	11,215	0.022	2,069
	30	15	0.003	38	0.003	25
2	20	8	0.005	828	0.004	241
	20	10	0.002	30	0.001	10
	25	10	0.132	63,667	0.032	5,977
	30	12	1.159	381,301	0.126	17,649
	30	15	0.038	14,454	0.004	56
3	20	8	0.014	4,372	0.003	189
	20	10	0.000	1	0.000	1
	25	10	0.227	77,262	0.008	822
	30	12	6.867	1,939,090	0.049	5,758
	30	15	0.000	1	0.000	1
4	20	8	0.005	1,020	0.005	465
	20	10	0.002	50	0.002	18
	25	10	0.084	37,061	0.037	7,174
	30	12	0.559	147,630	0.375	59,037
	30	15	0.037	11,997	0.007	410
5	20	8	0.028	10,753	0.005	553
	20	10	0.043	22,053	0.000	4
	25	10	1.223	428,673	0.033	4,857
	30	12	45.105 (1)	11,378,741	0.391	50,799
	30	15	10.242 (4)	4,265,014	0.001	9

Table 3.6: Results on the effectiveness of the path-related dominance criteria for $n/m \in \{2, 2.5\}$

Obviously, the results reveal that both algorithms perform very well in terms of number of solved instances. While WL quickly solved all 2500 instances, its variant WL' failed to solve 5 instances which is still a very good result. However, comparing the mean CPU times

and the mean number of explored nodes, WL is clearly superior to WL' as for all (n, m)-constellations and classes $Time(WL) \leq Time(WL')$ and $NN(WL) \leq NN(WL')$ hold. In fact, in almost all cases we have $Time(WL) < Time(WL')$ and $NN(WL) \ll NN(WL')$. The most remarkable improvements were obtained in class 5 for the two constellations where $n/m = 2$. Here, WL was able to reduce the number of explored nodes by up to six orders of magnitude and the mean CPU time by up to four orders of magnitude. All in all, the results impressively show that the path-related dominance criteria effectively limit the solution space for small ratios of n to m. Studying larger ratios in the context of the fourth experiment is not meaningful because most likely either both algorithms will solve a given instance at the root node or both will fail to find/verify an optimal solution within the time limit of 900 seconds.

3.5.5 General remarks

To summarize the results from our computational study on difficult $P||C_{max}$-instances, we can state that the performance of our proposed branch-and-bound algorithm WL depends strongly on the ratio of the number of jobs n to the number of machines m. While WL performs very well on instances where $n/m \leq 2.5$, its performance on instances with intermediate ratios of n to m deteriorates. In our first three experiments (see sections 3.5.1–3.5.3) we primarily compared the performance of the algorithms WL and HJ on a total of 350 instances – 230 instances with $n/m \leq 2.5$ and 120 instances with $2.5 < n/m \leq 5$. While WL solved almost all instances (219 of 230) with $n/m \leq 2.5$, only 72 of the remaining 120 could be solved. In contrast, HJ solved only 196 of 230 instances with $n/m \leq 2.5$ but 99 of the remaining 120 ones. However, most of those 99 instances were already solved at the root node because in HJ slightly tighter lower bounds are implemented than in WL.

One explanation for WL's superiority in solving instances with small ratios of n to m is the effectiveness of the novel path-related dominance criteria in limiting the solution space. In case of small n/m-values, the number of schedules which fulfill the path-conditions is comparatively small. Hence, the size of the branching tree is limited effectively and the tree can be searched efficiently using a branching scheme that bases upon job-assignments in non-increasing order of processing times as required by the path-related dominance criteria. However, the benefit of the path-related dominance criteria decreases with increasing ratio n/m which is due to an increased number of schedules that fulfill the path-conditions.

3.6 Conclusions

The Chapter addressed the classical makespan minimization problem on identical parallel machines for which we identified new structural patterns of optimal solutions. Later, we transferred these insights into novel *path-related* dominance criteria and implemented them in a branch-and-bound algorithm whose performance has been examined in an extensive computational study on benchmark instances from the literature. As one main result we can record that our procedure performs extremely well on instances where the ratio of n to m is not greater than 2.5. For those instances, application of the path-related criteria leads to an effective limitation of the solution space and our approach is superior to existing

procedures by Dell'Amico et al. (2008) and Haouari and Jemmali (2008b). However, for larger ratios the benefit of the new criteria decreases and results obtained by the other two procedures are in general at least as good as ours.

Based on the output of our study, we see several fruitful paths for future research. At first, it would be interesting to examine the impact of the path-related dominance criteria on solving similarly structured optimization problems since the results of our theoretical study do not only apply to $P||C_{max}$. Specifically, we think of related scheduling problems (e.g., $P||C_{min}$), the classical bin packing problem as well as the multiple knapsack problem.

Another encouraging direction is to explicitly consider job processing times at the determination of optimal solution properties instead of using only information on their order. The intention is to further restrict the set of potentially optimal solutions for certain (n, m)-constellations and distributions of processing times. We expect this technically challenging extension to result in even more effective dominance criteria. Moreover, it would be interesting to investigate whether the novel dominance criteria can also be effectively used in branching schemes that differ from the one implemented here.

Finally, it might be interesting to incorporate the new dominance criteria into a MIP formulation of $P||C_{max}$ and to analyze whether MIP solvers benefit from the new model formulation.

Chapter 4

Improved approaches to the exact solution of the machine covering problem

Summary

For the basic problem of scheduling a set of n independent jobs on a set of m identical parallel machines with the objective of maximizing the minimum machine completion time – also referred to as *machine covering* – we propose a new exact branch-and-bound algorithm. Its most distinctive components are a different symmetry-breaking solution representation, enhanced lower and upper bounds, and effective novel dominance criteria derived from structural patterns of optimal schedules. Results of a comprehensive computational study conducted on benchmark instances attest to the effectiveness of our approach, particularly for small ratios of n to m.

4.1 Introduction

One of the most fundamental and well studied $\mathcal{NP}$-hard problems in the field of machine scheduling is the makespan minimization on identical parallel machines where a set of n independent jobs $\mathcal{J}= \{J_1, \ldots, J_n\}$ with positive processing times $t_1, \ldots, t_n$ has to be assigned to m identical parallel machines $\mathcal{M} = \{M_1, \ldots, M_m\}$ so that the latest machine completion time is minimized. A related and in some sense dual but by far not as well studied $\mathcal{NP}$-hard problem is obtained when the objective is changed from minimizing the makespan C_{max} to maximizing the minimum completion time $C_{min} = \min\{C_1, \ldots, C_m\}$ – without introducing idle times – where C_i is the sum of processing times of all jobs assigned to M_i. While the former problem (which is denoted by $P||C_{max}$ using the three-field notation of Graham et al. (1979)) is a kind of packing problems, the latter problem (abbreviated as $P||C_{min}$), which is the subject of this Chapter, belongs to the class of covering problems and therefore is also referred to as the machine covering problem. It has been first described by Friesen and Deuermeyer (1981) in the context of spare parts

assignments to machines that undergo repeated repair. As another application of $P||C_{min}$, Haouari and Jemmali (2008a) mentioned (fair) regional allocations of investments.

Although both problems basically intend to balance the workload among a given set of resources, $P||C_{min}$ has received less attention than its much more prominent counterpart $P||C_{max}$. To the best of our knowledge, the rather scarce literature on $P||C_{min}$ is limited to a few studies on approximation algorithms and their worst-case ratios and up to now only one exact solution procedure has been proposed. For the well-known longest processing time rule (LPT), Deuermeyer et al. (1982) showed that the minimum completion time C_{min}^{LPT} of the LPT-schedule is never less than 3/4 times the optimal minimum completion time C_{min}^{*}. Ten years later, Csirik et al. (1992) tightened this performance bound by proving that $C_{min}^{LPT}/C_{min}^{*} \geq (3m-1)/(4m-2)$ is fulfilled for any fixed m. In this context, Walter (2013) recently examined the performance relationship between the LPT-rule and a restricted version of it – known as RLPT – and he proved that $C_{min}^{LPT} \geq C_{min}^{RLPT}$. Further publications are concerned with a polynomial-time approximation scheme (PTAS) (cf. Woeginger, 1997) and on-line as well as semi-on-line versions of $P||C_{min}$ (cf., e.g., Azar and Epstein, 1998, He and Tan, 2002, Luo and Sun, 2005, Ebenlendr et al., 2006, Cai, 2007, Tan and Wu, 2007, Epstein et al., 2011). The sole publication devoted to exact solution procedures is due to Haouari and Jemmali (2008a). The main features of their branch-and-bound algorithm are tight lower and upper bounds and a symmetry-breaking solution structure. However, except for small-sized instances, computational results revealed that their algorithm fails to (quickly) solve instances where the ratio of n to m ranges between two and about three.

To overcome this drawback, we approach the machine covering problem from a similar perspective as done by Walter and Lawrinenko (2014) for the dual problem $P||C_{max}$. In their very recent contribution, the authors present structural properties of (potentially) makespan-optimal schedules. These properties are then transformed into problem-specific dominance criteria and implemented in a tailored branch-and-bound algorithm that performs very well on instances with small n/m-values.

Motivated by Walter and Lawrinenko's results, in this Chapter we propose a tailored branch-and-bound algorithm for problem $P||C_{min}$. Our contribution differs substantially from the contribution by Walter and Lawrinenko (2014) in the following respects: (i) we show that their central result on makespan-optimal schedules also applies to problem $P||C_{min}$, (ii) we extend their basic dominance criterion by several novel $P||C_{min}$-specific dominance criteria derived from properties of optimal $P||C_{min}$-schedules, and (iii) we develop new upper bounds on C_{min}^{*}. Some of these bounds are derived from the solution structure, while others exploit the coherence between $P||C_{min}$ and the bin covering problem (BCP). The latter problem consists in packing a set of indivisible items into as many bins as possible so that the total weight of each bin equals at least C. To the best of our knowledge, this coherence has not been mentioned before in the literature.

The Chapter is organized as follows. In the technical part we describe properties of C_{min}-optimal schedules (cf. Sect. 4.2) and translate them into novel $P||C_{min}$-specific dominance criteria (cf. Sect. 4.3). In the algorithmic part, we provide a concise description of the proposed branch-and-bound algorithm (Sect. 4.4) and we evaluate its performance on different sets of benchmark instances (cf. Sect. 4.5). The Chapter concludes with a short summary and some interesting ideas for future research in Sect. 4.6.

For economy of notation, we usually identify both machines and jobs by their index. Moreover, w.l.o.g, we assume the jobs to be indexed so that $t_1 \geq \ldots \geq t_n$. In addition, to avoid trivial instances we presuppose $n > m \geq 2$.

4.2 Theoretical background

In this section, we derive structural properties of C_{min}-optimal schedules, which will be transformed later into problem-specific dominance criteria.

4.2.1 Solution representation and illustration

Throughout this Chapter, we assume schedules to be *non-permuted*. According to Walter and Lawrinenko (2014), a schedule $S \in \{1, 2, \ldots, m\}^n$ – where $S(j) = i$ means that job j is assigned to machine i – is said to be non-permuted if S fulfills the following two conditions:

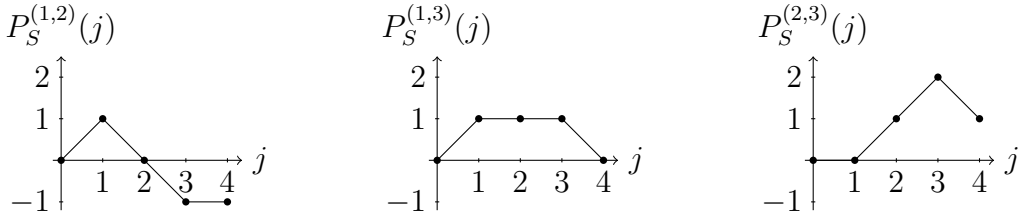
(i) $S(1) = 1$

(ii) $S(j) \in \left\{1, \ldots, \min\{m, 1 + \max_{1 \leq k \leq j-1} S(k)\}\right\}$ for all $j = 2, \ldots, n$.

Note that due to this representation, symmetric reflections obtained by a simple renumbering of the machines are avoided.

Furthermore, instead of using Gantt charts we adopt the illustration of schedules as sets of *paths* which has also been introduced by Walter and Lawrinenko (2014). More precisely, a schedule S is represented by $\binom{m}{2}$ paths $P_S^{(i_1,i_2)}$ – one for each pair (i_1, i_2) of machines $(1 \leq i_1 < i_2 \leq m)$. Each path is a string of length $n + 1$ where the j-th entry $(j = 1, \ldots, n)$ represents the difference between the number of jobs assigned to i_1 and i_2 in S after the assignment of the j *longest* jobs. Additionally, to allow for initially empty machines we set $P_S^{(i_1,i_2)}(0) = 0$ for all pairs. In a graphical illustration, the entries of a path are linearly connected (see Ex. 4.2.1).

Example 4.2.1
Let $n = 4$, $m = 3$, and consider the non-permuted schedule $S = (1, 2, 2, 3)$. The corresponding paths $P_S^{(1,2)} = (0, 1, 0, -1, -1)$, $P_S^{(1,3)} = (0, 1, 1, 1, 0)$, and $P_S^{(2,3)} = (0, 0, 1, 2, 1)$ are illustrated below.



4.2.2 Potential optimality

The concept of potential optimality has recently been proposed by Walter and Lawrinenko (2014) and originates from the question: "Are there certain general patterns in the structure of schedules that cannot lead to optimal solutions?". Admittedly, at first glance this

approach appears to be a bit unusual as actually we intend to derive properties of optimal solutions. However, by identifying properties of solutions that can never become (uniquely) optimal we also implicitly gain insights into the structure of solutions that have the potential to become (uniquely) optimal – which are therefore called *potentially* (unique) optimal.

At this point, we want to mention that the concept of potential optimality is to some extent related with the concept of inverse optimization (for a review, see Ahuja and Orlin, 2001, as well as Heuberger, 2004) where unknown exact values of some adjustable parameters, e.g., processing times, should be determined within given boundaries in such a way that a pre-specified solution becomes optimal and the deviation between the determined and the given values of the parameters is minimal. Although inverse optimization has attracted many researchers in different areas of combinatorial optimization during the last two decades, applications to scheduling problems (e.g., see Koulamas, 2005, as well as Brucker and Shakhlevich, 2009 and 2011) are still rather rare. Our approach – which differs slightly from the basic idea of inverse optimization in that we intend to identify a preferably large set of solutions for which we cannot select processing times so that any of these solutions becomes uniquely $P||C_{min}$-optimal – constitutes another contribution in this field.

As will be seen next (cf. Theorem 4.2.2), solutions contained in the set $\mathcal{S}$ play a crucial role in the context of potentially unique optimal $P||C_{min}$-schedules. The set $\mathcal{S}$ is formally defined as

$$\mathcal{S} = \Big\{ S \,:\, \text{for each pair } (i_1, i_2) \text{ there exists either a } j \in \{3, \ldots, n\} \text{ so that } P_S^{(i_1,i_2)}(j) < 0,$$
$$\text{or } 0 < j_1 \leq j_2 < n \text{ so that } P_S^{(i_1,i_2)}(j) = 1 \text{ for } j = j_1, \ldots, j_2 \text{ and } P_S^{(i_1,i_2)}(j) = 0 \text{ else} \Big\}$$

and contains all schedules where each machine processes at least one job and each path of a pair of machines that processes more than two jobs in total has at least one negative entry. We say that schedules in $\mathcal{S}$ fulfill the *path-conditions* or, equivalently, all $\binom{m}{2}$ paths fulfill their respective path-condition.
Revisiting the schedule $S = (1, 2, 2, 3)$ introduced in Example 4.2.1, we readily see that S is no element of $\mathcal{S}$ although no machine remains empty and the paths $P_S^{(1,2)}$ and $P_S^{(1,3)}$ fulfill their path-condition. However, the path $P_S^{(2,3)}$ does not fulfill its path-condition as it has no negative entry although the two machines process more than two jobs in total.

Our main theorem on potential optimality reads as follows.

Theorem 4.2.2
Let S be a schedule which is no element of $\mathcal{S}$. Then, S is not a potentially unique C_{min}-optimal solution.

Proof
Consider an arbitrary schedule S which is no element of $\mathcal{S}$. Then, there exists a pair of machines – say (i_1, i_2) – whose corresponding path does not fulfill the path-condition. These two machines and the respective set of jobs currently assigned to them constitute a solution to the machine covering problem on two identical parallel machines. Since machine covering and makespan minimization on two identical parallel machines are equivalent, we can apply a result by Walter and Lawrinenko (2014). They proved that with their so

called *two-machine path-modification* a two-machine schedule which does not fulfill its path-condition can be turned into a schedule whose respective path does fulfill its path-condition without increasing the maximum completion time of i_1 and i_2. Note that the latter is equivalent to the fact that the minimum completion time of i_1 and i_2 does not decrease during the modification of the given schedule.

As the two-machine path-modification does not affect any jobs on the remaining $m-2$ machines, the minimum completion time of the transformed schedule cannot be smaller than the minimum completion time of S. Hence, S cannot be potentially unique C_{min}-optimal. ■

Summarizing the result of Theorem 4.2.2, we know that every instance of the problem $P||C_{min}$ has an optimal solution where all corresponding $\binom{m}{2}$ paths fulfill their path-condition.
In what follows, we will make use of the previous result and deduce several $P||C_{min}$-specific dominance criteria – subsumed under the term *path-related* dominance criteria – which will later on prove to be effective in guiding the search of a tailored branch-and-bound algorithm towards schedules which are elements of $\mathcal{S}$.

4.3 Dominance criteria based on potential optimality

For a better understanding, we start with a brief repetition of the basic dominance criterion developed by Walter and Lawrinenko (2014). All other dominance criteria presented within this section are novel and $P||C_{min}$-specific.

4.3.1 The basic criterion

Given a partial solution $\tilde{S}$ where the k longest jobs have already been assigned, the basic criterion is readily obtained from the characterization of potentially unique optimal schedules (cf. Theorem 4.2.2). Recalling that each pair (i_1, i_2) of machines $1 \leq i_1 < i_2 \leq m$ has to fulfill its path-condition, for each $i_2 \in \{m, m-1, \ldots, 2\}$ we simply have to count the minimum number of jobs $v_{i_2}^k$ that still have to be assigned to i_2 so that all paths fulfill their path-condition. According to Walter and Lawrinenko (2014), $v_{i_2}^k$ can be computed as

$$v_{i_2}^k = \max_{\substack{i_1=1,\ldots,i_2-1 \\ PF_{\tilde{S}}^{(i_1,i_2)}(k)=0}} \left\{ P_{\tilde{S}}^{(i_1,i_2)}(k) \right\} + \gamma_{i_2}^k, \tag{4.1}$$

where $PF_{\tilde{S}}^{(i_1,i_2)}(k) = 0$ indicates that the pair (i_1, i_2) does currently not fulfill its path-condition and $\gamma_{i_2}^k$ is a correction term that is either 0 (iff all machines $i_1 < i_2$ process at most one of the first k jobs) or 1 (iff at least one machine $i_1 < i_2$ processes more than one job), respectively. Clearly, $v_{i_2}^k$ is set to 0 if all pairs (i_1, i_2) currently fulfill their path-condition. Then, the basic criterion reads as follows.

Criterion 4.3.1
If

$$\sum_{i=1}^{m} v_i^k > n - k \tag{4.2}$$

for some $k < n$, then the current partial solution can be fathomed.

With regard to the next section, we define $v_1^k = 0$ for all $k > 1$.

4.3.2 Further improvements

Up to now, the basic dominance criterion 4.3.1 does not consider any machine completion times and therefore offers the potential for some improvements. Clearly, in a new incumbent solution each machine completion time has to be at least as large as $L + 1$ where L is the best known minimum completion time so far. Based on this information, at first we will show how some of the demands v_i^k can be tightened and secondly we will check whether the current partial solution admits the possibility to become the new incumbent.

Recalling that we consider a partial solution where the k longest jobs have already been assigned, we let LOW^k denote the set of machines with current completion time C_i^k at most L, i.e. $LOW^k = \{i : C_i^k \leq L\}$, and $\delta_i^k = L + 1 - C_i^k$ for $i \in LOW^k$ denotes the gap between $L + 1$ and C_i^k (see Figure 4.1).

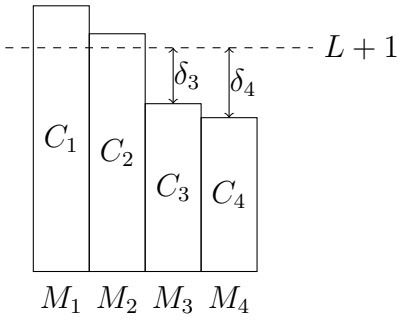


Figure 4.1: Illustration of δ_i

In other words, to improve on the currently best solution (i.e. the incumbent), machine $i \in LOW^k$ has to run at least δ_i^k units of time longer than now. Thus, at least

$$l_i^k = \min\left\{\beta \in \{1, \ldots, n-k\} : \sum_{j=1}^{\beta} t_{k+j} \geq \delta_i^k\right\} \tag{4.3}$$

jobs still have to be assigned to $i \in LOW^k$ in order to finally yield $C_i > L$. If no such β exists, then it is impossible to complete the current partial solution in such a way that a new incumbent solution is obtained. In this case, we set $l_i^k = \infty$. Eventually, the number of required jobs can be updated:

$$\bar{v}_i^k = \begin{cases} \max\{v_i^k, l_i^k\} & \text{if } i \in LOW^k \\ v_i^k & \text{else.} \end{cases} \tag{4.4}$$

Hence, a tighter version of Criterion 4.3.1 reads as follows.

Criterion 4.3.2
If

$$\sum_{i=1}^{m} \bar{v}_i^k > n - k \tag{4.5}$$

for some $k < n$, then the current partial solution can be fathomed.

After having updated the number of required jobs, we will incorporate the processing times of the remaining jobs and derive further criteria. Therefore, we let $T_{rem}^k = \sum_{j=k+1}^{n} t_j$ denote the *total remaining processing time* and we define $\Delta^k = \sum_{i \in LOW^k} \delta_i^k$. Then, it is easy to observe that there is no need to further consider a partial solution if $T_{rem}^k < \Delta^k$. In this case, it is not realizable to assign the remaining jobs to the machines in LOW^k so that finally $C_i > L$ is fulfilled for all $i \in LOW^k$, i.e., no matter how we complete the current partial solution, we cannot obtain a new incumbent solution.

In what follows, we are concerned with tightening the inequality $T_{rem}^k < \Delta^k$, i.e., we want to identify (at low computational costs) a feasible value $R^k > 0$ so that the current solution can already be excluded from further searching whenever

$$T_{rem}^k - R^k < \Delta^k \tag{4.6}$$

is fulfilled. For this purpose, in a first step we take a look at those machines which already run longer than L but require at least one more job to fulfill the path-conditions. We summarize these machines in the set $PATH^k = \{i : C_i^k > L \text{ and } v_i^k > 0\}$ and define

$$V_{PATH}^k = \sum_{i \in PATH^k} v_i^k \tag{4.7}$$

which gives the number of required jobs added over all $i \in PATH^k$. Then, we readily observe that at least V_{PATH}^k of the remaining $n-k$ jobs cannot be used to fill the gaps on the machines in LOW^k or, equivalently, no more than $n - k - V_{PATH}^k$ jobs are available for being assigned to the machines in LOW^k. Hence, we can conclude that R^k is at least as large as the sum of the V_{PATH}^k shortest processing times, i.e. $R^k = \sum_{j=1}^{V_{PATH}^k} t_{n-j+1}$, and (4.6) appears as follows.

Criterion 4.3.3
If

$$T_{rem}^k - \sum_{j=1}^{V_{PATH}^k} t_{n-j+1} < \Delta^k \tag{4.8}$$

for some $k < n$, then the current partial solution can be fathomed.

To shorten notation, in the remainder of this section, we set $n' := n - V_{PATH}^k$. Moreover, for sake of readability and since we always consider a partial solution after the assignment of the k longest jobs, we will omit the upper index k.

We will now show that there is still some potential to further increase the value of R. So far, we do not adequately take into account that the jobs are indivisible. Since preemption is not allowed, the processing of a job cannot be interrupted and continued

on another machine. Therefore, in contrast to our current criteria, if it is not realizable to select some of the remaining $n' - k$ jobs so that a machine $i \in LOW$ finishes exactly at time $L+1$, then the time difference (or surplus) $C_i - (L+1)$ on machine i cannot be used to increase the completion time of other machines in LOW. Recalling that each machine $i \in LOW$ requires at least $\bar{v}_i$ more jobs to fulfill its path-conditions and to allow for a new incumbent, we simply check whether the sum of the $\bar{v}_i$ shortest available jobs, i.e., $n', \ldots, n' - \bar{v}_i + 1$, already exceeds the gap δ_i on that machine. If this is the case, then R can be further increased as follows. For each $i \in LOW$ we compute

$$s_i = \max\{C_i + \sum_{j=1}^{\bar{v}_i} t_{n'-j+1} - (L+1), 0\} \tag{4.9}$$

which we call the *individual surplus* on machine i and

$$S = \sum_{i \in LOW} s_i \tag{4.10}$$

which represents a lower bound on the cumulative surplus caused by the machines $i \in LOW$. Then, it is obvious that in any complete solution, R will be at least as large as $\sum_{j=1}^{V^k_{PATH}} t_{n-j+1} + S$.
At this point, we shall remark that (4.9) (and (4.10)) still assume the V_{PATH} overall shortest jobs to be assigned to the machines in $PATH$. This assumption is correct because of the following trivial lemma which we state without giving a proof.

Lemma 4.3.4
If it is possible to fill all gaps on the machines in LOW using at most $z \leq n' - k$ of the jobs $k+1, \ldots, n$, then it is possible to fill all gaps on these machines using the z longest jobs $k+1, \ldots, k+z$.

As mentioned above, the calculation of the individual surplus s_i for each $i \in LOW$ (cf. Equation (4.9)) is a basic approach because the shortest available jobs are supposed to be repeatedly assignable to each machine in LOW. Clearly, this constitutes an essential simplification since in a feasible schedule each job has to be assigned to one machine. Taking this into account, for the subset of machines $LOW1 := \{i \in LOW : l_i = 1\}$ we will now propose an alternative approach to calculate a lower bound on the cumulative surplus caused by these machines. As will be seen, the restriction to the subset $LOW1$ of LOW ensures the alternative cumulative surplus to be easily and fast computable. The approach builds on the fact that in total (at least) $|LOW1|$ jobs have to be assigned to the machines in $LOW1$ and, since $l_i = 1$ for all $i \in LOW1$, it might be sufficient to assign a single job to each machine $i \in LOW1$ to fill the gaps. So, we can take the $|LOW1|$ shortest available jobs, i.e. $n', \ldots, n' - |LOW1| + 1$, and – assuming that each of which is assigned to exactly one machine in $LOW1$ – we can calculate a lower bound on the cumulative surplus caused by the machines in $LOW1$ as follows. Let

$$s'_i(j) = \max\{C_i + t_j - (L+1), 0\} \text{ for } i \in LOW1 \tag{4.11}$$

denote the *(single) surplus* generated by assigning job j to machine $i \in LOW1$ (see also Figure 4.2), the following lemma is readily verified.

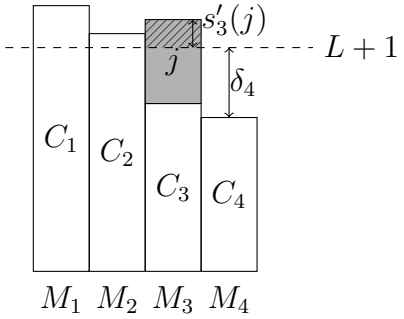


Figure 4.2: Illustration of $s'_i(j)$

Lemma 4.3.5
Consider two jobs j_1, j_2 with $t_{j_1} \geq t_{j_2}$ and two machines $i_1, i_2 \in LOW1$ with $C_{i_1} \geq C_{i_2}$. Then, $s'_{i_1}(j_2) + s'_{i_2}(j_1) \leq s'_{i_1}(j_1) + s'_{i_2}(j_2)$.

Proof
If $s'_{i_1}(j_2) = 0$, then $s'_{i_1}(j_2) + s'_{i_2}(j_1) = s'_{i_2}(j_1) \leq s'_{i_1}(j_1) \leq s'_{i_1}(j_1) + s'_{i_2}(j_2)$ since $C_{i_2} + t_{j_1} - (L+1) \leq C_{i_1} + t_{j_1} - (L+1)$.
If $s'_{i_2}(j_1) = 0$, then $s'_{i_1}(j_2) + s'_{i_2}(j_1) = s'_{i_1}(j_2) \leq s'_{i_1}(j_1) \leq s'_{i_1}(j_1) + s'_{i_2}(j_2)$ since $C_{i_1} + t_{j_2} - (L+1) \leq C_{i_1} + t_{j_1} - (L+1)$.
Now, assume that $C_{i_1} + t_{j_2} > L+1$ and $C_{i_2} + t_{j_1} > L+1$. Then,

$$\begin{aligned} s'_{i_1}(j_2) + s'_{i_2}(j_1) &= C_{i_1} + t_{j_2} - (L+1) + C_{i_2} + t_{j_1} - (L+1) \\ &= C_{i_1} + t_{j_1} - (L+1) + C_{i_2} + t_{j_2} - (L+1) \leq s'_{i_1}(j_1) + s'_{i_2}(j_2). \end{aligned}$$

■

According to Lemma 4.3.5, assigning the longer job to the machine with the larger gap in $LOW1$ and the shorter job to the machine with the smaller gap results in a smaller cumulative surplus (on these two machines) than the other way round. Clearly, this pairwise consideration can easily be extended to all machines in $LOW1$ as summarized in the following corollary.

Corollary 4.3.6
Assume the machines in $LOW1$ to be sorted according to non-increasing gaps. Then, the smallest cumulative surplus caused by the machines in $LOW1$ is obtained by assigning the $|LOW1|$ shortest available jobs in non-decreasing order of processing times to the machines in $LOW1$.

Proof
The corollary can easily be proved by contradiction. At first, observe that considering any $|LOW1|$ jobs out of $k+1, \ldots, n'$ cannot yield a smaller cumulative surplus than selecting the $|LOW1|$ shortest ones because the surpluses are monotonically increasing in increasing processing times.
Now, assume that the smallest cumulative surplus is not achieved by the assignment described in Corollary 4.3.6. Then, there must exist a pair of machines (and jobs) where the machine with the larger gap is assigned the job with the shorter processing time

and the machine with the smaller gap is assigned the job with the longer processing time. According to Lemma 4.3.5, swapping these two jobs results in a smaller cumulative surplus, which is a contradiction. ■

Formally, we can summarize Corollary 4.3.6 as follows: Let π be a permutation of the machines in $LOW1$ so that $C_{\pi(1)} \geq C_{\pi(2)} \geq \ldots \geq C_{\pi(|LOW1|)}$, then the smallest cumulative surplus $S1'$ is obtained by assigning job $n' - i + 1$ to machine $\pi(i)$ for $i = 1, \ldots, |LOW1|$, i.e.

$$S1' = \sum_{i=1}^{|LOW1|} s'_{\pi(i)}(n' - i + 1). \tag{4.12}$$

Note that again Lemma 4.3.4 is taken as granted in the computation of the cumulative surplus $S1'$ on the machines in $LOW1$.

Altogether, we have developed two different approaches to calculate a lower bound on the cumulative surplus caused by the machines in $LOW1$, namely $S1 := \sum_{i \in LOW1} s_i$ and $S1'$. It is readily verified that neither $S1$ dominates $S1'$ (think of cases where $\bar{v}_i = l_i = 1$ for all $i \in LOW1$) nor $S1'$ dominates $S1$. Hence, a lower bound on the cumulative surplus caused by all machines in LOW is given by

$$\bar{S} = \sum_{i \in LOW \setminus LOW1} s_i + \max\{S1, S1'\} = S - S1 + \max\{S1, S1'\}. \tag{4.13}$$

Consequently, we can further increase R by $\bar{S}$ and (4.6) results in the following criterion that is tighter than Criterion 4.3.3.

Criterion 4.3.7
If

$$T_{rem} - \sum_{j=1}^{V_{PATH}} t_{n'+j} - \bar{S} < \Delta \tag{4.14}$$

then the current partial solution can be fathomed.

Summarizing the results developed within this subsection, we can record the following. Given a partial solution where the k longest jobs have already been assigned, in order to obtain a new incumbent solution that fulfills the path-conditions at least $R = \sum_{j=1}^{V_{PATH}} t_{n'+j} + \bar{S}$ units of the total remaining processing time T_{rem} cannot be used effectively to fill the gaps on the machines in LOW – no matter how the remaining $n - k$ jobs will be assigned.

Example 4.3.8
By means of the following example, we briefly illustrate the functionality of the four proposed dominance criteria. Assume $n = 12$, $m = 5$, and the vector of processing times $T = (95, 93, 87, 86, 66, 63, 62, 55, 45, 25, 10, 6)$ for which a lower and an upper bound value of $L = 130$ and $U = 136$ can be determined (cf. Sect. 4.4), respectively. Furthermore, let $\tilde{S} = (1, 2, 3, 4, 5, 5, 3, 4)$ be the current partial schedule where the $k = 8$ longest jobs have already been assigned and $T_{rem} = \sum_{j=9}^{12} t_j = 45 + 25 + 10 + 6 = 86$.
Table 4.1 contains the current machine completion times C_i and the gaps δ_i. The numbers of required jobs according to Criteria 4.3.1 and 4.3.2 are given in Table 4.2. Since

$n - k = 4$ jobs still have to be assigned, the current partial schedule cannot be fathomed on the basis of the first two criteria.
From Tables 4.1 and 4.2 we get $LOW = LOW1 = \{1, 2, 5\}$, $PATH = \{4\}$, and $V_{PATH} = 1$ so that in a first step R is equal to $t_{12} = 6$. Since $T_{rem} - R = 86 - 6$ is not smaller than $\Delta = 36 + 38 + 2 = 76$, we cannot fathom the current partial solution yet (cf. Criterion 4.3.3).
Table 4.3 contains the results of the two alternative surplus computations for the machines in $LOW1$. According to Criterion 4.3.7, the current partial solution can be fathomed because $T_{rem} - 6 - \bar{S} = 86 - 6 - 15 = 65 < 76$.

M_i	1	2	3	4	5
C_i	95	93	149	141	129
δ_i	36	38	-	-	2

Table 4.1: Completion times and gaps

M_i	1	2	3	4	5	$\sum$
v_i	0	0	0	1	0	1
l_i	1	1	–	–	1	3
$\bar{v}_i$	1	1	0	1	1	4

Table 4.2: Numbers of required jobs

M_i	1	2	5	$\sum$
s_i	0	0	8	8
s'_i	0	7	8	15

Table 4.3: Surpluses

4.4 A branch-and-bound algorithm

This section elaborates on the details of the developed branch-and-bound algorithm.

4.4.1 Upper bounds

4.4.1.1 A trivial bound and its worst-case ratio

Let $T_{sum} = \sum_{j=1}^{n} t_j$, then $U_0 = \left\lfloor \frac{T_{sum}}{m} \right\rfloor$ represents the simplest upper bound on the optimal minimum machine completion time C^*_{min} (cf. Haouari and Jemmali, 2008a). It is readily verified that the worst-case performance of U_0 can be arbitrarily bad. For instance, assume $n = m + 1$ and consider the processing times $t_1 = K \gg 1$ and $t_j = 1$ for $j = 2, \ldots, n$. Then, $U_0 = \lfloor K/m \rfloor + 1$, whereas $C^*_{min} = 1$ so that the ratio U_0/C^*_{min} approaches infinity as K grows.

However, based on the following two observations the performance of U_0 can be drastically improved. Firstly, note that in case $n = m + k$ ($k \in \{1, \ldots, m-1\}$) at least $m - k$ machines will process exactly one job in any optimal solution. Thus, the minimum of t_{m-k} and $\lfloor \frac{1}{k} \sum_{j=m-k+1}^{n} t_j \rfloor$ is a valid upper bound on C^*_{min}. Secondly, note that any job whose processing time is greater than or equal to U_0 can be eliminated so that $\lfloor \frac{1}{m-|\mathcal{J}'|} \sum_{j \in \mathcal{J} \setminus \mathcal{J}'} t_j \rfloor$ where $\mathcal{J}' = \{j : t_j \geq U_0\}$ constitutes a valid upper bound on C^*_{min}. Note that $|\mathcal{J}'| \leq m - 1$. Moreover, note that the aforementioned elimination can possibly be repeated up to $m - 1$ times in an iterative manner.

As a result of the previous observations, we can reduce any given instance I to $\tilde{I}$ in such a way that (i) the number of remaining jobs is at least twice the number of remaining machines and (ii) the longest (remaining) processing time is smaller than $U_0(\tilde{I})$. Then, we can prove that $U_0(\tilde{I})$ is never more than $2 - 1/m$ times the optimal minimum completion time.

Theorem 4.4.1
Let I be an instance of $P||C_{min}$ which fulfills both $n \geq 2m$ and $t_1 < U_0(I)$. Then,

$$\frac{U_0(I)}{C^*_{min}(I)} \leq 2 - \frac{1}{m} \tag{4.15}$$

for all instances I and this bound is asymptotically tight for any fixed $m \geq 2$.

Proof
We prove the theorem by contradiction. Assume that $U_0(I)/C^*_{min}(I) > 2 - 1/m$. This is equivalent to $C^*_{min}(I) < U_0(I)/(2-1/m) \leq T_{sum}/(2m-1)$. So, in an optimal schedule, the completion time of at least one machine, say i_1, is less than $T_{sum}/(2m-1)$. Consequently, there exists at least another machine, say i_2, whose completion time C_{i_2} is at least $(T_{sum} - T_{sum}/(2m-1))/(m-1) = 2T_{sum}/(2m-1) > T_{sum}/m > t_1$. Thus, i_2 processes at least two jobs among which the shortest job's processing time is at most $\lfloor C_{i_2}/2 \rfloor$. Shifting this job from i_2 to i_1 yields a better schedule because the completion time of i_2 is still greater than or equal to $T_{sum}/(2m-1)$. This is a contradiction.

To verify that the bound is asymptotically tight for any fixed $m \geq 2$, consider $n = 2m$ jobs with processing times $t_1 = \ldots = t_{n-1} = K \gg 1$ and $t_n = 1$. Then, $C^*_{min} = K + 1$ whereas $U_0 = (2 - 1/m)K + 1/m$. Hence, the ratio of U_0 to C^*_{min} approaches $2 - 1/m$ as K grows. ■

4.4.1.2 Improvements derived from $P||C_{max}$

Following Haouari and Jemmali (2008a), an upper bound on C^*_{min} can be derived from a known lower bound $L_{C_{max}}$ on the optimal makespan by computation of

$$U_1 = \left\lfloor \frac{T_{sum} - L_{C_{max}}}{m-1} \right\rfloor. \tag{4.16}$$

To obtain a good lower bound on the optimal makespan, we implemented an enhanced version (due to Haouari and Jemmali, 2008a) of the bound L_3 by Dell'Amico and Martello (1995). For further details, we refer the reader to the literature.

4.4.1.3 Lifting procedure and further enhancement

Haouari and Jemmali (2008a) proposed two procedures to tighten upper bounds for $P||C_{min}$. The first one is a lifting procedure that bases upon the fact that in any feasible schedule there exists at least a set of l machines ($1 \leq l \leq m$) that process at most

$$\mu_l(n) = l\lfloor n/m \rfloor + \max\{0, n - m(\lfloor n/m \rfloor + 1) + l\} \tag{4.17}$$

jobs (cf. Haouari and Jemmali, 2008a). Then, a lifted bound can be obtained by applying an upper bound procedure on the partial instance restricted to l machines and the $\mu_l(n)$ longest jobs.

The second procedure (cf. Haouari and Jemmali, 2008a) aims at enhancing a given upper bound value U by solving a specific subset sum problem (SSP) that checks whether there exists a subset of $\mathcal{J}$ whose processing times sum up to exactly U. If no such subset exists, the smallest realizable sum (denoted by U_{SSP}) of processing times that does not exceed U constitutes an upper bound.

4.4.1.4 Improvements derived from bin covering

To the best of our knowledge, we are the first to describe the coherence between $P||C_{min}$ and the bin covering problem (BCP) in order to improve upper bounds on C^*_{min}. The idea is to transform a given $P||C_{min}$-instance into a BCP-instance where (i) jobs and processing times correspond to items and weights, respectively, and (ii) the capacity C of the bins is set to the best known upper bound on the minimum completion time. Then, a procedure is applied to determine an upper bound on the maximum number of bins that can be covered. If this number is at most $m-1$, then the optimal minimum completion time of the corresponding $P||C_{min}$-instance is at most $C-1$.

In our implementation, we used four BCP-upper bounds (U_0 from Peeters and Degraeve, 2006, as well as U_1, U_2, and U_3 from Labbé et al., 1995, including their reduction criteria 1 and 2) and an improvement procedure (see Theorem 5 in Labbé et al., 1995). Again, we refrain from reporting any further details on these bounding techniques but refer the interested reader to the literature.

4.4.1.5 Bounds derived from the solution structure

Assume that a lower bound L on C^*_{min} is given (cf. Sect. 4.4.2). Then, in order to generate a new incumbent solution, i.e. $C_{min} > L$, we can deduce that at least

$$j_{min} = \min\{k : t_1 + \ldots + t_k > L\} \tag{4.18}$$

and at most

$$j_{max} = \max\{k : t_{n-k+1} + \ldots + t_n \leq \bar{C}\} \tag{4.19}$$

jobs have to be assigned to each machine and $\bar{C}$ is defined as

$$\bar{C} = \max\left\{C : \left\lfloor \frac{T_{sum} - C}{m-1} \right\rfloor > L\right\}. \tag{4.20}$$

Note that if a machine's completion time is greater than $\bar{C}$ it is impossible that each of the remaining $m-1$ machines runs longer than L.
When the special case $j_{max} = j_{min}+1$ occurs, an immediate upper bound is obtained after determining the number of machines m_{min} (m_{max}) on which exactly j_{min} (j_{max}) jobs are processed each. It is readily verified that $m_{min} = j_{max} \cdot m - n$ and $m_{max} = n - j_{min} \cdot m$. The resulting upper bound is

$$U(L) = \min\left\{\left\lfloor \frac{1}{m_{min}} \sum_{j=1}^{j_{min} \cdot m_{min}} t_j \right\rfloor, \left\lfloor \frac{1}{m_{max}} \sum_{j=1}^{j_{max} \cdot m_{max}} t_j \right\rfloor\right\}. \tag{4.21}$$

In case $j_{max} = 2$, note that an optimal solution is readily obtained by assigning job j ($j = 1, \ldots, m$) to machine j and job $m+k$ ($k = 1, \ldots, n-m$) to machine $m-k+1$.
If $j_{max} > j_{min}+1$ we propose the following strategy. Firstly, if $\lfloor (T_{sum} - t_1 - \ldots - t_{j_{min}})/(m-1) \rfloor \leq L$, it is impossible to obtain a new incumbent by assigning the longest j_{min} jobs to the same machine. In general, a new incumbent can only be obtained at all if none of the machines runs longer than $\tilde{C} := \max\left\{C > L : \left\lfloor \frac{T_{sum}-C}{m-1} \right\rfloor > L\right\}$. Thus, if there exists no j_{min}-element subset of $\mathcal{J}$ whose cumulative processing time falls into the interval $[L+1, \tilde{C}]$

we can increase j_{min}.
Instead of checking each subset individually, we solve the following binary program (requiring pseudo-polynomial time) for a possible increase of j_{min}:

$$\text{Minimize } \tilde{j}_{min} = \sum_{j=1}^{n} x_j \tag{4.22}$$

subject to

$$L + 1 \leq \sum_{j=1}^{n} t_j x_j \leq \tilde{C} \tag{4.23}$$

$$x_j \in \{0, 1\} \qquad \forall j = 1, \ldots, n. \tag{4.24}$$

In case that now $j_{max} = \tilde{j}_{min} + 1$, an immediate upper bound can be determined as described above.
Secondly, we consider restricted instances with l ($l = m - 1, m - 2, \ldots, 2$) machines and the longest μ_l out of all n jobs where μ_l is determined according to Eq. (4.17). Then, for each l, we compute a (restricted) C_{min}-lower bound L_l by application of the LPT-rule and we take the maximum of L and L_l (note that the optimal restricted C_{min}-value is at least as large as C^*_{min} and thus L) to determine $j_{min}(l)$ (cf. Eq. (4.18)) as well as $j_{max}(l)$ (cf. Eq. (4.19)). In case $j_{max}(l) = j_{min}(l) + 1$, we obtain an upper bound $U(L_l)$ according to Eq. (4.21).

4.4.2 Lower bounds

We implemented three construction procedures as well as an improvement procedure. The first constructive algorithm is the prominent longest processing time (LPT)-rule due to Graham (1969). As a second procedure, we implemented a randomized LPT-version due to Haouari and Jemmali (2008a) that randomly decides in each iteration whether the longest or the second longest unassigned job is assigned to the next machine available. Our third procedure is an adaptation of a construction heuristic – referred to as Multi-Subset (MS) (cf. Alvim et al. (2004)) – and consists of two phases. In the first phase, the machines are considered one by one. For each machine, a subset of the yet unassigned jobs is determined (by solving a subset sum problem) so that the longest unassigned job is contained and the sum of the respective processing times is closest to – without exceeding – a given target value T. If not all jobs are assigned after phase 1, the second phase completes the partial solution by assigning the remaining jobs according to the LPT-rule. We used two different values for T, namely $T = UB_{best}$ and $T = LB_{best} + 1$ where UB_{best} and LB_{best} are the best known upper and lower bound value so far, respectively. The better of the two solutions produced by MS is chosen as *the* MS-solution.

The implemented improvement heuristic – referred to as Multi-Start Local Search (MSLS) – is also due to Haouari and Jemmali (2008a). Starting with an initial solution, the procedure attempts to balance the workloads of the machines by iteratively solving a sequence of specific $P2||C_{min}$-instances. For further details we refer to Haouari and Jemmali (2008a).

4.4.3 Application of the bounds

At the root node, the bounds are computed in the following order (as long as no optimal solution has been verified). Presupposing a (possibly preprocessed) instance where $n \geq 2m$, at first we determine U_0 and we apply the LPT-rule yielding a global lower bound value L_G. Secondly, we make use of the lifting and enhancement procedure (see Sect. 4.4.1.3), i.e. for $l = m, m-1, \ldots, 2$ we compute U_1 restricted to l machines and the μ_l longest jobs and afterward we solve for each l the corresponding SSP. Thirdly, we iteratively apply the bounding techniques derived from bin covering (see Sect. 4.4.1.4) as long as at least one of them leads to an improved global upper bound U_G. In a fourth step, we try to further improve L_G by application of (i) MS and (ii) MSLS (see Sect. 4.4.2). The latter is applied to the LPT-solution, the MS-solution as well as to 25 randomized LPT-solutions. Lastly, we apply our upper bounds derived from the solution structure as derived within Sect. 4.4.1.5.

At each node in the tree, we pursue the following two ideas to obtain local upper bounds. Firstly, we adopt the rationale behind U_0 as follows. In the current partial solution we replace each currently unassigned job j ($j = k+1, \ldots, n$) by t_j jobs of length 1 and apply the LPT-rule to assign the T_{rem}^k jobs of length 1 to the current partial solution. Clearly, the resulting minimum completion time constitutes a local upper bound which we denote by U_0^{mod}.

Secondly, we partition $\mathcal{M}$ into two subsets $M_{UB^-} = \{M_i \in \mathcal{M} : C_i^k < UB_{loc}\}$ and $M_{UB^+} = \mathcal{M} \setminus M_{UB^-}$ where UB_{loc} denotes the best known local upper bound for the considered node. Note that UB_{loc} is the minimum of U_0^{mod} and the parent node's upper bound value. Then, we compute the following modified variant U_1^{mod} of U_1:

$$U_1^{mod} = \left\lfloor \frac{T_{sum} - \sum_{i \in M_{UB^+}} C_i^k - L_3}{|M_{UB^-}| - 1} \right\rfloor. \tag{4.25}$$

Here, L_3 is computed for a transformed instance restricted to (i) the machines in M_{UB^-}, (ii) the $n-k$ remaining jobs, and (iii) $|M_{UB^-}|$ fictitious jobs having processing times C_i^k ($i \in M_{UB^-}$).

To possibly improve on the lower bound, we apply the LPT-rule to partial solutions for which at least m jobs have already been assigned and no more than $2m$ jobs remain unassigned.

4.4.4 The branching scheme

We implemented a depth-first branching scheme which has originally been proposed by Dell'Amico and Martello (1995) in the context of solving $P||C_{max}$. We decided on this scheme as it allows for a straightforward incorporation of the path-related dominance criteria. It is different from the one developed by Haouari and Jemmali (2008a) and works as follows. At level k, the current node generates at most m son-nodes by sequentially assigning job k, i.e. the longest unassigned job, to each machine M_i that fulfills both $C_i^{k-1} < UB_{loc}$ and $|M_i| < j_{max}$. The corresponding machines are selected according to increasing current completion times.
Clearly, each new incumbent solution updates the global lower bound. As soon as an optimal solution has been identified, the branching process stops immediately.

4.4.5 Dominance criteria

In case that a current partial solution at level k cannot be fathomed due to the computation of the local upper bounds and not each pair of machines currently fulfills its path-condition, we apply the path-related dominance criteria in the same order as they are introduced in Sect. 4.3.

Furthermore, we apply the following criterion derived from Sect. 4.4.1.5.

Criterion 4.4.2
If $|M_i| + \bar{v}_i^k > j_{max}$ *for some* $i \in \{1, \ldots, m\}$*, then the current partial solution can be fathomed.*

The branching process itself can be limited according to four dominance criteria that have originally been introduced by Dell'Amico and Martello (1995). In Walter and Lawrinenko (2014) it is shown, how these four criteria have to be modified in order to be compatible with the path-related criteria. For further details we refer to the literature.

4.5 Computational study

We have coded the algorithm described within Sect. 4.4 in C++ using the Visual C++ 2010 compiler and experimentally tested its performance on (i) various difficult sets of $P||C_{min}$-instances as reported by Haouari and Jemmali (2008a) and (ii) a large set of instances from the literature. Our computational experiments were performed on a personal computer with an Intel Core i7-2600 processor and 8GB RAM while running Windows 7 Professional SP 1 (64-bit). The maximal computation time per instance was set to 600 seconds.

4.5.1 Performance on Dell'Amico and Martello's instances

In a first experiment, we have run our algorithm (denoted by WWL) on the following five problem classes originally proposed by Dell'Amico and Martello (1995):

- Class 1: discrete uniform distribution on $[1, 100]$
- Class 2: discrete uniform distribution on $[20, 100]$
- Class 3: discrete uniform distribution on $[50, 100]$
- Class 4: cut-off normal distribution with $\mu = 100$ and $\sigma = 50$
- Class 5: cut-off normal distribution with $\mu = 100$ and $\sigma = 20$

According to the results stated in the paper by Haouari and Jemmali (2008a), with their algorithm (denoted by HJ) branching was required for only 166 out of 2,050 instances. Since our global bounds are in general at least as strong as theirs, there is no need to reconsider all of their investigated constellations. Instead, we restricted our study to the difficult (n, m)-constellations $(10, 3), (10, 5), (25, 10), (25, 15), (50, 15)$ where branching was required by HJ. Like Haouari and Jemmali (2008a), for each constellation we randomly generated 10 independent instances resulting in a total of 250 instances. A summary of the

results is provided in Table 4.4. In this table we document the mean CPU time in seconds (labeled as "Time") as well as the mean number of explored nodes ("NN"). Moreover, the number of unsolved instances if greater than 0 is given in brackets. The results of algorithm HJ are taken from Haouari and Jemmali (2008a). Their algorithm was coded in Visual C++ 6.0 and ran on a Pentium IV 3.2GHz with 3GB RAM. The time limit for HJ was set to 800 seconds.
Additionally, in Table 4.4, we report on the performance of the new upper and lower bound techniques (cf. columns labeled as "UB_{impr}" and "LB_{MS}") introduced within the sections 4.4.1.4, 4.4.1.5, and 4.4.2. Concerning the new upper bounds, the column labeled as "#" gives the number of instances where the new upper bounds improved on the best upper bound obtained by the existing procedures (cf. Sections 4.4.1.1–4.4.1.3). Numbers in brackets indicate for how many instances (if less than 10) the application of at least one of the two new upper bounding techniques has been needed. The columns labeled as "avg" and "max" give the average and maximum relative deviation (in %) between the best existing upper bound and the best new upper bound. Concerning the new lower bound, the column labeled as "#" gives the number of instances where Multi-Subset (MS) generates the best lower bound value. Numbers in brackets indicate for how many instances (if less than 10) the application of MS has been needed. For those cases, the columns labeled as "avg" and "max" give the average and maximum relative deviation (in %) between the best lower bound value and the value produced by MS.

			HJ		WWL							
							UB_{impr}			LB_{MS}		
n	m	Class	Time	NN	Time	NN	#	avg	max	#	avg	max
10	3	1	0.001	90	0.016	21	0	0	0	4	2.307	5.645
		2	0.001	137	0.010	13	1	0.054	0.543	4	1.627	2.183
		3	0.002	48	0.020	16	0	0	0	3	1.161	3.309
		4	0.002	215	0.011	6	2	0.095	0.610	5	1.342	2.215
		5	0.002	100	0.012	30	0	0	0	3	1.265	2.652
10	5	1	0.002	128	0.010	3	5 (9)	3.179	13.084	4 (5)	0.323	1.613
		2	0.002	152	0.009	3	9	5.573	12.598	1 (3)	10.149	19.200
		3	<0.001	188	0.015	2	9	1.868	3.472	0 (1)	4.138	4.138
		4	0.003	150	0.009	2	9	2.939	8.938	0 (1)	7.292	7.292
		5	0.002	140	0.009	5	8	7.093	18.239	1 (4)	4.968	10.329
25	10	1	85.000	19,649,828	0.115	3,534	0	0	0	0	3.300	5.185
		2	129.000 (1)	31,675,367	0.083	1,965	5	0.325	0.704	0	5.226	8.725
		3	1.380	235,178	0.027	34	3	0.340	1.143	0	3.308	5.650
		4	19.625	4,294,945	0.018	13	5	1.001	3.196	0	1.830	3.644
		5	38.675	10,014,760	1.599	53,203	5	0.708	3.968	0	5.412	8.696
25	15	1	128.000 (1)	24,572,119	0.019	26	1 (5)	0.606	3.030	1 (5)	10.868	28.000
		2	1.900	268,948	0.009	4	1 (1)	5.952	5.952	0 (1)	2.500	2.500
		3	0.006	42	0.023	1	0 (0)	–	–	0 (0)	–	–
		4	77.878	14,537,221	0.009	1	0 (0)	–	–	0 (0)	–	–
		5	0.005	48	0.010	22	0 (1)	0	0	0 (1)	1.515	1.515
50	15	1	0.003	17	4.050 (1)	95,149	0	0	0	0	6.963	12.500
		2	0.001	21	0.015 (1)	1	0	0	0	1	3.021	6.283
		3	0.001 (2)	1	104.883 (6)	2,740,736	0	0	0	0	4.740	6.996
		4	20.335	4,498,587	93.392 (2)	1,961,145	0	0	0	0	5.683	6.548
		5	0.001 (2)	1	94.611 (4)	2,146,798	0	0	0	0	5.883	15.805

Table 4.4: Results on difficult instances generated according to Dell'Amico and Martello (1995)

As can be seen from the entries in Table 4.4, except for constellation $(50, 15)$ where algorithm HJ performed generally better than WWL, the opposite is true for each of the four other constellations. Here, our algorithm is strictly superior to algorithm HJ in terms of NN. Moreover, WWL was able to quickly solve all 200 instances while algorithm HJ

failed to solve two of them. The dominance of WWL is particularly impressive for the constellations $(25, 10)$ and $(25, 15)$ where the average number of generated nodes and the computation time were reduced to fractions of up to $1/14,500,000$ and $1/8,650$, respectively. However, the benefit of the dominance criteria (cf. Sect. 4.3) seems to be limited to constellations where $n/m \leq 2.5$. These observations comply with the findings in Walter and Lawrinenko (2014) for problem $P||C_{max}$.

The performance of the new bounding techniques can be summarized as follows. Starting with the upper bounds, application of the new techniques has been required for 206 out of 250 instances and improvements were achieved for 63 of them (mostly due to the bounds derived from BCP). While the overall average relative improvement is about 1.153%, maximum relative improvements of up to almost 19% were obtained (cf. constellation $(10, 5)$, Class 5). The new upper bounding techniques performed remarkably well at constellation $(10, 5)$ where upper bounds could be tightened for 40 (of 49) instances.
With regard to the new lower bound, application of the MS-heuristic has been required for 171 out of 250 instances (note that for the remaining 79 instances the LPT-solution had already been identified as an optimal solution). For 27 of the 171 instances, MS generated the best lower bound or, in other words, the MSLS-heuristic was not able to improve the MS-solution. All in all, our proposed construction heuristic MS performed quite well yielding a deviation from the MSLS-value of less than 3.5% on average.

4.5.2 Performance on Haouari and Jemmali's instances

In the second experiment, we have run our algorithm on a class of problems where the processing times are drawn from a discrete uniform distribution on $[1, n]$ as proposed by Haouari and Jemmali (2008a). Again, we restricted our study to those (n, m)-constellations where not all of the 10 generated instances in Haouari and Jemmali (2008a) had been solved at the root node by their algorithm. This time, these are the three constellations $(10, 5)$, $(25, 10)$, and $(25, 15)$. For each of them we randomly generated 10 instances. Table 4.5 provides a comparison of our results with the ones reported in Haouari and Jemmali (2008a).

		HJ		WWL							
						UB_{impr}			LB_{MS}		
n	m	Time	NN	Time	NN	#	avg	max	#	avg	max
10	5	<0.001	101	0.013	1	0 (0)	–	–	0 (0)	–	–
25	10	21.165	3,954,509	0.017	169	0 (6)	0	0	1 (6)	4.163	6.250
25	15	73.786	12,301,985	0.014	15	1 (4)	1.191	4.762	1 (3)	9.450	13.636

Table 4.5: Results on instances generated according to Haouari and Jemmali (2008a)

Obviously, our findings of the first experiment also apply to the results obtained for the second experiment (see Table 4.5) which reveal a clear dominance of our algorithm for the constellations $(25, 10)$ and $(25, 15)$. Compared to algorithm HJ, our approach reduced the average number of generated nodes and the computation time to fractions of up to $1/820,000$ and $1/5,270$, respectively.

4.5.3 Performance on benchmark instances

In a third experiment, we investigated WWL's performance on 780 benchmark instances originally proposed by França et al. (1994) as well as Frangioni et al. (2004) for problem $P||C_{max}$. While the former set consists of 390 uniform instances, the latter contains 390 non-uniform instances. For a detailed description, we refer to the literature. All instances can be downloaded from http://www.or.deis.unibo.it/ research.html. To the best of our knowledge, we are the first to use this established benchmark set in the context of $P||C_{min}$.

Table 4.6 reports on the results obtained by our algorithm. In addition to the mean CPU time and the mean number of nodes, this time we also document the average as well as maximum relative deviation in % (labeled as "Gap" and "maxGap", respectively) between the best upper bound value and the best lower bound value computed at the root node. Averages are taken over all 10 instances per triple $(n, m, Interval)$.

			Uniform				Non-Uniform			
n	m	Interval	Time	NN	Gap	maxGap	Time	NN	Gap	maxGap
10	5	$[1, 10^2]$	0.011	5	0.820	5.747	0.015	1	0	0
		$[1, 10^3]$	0.009	5	0.681	6.570	0.010	1	0	0
		$[1, 10^4]$	0.034	7	2.643	9.320	0.010	1	0	0
50	5	$[1, 10^2]$	0.009	1	0	0	– (10)	–	3.078	3.807
		$[1, 10^3]$	0.017	1	0	0	– (10)	–	3.561	4.224
		$[1, 10^4]$	0.125	1	0	0	– (10)	–	3.605	4.274
	10	$[1, 10^2]$	0.012	1	0	0	– (10)	–	13.826	17.085
		$[1, 10^3]$	0.016 (5)	1	0.021	0.044	– (10)	–	14.397	17.485
		$[1, 10^4]$	– (10)	–	0.016	0.025	– (10)	–	14.471	17.561
	25	$[1, 10^2]$	0.192	3,881	3.280	10.000	0.009	1	0	0
		$[1, 10^3]$	3.124 (3)	29,326	5.292	16.404	0.021	1	0	0
		$[1, 10^4]$	25.984 (5)	17,577	4.850	10.069	0.103	1	0	0
100	5	$[1, 10^2]$	0.009	1	0	0	0.017	1	0	0
		$[1, 10^3]$	0.033	1	0	0	0.117 (1)	1	0.001	0.005
		$[1, 10^4]$	0.182	1	0	0	2.236	1	0	0
	10	$[1, 10^2]$	0.012	1	0	0	– (10)	–	3.460	4.566
		$[1, 10^3]$	0.037	1	0	0	– (10)	–	3.813	4.813
		$[1, 10^4]$	0.335	1	0	0	– (10)	–	3.859	4.856
	25	$[1, 10^2]$	0.015	1	0	0	0.035 (6)	1	12.560	22.923
		$[1, 10^3]$	– (10)	–	0.075	0.112	– (10)	–	21.047	23.617
		$[1, 10^4]$	– (10)	–	0.051	0.087	– (10)	–	21.127	23.741
500	5	$[1, 10^2]$	0.006	1	0	0	0.085	1	0	0
		$[1, 10^3]$	0.334	1	0	0	0.567	1	0	0
		$[1, 10^4]$	2.679	1	0	0	5.293	1	0	0
	10	$[1, 10^2]$	0.019	1	0	0	0.127	1	0	0
		$[1, 10^3]$	0.395	1	0	0	0.787	1	0	0
		$[1, 10^4]$	2.418	1	0	0	4.989	1	0	0
	25	$[1, 10^2]$	0.058	1	0	0	0.114	1	0	0
		$[1, 10^3]$	0.443	1	0	0	1.076	1	0	0
		$[1, 10^4]$	3.027	1	0	0	7.302	1	0	0
1000	5	$[1, 10^2]$	0.008	1	0	0	0.304	1	0	0
		$[1, 10^3]$	0.434	1	0	0	1.825	1	0	0
		$[1, 10^4]$	9.559	1	0	0	19.402	1	0	0
	10	$[1, 10^2]$	0.011	1	0	0	0.220	1	0	0
		$[1, 10^3]$	1.044	1	0	0	1.798	1	0	0
		$[1, 10^4]$	8.543	1	0	0	17.481	1	0	0
	25	$[1, 10^2]$	0.063	1	0	0	0.373	1	0	0
		$[1, 10^3]$	1.630	1	0	0	3.568	1	0	0
		$[1, 10^4]$	8.857	1	0	0	24.293	1	0	0

Table 4.6: Results on benchmark instances

The results indicate that WWL's performance on the uniform instances is better than its performance on the non-uniform instances. While 347 of 390 uniform instances have

been solved optimally, only 273 of 390 non-uniform instances have been solved optimally resulting in a total number of 620 solved benchmark instances within the time limit of 600 seconds per instance. The unsolved uniform instances belong to the three (n, m)-constellations $(50, 10)$, $(50, 25)$, and $(100, 25)$ and the intervals $[1, 10^3]$ and $[1, 10^4]$. In contrast, all but one of the unsolved instances of the non-uniform set belong to the four (n, m)-constellations $(50, 5)$, $(50, 10)$, $(100, 10)$, and $(100, 25)$ and all three intervals. Taking a look at the entries in the columns Gap and maxGap of the unsolved instances, we record considerably smaller values for the uniform than for the non-uniform instances. For the latter, we observed average and maximum deviations of up to 21.1% and 23.7%, respectively. This indicates that non-uniform instances seem to be more difficult to solve than uniform instances.

To allow for future comparisons, we document detailed results on the best found objective function value as well as the best found upper bound value for each of the 780 instances in Appendix B.

4.6 Conclusions

The present Chapter addressed the machine covering problem $P||C_{min}$ and its exact solution. We identified structural properties of optimal schedules from which we deduced the so called *path-related* dominance criteria. Representing the key characteristic of the proposed branch-and-bound algorithm, these novel criteria proved to be effective in limiting the search space – particularly in the case of rather small ratios of n to m. For those constellations our approach is superior to the one presented in Haouari and Jemmali (2008a).

For future research on $P||C_{min}$ we suggest three interesting directions with regard to exact as well as heuristic procedures. Firstly, a quite promising advancement of our branch-and-bound algorithm might consist in the implementation of a sophisticated branching scheme that (even) more directly exploits the structural properties of potentially (unique) optimal solutions and thus enforces the generation of respective (partial) solutions. Secondly, we encourage the development of an innovative dynamic programming (DP) formulation of $P||C_{min}$ that makes use of the structural properties identified in this Chapter to trim the state space of the DP aiming at a possible improvement on the time complexity or the space requirements of the sole PTAS existing so far. Thirdly, due to the exponential nature of exactly solving the machine covering problem, efficient (meta-) heuristic approaches such as Tabu search or population-based algorithms are still needed to tackle large-sized instances. Here, it is conceivable to punish the non-fulfillment of the path-conditions by adequately reducing the fitness value of such solutions.

Chapter 5

A note on minimizing the normalized sum of squared workload deviations on m parallel processors

Summary

In this note we provide a counter-example to a central result by Ho et al. (2009) who proved that a schedule which minimizes the normalized sum of squared workload deviations is necessarily a makespan-optimal one. We explain why their proof is incorrect and present some computational results revealing the difference between workload balancing and makespan minimization.

5.1 Introduction

Given a set $\mathcal{J}$ of n independent jobs with positive integer processing times t_j ($j = 1, \ldots, n$) and a set $\mathcal{M}$ of m identical parallel machines, the workload balancing problem introduced by Ho et al. (2009) consists in assigning each job to a machine so that the normalized sum of squared workload deviations (denoted as *NSSWD*) is minimized. More formally, the performance measure NSSWD is defined as $[\sum_{i=1}^{m}(C_i - \mu)^2]^{1/2}/\mu$ where C_i denotes the completion time of machine i and μ is the average machine completion time, i.e., $\mu = \sum_{i=1}^{m} C_i/m$.

Ho et al. (2009) discussed some properties of the NSSWD measure and settled the complexity of the problem. Based on their result that "a non makespan-optimal schedule can be improved in terms of NSSWD by reduction in its maximum machine completion time" (cf. Proposition 3), Ho et al. (2009) concluded that (i) "a NSSWD-optimal schedule is necessarily a makespan-optimal schedule" (cf. Proposition 4) and (ii) problem $P||$NSSWD is $\mathcal{NP}$-hard. Furthermore, to solve problem $P||$NSSWD, Ho et al. (2009) proposed an algorithm – called Workload Balancing (WB) – that builds on existing procedures for makespan minimization.

5.2 A counter-example and corrected results

Clearly, in case of $m = 2$ machines, the above-mentioned result (cf. Proposition 4 in Ho et al., 2009) is correct as minimizing NSSWD is equivalent to makespan-minimization. However, in the general case of $m \geq 3$ machines the equivalence of the two criteria does not hold any longer. Additionally, in contrast to Proposition 4, NSSWD-optimality does not imply makespan-optimality as demonstrated by the following counter-example consisting of three machines and six jobs with processing times $t_1 = 18$, $t_2 = 13$, $t_3 = 11$, $t_4 = 9$, $t_5 = 8$, and $t_6 = 7$. The corresponding optimal solutions – which are unique for each of the two criteria – are illustrated in Fig. 5.1. While the makespan of the NSSWD-optimal

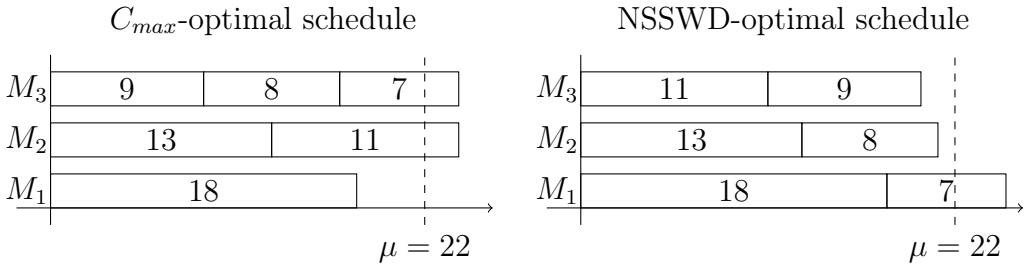


Figure 5.1: A counter-example for $m = 3$ machines.

schedule is 25, the optimal makespan is 24. Conversely, the makespan-optimal solution's NSSWD-value is about 0.2227 while the optimal NSSWD-value is about 0.1701. Using this counter-example, one readily obtains a counter-example for any fixed number of machines $m \geq 4$ by adding $m - 3$ jobs of length 22 each.

Next, we briefly explain why the proof of Proposition 3 in Ho et al. (2009) is incomplete and thus incorrect. In their proof, the authors merely showed that the NSSWD-value of a non makespan-optimal schedule S decreases if S is modified to S' where (i) the makespan of S' is smaller than the makespan of S and (ii) the modification affects exactly two machines while all other machine completion times remain unchanged. Indeed, this part is correct. However, as can be seen from the right-hand side of Fig. 5.1, in some situations a reduction in the makespan is only achievable if more than two machines are involved in the modification. This is the crucial point which had not been regarded by Ho et al. (2009). We shall also remark that the mistake seems to be unrecognized in a proceeding work by Cossari et al. (2012) who also claimed that NSSWD-optimality implies C_{max}-optimality.

As a consequence of the incorrectness of Proposition 3 and 4, one cannot conclude that problem $P||$NSSWD is $\mathcal{NP}$-hard because problem $P||C_{max}$ is $\mathcal{NP}$-hard. However, $P||$NSSWD is indeed $\mathcal{NP}$-hard as can be shown by a straightforward reduction from PARTITION which is well-known to be $\mathcal{NP}$-complete (cf. Garey and Johnson, 1979).

5.3 Computational study

As Ho et al. (2009) claimed a positive correlation between the NSSWD and the makespan criterion, they proposed a heuristic algorithm which bases upon existing algorithms for

makespan minimization. In light of the presented counter-example it is questionable whether their approach is meaningful. Addressing this issue, we conducted some experiments of the following kind. For a given instance we determined all NSSWD-optimal solutions as well as all makespan-optimal solutions (except for permutations of the machines) via complete enumeration. In case that at least one NSSWD-optimal solution is not makespan-optimal, we computed the relative deviation between the makespan of a NSSWD-optimal solution and the optimal makespan averaged over all NSSWD-optimal solutions. Furthermore, in case that at least one makespan-optimal solution is not NSSWD-optimal, we computed the respective average deviation from the optimal NSSWD value (analogous to the former case).

Due to the exponential nature of our computational study, we merely tested a few rather small (m, n)-constellations (cf. Tab. 5.1 and 5.2) in order to obtain reliable results. Depending on m and n we randomly generated 10^a independent instances. The processing times are drawn from a discrete uniform distribution on $[1, d]$. The parameter a ranges between 2 and 6 and six different d-values are studied, namely $d = 50, 100, 300, 500, 1000, 10000$ (cf. Tab. 5.1 and 5.2).

The following two tables summarize the results of our computational study. Tab. 5.1

		d	50	100	300	500	1000	10000
m	n	a						
3	6	6	0.52 (2.39)	0.64 (1.98)	0.75 (1.75)	0.77 (1.67)	0.79 (1.64)	0.80 (1.61)
	7	6	0.83 (1.84)	1.19 (1.45)	1.53 (1.17)	1.58 (1.12)	1.65 (1.08)	1.71 (1.05)
	8	6	0.97 (1.36)	1.73 (1.01)	2.48 (0.75)	2.69 (0.71)	2.84 (0.67)	2.98 (0.64)
	9	5	0.82 (1.04)	1.95 (0.72)	3.54 (0.50)	3.95 (0.45)	4.31 (0.41)	4.66 (0.39)
	10	5	0.41 (0.84)	1.59 (0.54)	3.99 (0.32)	4.93 (0.29)	5.56 (0.25)	6.29 (0.23)
	11	4	0.12 (0.67)	0.82 (0.45)	3.48 (0.21)	4.98 (0.18)	6.04 (0.16)	8.33 (0.13)
	12	4	0.02 (0.59)	0.23 (0.28)	2.88 (0.15)	4.20 (0.13)	6.57 (0.10)	9.36 (0.08)
	13	3	0.00 (0.00)	0.00 (0.00)	1.60 (0.11)	3.20 (0.10)	5.00 (0.06)	11.80 (0.04)
	14	3	0.00 (0.00)	0.00 (0.00)	0.20 (0.13)	0.90 (0.07)	3.70 (0.05)	10.20 (0.02)
	15	3	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.30 (0.04)	2.90 (0.03)	12.00 (0.01)
	16	2	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	3.00 (0.03)	13.00 (0.01)
4	7	6	0.38 (2.49)	0.50 (2.05)	0.57 (1.75)	0.59 (1.71)	0.59 (1.66)	0.63 (1.58)
	8	5	1.20 (2.41)	1.51 (1.93)	1.79 (1.63)	1.88 (1.61)	1.87 (1.54)	1.97 (1.52)
	9	5	1.70 (1.97)	2.36 (1.50)	2.93 (1.26)	3.08 (1.17)	3.24 (1.15)	3.37 (1.12)
	10	4	1.96 (1.50)	3.05 (1.15)	4.45 (0.83)	4.97 (0.79)	4.79 (0.78)	5.06 (0.71)
	11	3	2.00 (0.99)	3.60 (0.89)	5.70 (0.50)	7.50 (0.52)	7.40 (0.49)	8.40 (0.54)
	12	3	1.30 (0.70)	3.90 (0.64)	5.90 (0.37)	9.70 (0.32)	9.70 (0.37)	11.00 (0.31)
	13	2	0.00 (0.00)	4.00 (0.37)	9.00 (0.32)	13.00 (0.23)	7.00 (0.44)	12.00 (0.11)
5	8	5	0.26 (2.52)	0.32 (2.00)	0.35 (1.82)	0.35 (1.74)	0.34 (1.63)	0.42 (1.60)
	9	4	0.85 (2.26)	1.21 (1.91)	1.29 (1.81)	1.27 (1.54)	1.47 (1.51)	1.52 (1.41)
	10	4	1.92 (2.37)	2.15 (1.95)	2.59 (1.67)	2.87 (1.49)	2.95 (1.47)	3.17 (1.44)
	11	3	1.80 (1.70)	3.80 (1.64)	4.00 (1.16)	5.00 (1.22)	4.00 (1.33)	4.60 (1.23)
	12	2	3.00 (1.49)	2.00 (1.18)	6.00 (1.30)	2.00 (1.69)	6.00 (1.03)	8.00 (0.42)

Table 5.1: Results for NSSWD-optimal schedules concerning the makespan criterion (in %)

concerns NSSWD-optimal solutions with respect to their performance for the makespan criterion. The first entry in each table cell provides the rate of instances where not all NSSWD-optimal solutions are makespan-optimal. For those instances, the entries in brackets give the average relative deviations from the optimal makespan. In short, our results reveal that NSSWD-optimal solutions perform generally very well with respect to the makespan criterion in terms of relative deviations (cf. entries in brackets in Tab. 5.1) although we identified constellations where the probability that not all NSSWD-optimal

solutions are makespan-optimal is larger than 10%.

Tab. 5.2 concerns makespan-optimal solutions with respect to their performance for the NSSWD criterion. Following the layout of Tab. 5.1, the first entry in each table cell provides the rate of instances where not all makespan-optimal solutions are NSSWD-optimal. For those instances, the entries in brackets give the average relative deviations from the optimal NSSWD value. As can be seen from Tab. 5.2, the first entry in each table

		d	50	100	300	500	1000	10000
m	n	a						
3	6	6	57.68 (21.64)	58.94 (19.58)	59.67 (18.27)	59.82 (17.99)	59.92 (17.79)	60.09 (17.59)
	7	6	57.02 (25.13)	58.26 (22.03)	59.02 (20.00)	59.23 (19.58)	59.38 (19.27)	59.46 (19.01)
	8	6	54.58 (29.29)	55.90 (24.64)	56.33 (21.54)	56.53 (20.93)	56.71 (20.45)	56.78 (19.99)
	9	5	51.82 (34.43)	53.11 (27.73)	53.46 (22.99)	53.33 (21.96)	53.71 (21.17)	53.31 (20.48)
	10	5	48.28 (39.62)	51.22 (31.74)	50.34 (24.64)	50.48 (22.98)	50.55 (21.71)	50.11 (20.64)
	11	4	42.88 (44.21)	48.28 (37.50)	47.66 (27.56)	47.58 (24.48)	47.04 (22.46)	46.63 (20.25)
	12	4	36.90 (47.61)	44.06 (42.82)	47.36 (31.48)	46.80 (26.95)	45.44 (23.57)	43.50 (20.75)
	13	3	34.90 (49.15)	39.60 (48.46)	43.90 (37.78)	47.30 (31.19)	45.90 (25.83)	43.50 (20.89)
	14	3	34.00 (49.62)	35.10 (48.56)	42.90 (42.96)	45.10 (37.08)	44.20 (29.45)	39.60 (21.05)
	15	3	33.10 (50.49)	34.90 (50.74)	37.90 (45.21)	43.80 (42.00)	42.10 (36.88)	38.70 (20.88)
	16	2	32.00 (50.26)	31.00 (50.43)	35.00 (47.86)	39.00 (45.53)	46.00 (37.54)	35.00 (22.93)
4	7	6	80.87 (26.75)	82.00 (24.93)	82.73 (23.75)	82.92 (23.49)	82.93 (23.30)	83.13 (23.17)
	8	5	82.14 (29.83)	83.22 (27.33)	84.23 (25.67)	84.07 (25.38)	84.34 (25.11)	84.50 (24.85)
	9	5	81.16 (34.41)	82.55 (30.73)	83.37 (28.15)	83.59 (27.67)	83.50 (27.31)	83.73 (27.08)
	10	4	79.27 (39.37)	80.32 (33.48)	80.94 (29.87)	81.47 (28.85)	81.17 (28.48)	80.60 (28.06)
	11	3	75.90 (43.56)	79.30 (35.38)	80.40 (32.30)	79.00 (31.10)	78.90 (29.45)	79.80 (27.19)
	12	3	70.00 (49.99)	73.80 (40.40)	75.50 (32.60)	76.80 (31.92)	76.00 (28.85)	72.30 (27.71)
	13	2	67.00 (53.79)	78.00 (41.19)	70.00 (40.13)	72.00 (27.61)	81.00 (24.35)	74.00 (25.47)
5	8	5	89.94 (33.94)	90.56 (32.24)	90.87 (31.10)	90.91 (30.95)	90.80 (30.65)	90.84 (30.48)
	9	4	92.02 (34.30)	93.07 (32.19)	92.83 (30.49)	92.93 (30.47)	93.09 (30.27)	92.82 (30.34)
	10	4	92.80 (37.77)	93.43 (34.97)	93.75 (32.79)	93.69 (32.62)	93.48 (32.30)	93.98 (31.97)
	11	3	92.60 (43.63)	93.20 (39.43)	93.40 (34.78)	93.60 (36.06)	92.90 (34.94)	94.80 (35.95)
	12	2	95.00 (49.71)	93.00 (40.95)	91.00 (37.14)	88.00 (38.85)	90.00 (39.88)	92.00 (38.47)

Table 5.2: Results for makespan-optimal schedules concerning the NSSWD criterion (in %)

cell is considerably larger compared to those in Tab. 5.1, i.e., with high probability (in some cases about 95%) not all makespan-optimal solutions are NSSWD-optimal. Regarding the entries in brackets, at least for small ratios of n to m we believe that any makespan-optimal schedule is not necessarily "a good candidate to be close to an optimal or near-optimal NSSWD solution" as stated in Ho et al. (2009). Nevertheless, we suppose that generally at least one of the makespan-optimal solutions performs moderately well for the NSSWD criterion.

Chapter 6

Reduction criteria, upper bounds, and a dynamic programming based heuristic for the k_i-partitioning problem

Summary

This Chapter addresses the k_i-partitioning problem that asks for an assignment of n jobs to m parallel machines so that the minimum machine completion time is maximized and the number of jobs on each machine does not exceed a machine-dependent cardinality limit k_i $(i = 1, \ldots, m)$. We propose different preprocessing as well as lifting procedures and derive several upper bound arguments. Furthermore, we introduce suited construction heuristics as well as an effective dynamic programming based improvement procedure. Results of a comprehensive computational study on a large set of randomly generated instances indicate that our algorithm quickly finds (near-)optimal solutions.

6.1 Introduction

6.1.1 Problem definition

In this Chapter we investigate the k_i-partitioning problem where we are given a set $\mathcal{M}$ of $m \geq 2$ parallel machines, each having an associated machine-dependent cardinality limit k_i $(i = 1, \ldots, m)$ on the maximal number of jobs that can be processed by machine i, and a set $\mathcal{J}$ of n jobs $(m < n \leq \sum_{i=1}^{m} k_i)$ with positive integer processing times $t_j \in \mathbb{N}$ $(j = 1, \ldots, n)$. Let C_i denote the completion time of machine i $(i = 1, \ldots, m)$, which is simply defined as the sum of the processing times of all jobs assigned to i, the objective is to find an assignment (or schedule) that maximizes the minimum machine completion time $C_{min} = \min\{C_1, \ldots, C_m\}$ without exceeding the cardinality limits. Without loss of generality, we assume the jobs and the machines to be labeled so that $t_1 \geq t_2 \geq \ldots \geq t_n > 0$ and $0 < k_1 \leq k_2 \leq \ldots \leq k_m$, respectively.

Introducing binary variables x_{ij} which take the value 1 if job j is assigned to machine i and 0 otherwise, a straightforward formulation of the k_i-partitioning problem as an integer

linear program consisting of objective function (6.1) subject to (6.2)–(6.5) is provided below.

$$\text{Maximize } C_{min} \tag{6.1}$$

$$\text{s.t. } \sum_{j=1}^{n} t_j \cdot x_{ij} \geq C_{min} \qquad i = 1, \ldots, m \tag{6.2}$$

$$\sum_{i=1}^{m} x_{ij} = 1 \qquad j = 1, \ldots, n \tag{6.3}$$

$$\sum_{j=1}^{n} x_{ij} \leq k_i \qquad i = 1, \ldots, m \tag{6.4}$$

$$x_{ij} \in \{0, 1\} \qquad i = 1, \ldots, m;\ j = 1, \ldots, n \tag{6.5}$$

Objective function (6.1) maximizes the minimum machine completion time C_{min}, which is determined by inequalities (6.2). Constraints (6.3) ensure that each job is assigned to exactly one machine and constraints (6.4) represent the machine-dependent cardinality limits. Finally, the domains of the binary variables are set by (6.5).

Obviously, the k_i-partitioning problem is a generalization of the classical machine covering problem $P||C_{min}$ which is obtained by dropping constraints (6.4) or, equivalently, by setting $k_i = n$ for all i (i.e. (6.4) become redundant). Since problem $P||C_{min}$ is well-known to be $\mathcal{NP}$-hard (cf. Haouari and Jemmali, 2008a), its generalized version with a limited number of jobs per machine is $\mathcal{NP}$-hard, too.

As mentioned by Dell'Amico et al. (2006), a possible application of the k_i-partitioning problem arises for instance in the context of flexible manufacturing systems. Here, n is the number of different types of operations, t_j represents the total time required to execute all operations of type j (which have to be assigned to the same cell), m is the number of cells, and k_i represents the capacity of the specific tool magazine of cell i, i.e. k_i restricts the number of types of operations cell i can perform. Another possible application arises in the context of fairly distributing investment projects among different regions (cf. Haouari and Jemmali, 2008a). Here, the task is to allocate n projects with individual revenues t_j to m regions so that the minimal total revenue of the regions is maximized. If we assume the regions to have individual (staff) capacities to manage and administrate the allocated projects, k_i represents the maximum number of projects that can be handled by region i.

6.1.2 Literature Review

To the best of the authors' knowledge, specific literature on parallel machine scheduling problems with (machine-dependent) cardinality limits is rather rare. In particular, so far there exists only one contribution to the k_i-partitioning problem with the objective of maximizing the minimum completion time. The contribution stems from He et al. (2003) who proposed an approximation algorithm (dubbed HARMONIC2) and studied its worst-case ratio. He et al. (2003) also treated the case where the cardinality limits of the machines are identical, i.e. $k_i = k$ for all $i = 1, \ldots, m$. The corresponding problem is called

k-partitioning. Chen et al. (2002) analyzed the worst-case performance of a modified LPT algorithm for 3-partitioning and a variant called Kernel 3-partitioning.

When the objective of maximizing the minimum completion time is altered into the more popular minimization of the maximum completion time (i.e. the makespan), we noticed just three contributions that address machine-dependent cardinality limits k_i. The first one is due to Dell'Amico et al. (2006) who provided reduction criteria, lower bound procedures, and a scatter search algorithm. In the second contribution, Zhang et al. (2009) used an extension of Jain's Iterative Rounding Method to obtain a polynomial time 3-approximation algorithm. The third contribution stems from Kellerer and Kotov (2011) who presented an elementary 3/2-approximation algorithm whose running time is linear in n.

Considering machine-independent cardinality limits, i.e. $k_i = k$ $(i = 1, \ldots, m)$, a few more papers exist. In their extensive study on the $\mathcal{NP}$-hard k-partitioning problem, Babel et al. (1998) derived different lower bound arguments and introduced several approximation algorithms along with their worst-case behaviors. Dell'Amico and Martello (2001) developed further lower bound procedures and investigated their worst-case performances. In a follow-up paper, Dell'Amico and Martello (2004) introduced heuristic and metaheuristic solution procedures such as a scatter search algorithm and they compared their computational performances with a branch-and-bound algorithm. In the special case $k = 3$, Kellerer and Woeginger (1993b) analyzed the worst-case performance of a modified version of the LPT algorithm and Kellerer and Kotov (1999) introduced a 7/6-approximation algorithm. The complexity of Kernel 3-partitioning and the worst-case performance of a modified LPT algorithm have been examined by Chen et al. (1996). Besides, Woeginger (2005) established the existence of a fully polynomial time approximation scheme (FPTAS) for the special case $m = 2$.

Regarding the balanced variant of the identical parallel machine scheduling problem with minimum makespan objective, i.e. where each k_i either equals $\lfloor n/m \rfloor$ or $\lceil n/m \rceil$, Tsai (1992) developed a heuristic algorithm for the case $m = 2$ and analyzed its asymptotic behavior. Tsai proved that the absolute difference between the optimal makespan and the heuristic makespan is bounded by $\mathcal{O}(\log n/n^2)$, almost surely, when the processing times are independently drawn from a uniform distribution on $[0, 1]$. Also for $m = 2$, Mertens (1999) proposed a complete anytime algorithm. Michiels et al. (2012) investigated the worst-case performance of Karmarkar and Karp's Differencing Method. They proved that the performance ratio is precisely $2 - 1/m$ for any fixed $m \geq 2$. When k is given instead of m, they showed that $2 - \sum_{i=0}^{k-1} i!/k!$ is a lower bound and $2 - 1/(k-1)$ is an upper bound on the performance ratio for any fixed k. By means of a novel approach in which the ratios are explicitly calculated using mixed integer linear programming, Michiels et al. (2012) also proved that their lower bound is tight for $k \leq 7$.

Returning to the objective of maximizing the minimum completion time, we finish our literature review with a selection of substantial contributions to the problem version where no cardinality limits are present, i.e. $P||C_{min}$. Problem $P||C_{min}$ has been first mentioned in Friesen and Deuermeyer (1981) and Deuermeyer et al. (1982) derived a bound on the worst-case performance of the LPT algorithm. Later, Csirik et al. (1992) tightened this bound. Woeginger (1997) presented a polynomial-time approximation scheme (PTAS), and Haouari and Jemmali (2008a) provided an exact branch and bound algorithm along

with tight upper and lower bound procedures. Recently, Walter (2013) examined the performance relationship between the LPT algorithm and its restricted version RLPT and Walter et al. (2016) developed improved approaches to the exact solution of $P||C_{min}$ including novel dominance rules and new upper bound procedures.

6.1.3 Contribution and Chapter structure

Motivated by the research gap in the field of upper and lower bound procedures for the k_i-partitioning problem where C_{min} is to be maximized, in this Chapter we provide new theoretical insights into the problem and propose different approaches towards its efficient solution. Our first major contribution concerns the cardinality limits for which we present procedures to tighten them. Here, we do not only focus on the explicitly given upper limits but also on the derivation of tight (implicit) lower cardinality limits. Secondly, we derive several upper bound arguments and two lifting procedures to tighten the bounds. Eventually, suited solution algorithms – such as fast LPT-based construction heuristics, an exact dynamic programming approach to solve the two-machine case, and a well-performing local search improvement algorithm for multiple machines – constitute the third part of our contribution.

The remainder of the Chapter is organized as follows. In Section 6.2, we present methods to preprocess a given problem instance. Lifting as well as upper bound procedures are developed in Section 6.3. Then, Section 6.4 introduces tailor-made construction and improvement heuristics, whose computational performance is tested in a comprehensive computational study (Section 6.5). Finally, Section 6.6 concludes the Chapter with a brief summary and ideas for future research.

6.2 Preprocessing

Preprocessing is a proved means to reduce the size of a problem instance, and thus the solution space, often resulting in tighter bounds and an enhanced performance of algorithms in terms of solution quality and/or computation time. In this section, we provide two different approaches to preprocess instances of the k_i-partitioning problem. While the first one aims at tightening the cardinality limits of the machines (Section 6.2.1), the second one intends to reduce the dimension of the problem by eliminating machines and jobs in advance (Section 6.2.2).

6.2.1 Tightening the cardinality limits

As the sum of the cardinality limits $\sum_{i=1}^{m} k_i$ can be greater than the number of jobs n (cf. Section 6.1), there is basically some potential to tighten the explicitly given upper limits k_i on the maximal number of jobs that can be processed by machine i. At the same time, lower limits l_i on the minimal number of jobs that have to be processed by machine i in any feasible schedule can be derived. Clearly, since we assume that $n > m$, in an optimal solution none of the machines will remain empty, i.e. we are implicitly given lower limits $l_i = 1$ $(i = 1, \ldots, m)$. In what follows, we show how to make use of the upper cardinality limits to tighten the lower cardinality limits and vice versa. Later on, in

Section 6.3 and 6.4, the enhanced cardinality limits will not only help us to establish tight upper bounds on the optimal objective value but also to generate high-quality solutions.

To shorten notation, for $1 \leq i_1 \leq i_2 \leq n$ we define

$$\begin{aligned} K_{i_1,i_2} &= \min\left\{\sum_{s=i_1}^{i_2} k_s,\ n - \sum_{s=1}^{i_1-1} l_s - \sum_{s=i_2+1}^{m} l_s\right\}, \\ L_{i_1,i_2} &= \max\left\{\sum_{s=i_1}^{i_2} l_s,\ n - \sum_{s=1}^{i_1-1} k_s - \sum_{s=i_2+1}^{m} k_s\right\} \end{aligned} \tag{6.6}$$

and set $K_{i_1,i_2} = L_{i_1,i_2} = 0$ in case $i_1 > i_2$. It is readily verified that K_{i_1,i_2} represents the maximum number of jobs that can be processed by the machine-subset $\{i_1, \ldots, i_2\}$: While the first term ($\sum_{s=i_1}^{i_2} k_s$) is due to their individual upper cardinality limits, the second term takes into account that the remaining machines have to process at least $\sum_{s=1}^{i_1-1} l_s + \sum_{s=i_2+1}^{m} l_s$ jobs in order to satisfy their lower limits. Using a similar argument reveals that L_{i_1,i_2} gives the minimum number of jobs that have to be processed on $\{i_1, \ldots, i_2\}$.

Having in mind that in each feasible solution at most $K_{i,m}$ jobs will be assigned to the last $m - i + 1$ machines, there exists at least one machine in $\{i, \ldots, m\}$ which cannot process more than $\lfloor K_{i,m}/(m - i + 1) \rfloor$ jobs. Moreover, since at most $K_{i,q}$ $(i \leq q \leq m)$ jobs will be assigned to $q - i + 1$ machines $\{i, \ldots, q\}$, there exists at least one machine among them which cannot process more than $\lfloor K_{i,q}/(q - i + 1) \rfloor$ jobs. As a consequence, we obtain the following decreased upper cardinality limits

$$k_i = \min_{q=i,\ldots,m} \left\{ \left\lfloor \frac{K_{i,q}}{q - i + 1} \right\rfloor \right\},\ i = 1, \ldots, m. \tag{6.7}$$

Note that if $K_{i,q} < \sum_{s=i}^{q} k_s$ for at least one $q \in \{i, \ldots, m\}$, then the upper limit of machine i can be decreased according to (6.7). In particular, for the first machine we receive $k_1 \leq \lfloor n/m \rfloor$ which is already quite clear from the fact that not every machine can process more than $\lfloor n/m \rfloor$ jobs.

Analogously, since the $i - q + 1$ machines $\{q, \ldots, i\}$ process at least $L_{q,i}$ jobs, there will be at least one machine among them which processes at least $\lceil L_{q,i}/(i - q + 1) \rceil$ jobs. Thus, increased lower cardinality limits are

$$l_i = \max_{q=1,\ldots,i} \left\{ \left\lceil \frac{L_{q,i}}{i - q + 1} \right\rceil \right\},\ i = 1, \ldots, m. \tag{6.8}$$

Another option to increase the lower and to decrease the upper cardinality limits, respectively, is described in Walter et al. (2016). Given a valid lower bound LB (see Section 6.4) on the optimal minimum completion time C^*_{min}, any improving solution has to process at least $\bar{l}$

$$\bar{l} = \arg\min_{h=1,\ldots,n} \left\{ \sum_{j=1}^{h} t_j > \mathrm{LB} \right\} \tag{6.9}$$

jobs on each machine, i.e $l_i = \max\left\{l_i, \bar{l}\right\}$ $(i = 1, \ldots, m)$. On the other hand, in an improving solution no machine can process more than

$$\bar{k} = \arg\max_{h=1,\ldots,n} \left\{ \sum_{j=n-h+1}^{n} t_j \leq \bar{C} \right\} \tag{6.10}$$

jobs, i.e. $k_i = \min\left\{k_i, \bar{k}\right\}$ $(i = 1, \ldots, m)$, where

$$\bar{C} = \arg\max_{C \in \mathbb{N}} \left\{ \left\lfloor \frac{\sum_{j=1}^{n} t_j - C}{m-1} \right\rfloor > \mathrm{LB} \right\}.$$

Note that if a machine's completion time exceeds $\bar{C}$, then it is not possible that each of the remaining $m-1$ machines runs longer than LB.

Taking into account that a job will not be processed by more than one machine, an enhanced version of (6.9) is

$$\bar{l}_i = \arg\min_{h=1,\ldots,n-i+1} \left\{ \sum_{j=i}^{h+i-1} t_j > \mathrm{LB} \right\} \tag{6.11}$$

and we obtain $l_i = \max\left\{l_i, \bar{l}_i\right\}$ $(i = 1, \ldots, m)$. The idea of disregarding the largest $i-1$ jobs when computing enhanced l_i-values bases upon the lifting procedure as described in Section 6.3.1 (see proof of Theorem 6.3.1). Analogously,

$$\bar{k}_{m-i+1} = \arg\max_{h=1,\ldots,n-i+1} \left\{ \sum_{j=n-h-i+2}^{n-i+1} t_j \leq \bar{C} \right\} \tag{6.12}$$

leads to tightened k_i-values by setting $k_{m-i+1} = \min\left\{k_{m-i+1}, \bar{k}_{m-i+1}\right\}$ $(i = 1, \ldots, m)$. Note that if $L_{1,m} > n$ or $K_{1,m} < n$ after application of (6.9)–(6.12), then we immediately obtain that LB equals the optimal objective value.

Remark 6.2.1
Assuming that the machines are sorted according to non-decreasing k_i-values and initializing the lower limits with $l_i = 1$, application of (6.7)–(6.12) will maintain the order of the machines, i.e. we still have $k_1 \leq \ldots \leq k_m$ and $l_1 \leq \ldots \leq l_m$.

6.2.2 Reduction Criteria

Reduction criteria play a crucial role when it comes to reduce a problem's size (or dimension) and thus the solution space. We start with a straightforward reduction criterion that has already been stated in Dell'Amico et al. (2006).

Criterion 6.2.2
If $k_1 = \ldots = k_{h'} = 1 < k_{h'+1}$, then there exists an optimal solution in which job 1 is processed by machine 1, job 2 by machine 2,..., and job h' by machine h'.

Our other two criteria exploit the lower cardinality limits and require a valid upper bound (denoted by UB) on C^*_{min}. Further details on how to compute upper bounds are to be found in Section 6.3.

Criterion 6.2.3
If the sum of the smallest l_m processing times is greater than or equal to UB, then there exists an optimal solution in which machine m solely processes the jobs $\{n - l_m + 1, \ldots, n\}$.

Clearly, if the condition stated in Criterion 6.2.3 is met, we can reduce the dimension of the problem to $m-1$ machines and $n-l_m$ jobs by fixing the assignment of the shortest l_m jobs to machine m in advance. Afterward, we can check if Criterion 6.2.3 also applies to the reduced problem. If this is the case, the problem's dimension reduces further. This process can be iterated until the condition is no longer fulfilled.
There are two minor drawbacks of the latter approach. First, within each iteration only a single machine is considered and second, the process immediately stops as soon as the condition stated in Criterion 6.2.3 is not fulfilled any longer. So, if for instance the sum of the smallest l_m processing times is already smaller than UB, then the criterion cannot be applied at all. To overcome these shortcomings, we propose an enhanced iterative reduction procedure as depicted in Figure 6.1. In iteration i, we consider the i machines with the largest lower cardinality limits simultaneously. According to their lower limits, these machines have to process at least $L_{m-i+1,m}$ jobs in total. Selecting the shortest $L_{m-i+1,m}$ jobs, we compute a lower bound (e.g. by application of one of our procedures introduced in Section 6.4) on C^*_{min} for the corresponding partial problem (denoted by $PP(\{m-i+1,\ldots,m\},\{n-L_{m-i+1,m}+1,\ldots,n\})$). All in all, the procedure seeks the largest i so that the respective lower bound is at least as large as UB. Then, clearly, we can feasibly reduce the dimension of the problem by removing the machines $m, m-1,\ldots,m-i+1$ and the shortest $L_{m-i+1,m}$ jobs. A similar reduction procedure for the "dual" k_i-partitioning problem with makespan objective is used in Dell'Amico et al. (2006).

1. $i := 1;\ l_{sum} := l_m;\ \tilde{i} := 0;\ \tilde{j} := 0$
2. **while** $i < m$
3. **if** $\mathrm{LB}(PP(\{m-i+1,\ldots,m\},\{n-L_{m-i+1,m}+1,\ldots,n\})) \geq \mathrm{UB}$
4. $\tilde{i} := i;\ \tilde{j} := L_{m-i+1,m}$
5. **end**
6. $i := i+1;\ l_{sum} := l_{sum} + l_{m-i+1}$
7. **end**
8. Remove machines $\{m-\tilde{i}+1,\ldots,m\}$ and jobs $\{n-\tilde{j}+1,\ldots,n\}$

Figure 6.1: Reduction procedure 1

For problem $P||C_{min}$, Walter et al. (2016) used another trivial reduction criterion which removes a machine and a job if the corresponding processing time is already greater than or equal to UB. Taking the cardinality constraints into account leads us to the following enhanced version of their criterion.

Criterion 6.2.4
If the sum of the largest processing time t_1 and the smallest l_1-1 processing times is greater than or equal to UB, then there exists an optimal solution in which the first machine solely processes the jobs $\{1, n-l_1+2,\ldots,n\}$.

The correctness of Criterion 6.2.4 is readily verified. As the longest job has to be processed by any machine and all lower cardinality limits are greater than or equal to l_1, the machine that processes job 1 has to process at least l_1-1 other jobs as well. Thus, its

completion time will be at least as large as $t_1 + \sum_{i=1}^{l_1-1} t_{n-i+1}$. Clearly, if the completion time is already greater than or equal to UB, then it is not meaningful to assign any other jobs than the shortest $l_1 - 1$ ones to the machine which processes job 1. As a consequence, the dimension of the problem can be reduced by one machine and l_1 jobs.
It is quite obvious, that Criterion 6.2.4 can also be repeatedly applied in a straightforward manner. However, this raises the same issues as with Criterion 6.2.3 so that here, too, we propose an enhanced iterative procedure as shown in Figure 6.2. This time, in iteration i, we simultaneously consider the first i machines, i.e. the ones with the smallest l_i-values, the i largest jobs, and the $L_{1,i}-i$ shortest jobs. For the corresponding partial problem (denoted by $PP(\{1,\ldots,i\},\{1,\ldots,i,n-L_{1,i}+i+1,\ldots,n\})$) a lower bound on C^*_{min} is determined. The procedure seeks the largest i so that the respective lower bound is greater than or equal to UB and removes the machines $1,\ldots,i$ and the jobs $1,\ldots,i,n-L_{1,i}+i+1,\ldots,n$.

1. $i := 1;\ l_{sum} := l_1;\ \tilde{i} := 0;\ \tilde{j} := 0$
2. **while** $i < m$
3. **if** $\mathrm{LB}(PP(\{1,\ldots,i\},\{1,\ldots,i,n-L_{1,i}+i+1,\ldots,n\})) \geq \mathrm{UB}$
4. $\tilde{i} := i;\ \tilde{j} := L_{1,i}$
5. **end**
6. $i := i+1;\ l_{sum} := l_{sum} + l_i$
7. **end**
8. Remove machines $\{1,\ldots,\tilde{i}\}$ and jobs $\{1,\ldots,\tilde{i},n-\tilde{j}+\tilde{i}+1,\ldots,n\}$

Figure 6.2: Reduction procedure 2

6.3 Bounding the optimal objective value

In this section we are concerned with methods to bound the optimal objective value of the k_i-partitioning problem from above. Specifically, we introduce two lifting procedures (Section 6.3.1) and derive several upper bound arguments (Section 6.3.2). The lifting procedures are used to tighten the upper bounds which in turn help us to benchmark our heuristics (see Section 6.4) when optimal solutions are not available.

6.3.1 Lifting procedures

The basic rationale behind lifting is to identify partial problem instances $PP(M,J)$ of a given instance $(\mathcal{M},\mathcal{J})$ where $M \subseteq \mathcal{M}$, $J \subseteq \mathcal{J}$, and the optimal objective value of $PP(M,J)$ is greater than or equal to the optimal value of the given (initial) instance. Then, by application of upper bound procedures to $PP(M,J)$ we also obtain upper bounds on the optimal value of the initial instance. Clearly, the more (non-trivial) partial problems we identify, the more likely we obtain a tighter upper bound for the initial instance. For this purpose, we next describe two specific approaches (cf. Corollary 6.3.2 and 6.3.3) to determine a set $\mathcal{P}(\mathcal{M},\mathcal{J})$ of partial problem instances $PP(M,J)$ satisfying the previously mentioned properties.

The first one uses the fact that the i longest jobs ($i \leq m-1$) cannot be assigned to more than i machines and the remaining jobs will be assigned to at least $m-i$ machines. For the computation of a feasible upper bound, the remaining jobs can be assumed to be assigned to the $m-i$ machines with the largest upper cardinality limits. Formally, we obtain the following theorem.

Theorem 6.3.1
For all $i = 1, \ldots, m$, the partial instance $PP(\{i, \ldots, m\}, \{i, \ldots, i + K_{i,m} - 1\})$ is an element of $\mathcal{P}(\mathcal{M}, \mathcal{J})$.

Proof
Since a job cannot be split among different machines, in any feasible schedule the longest $i-1$ jobs will be assigned to at most $i-1$ machines. Hence, there exist at least $m-i+1$ machines which do not process any of the first $i-1$ jobs. The maximum number of jobs that can be assigned to $m-i+1$ machines is $K_{i,m}$ which is obviously obtained by considering the last $m-i+1$ machines. Thus, the objective value of the optimal assignment of the job-set $\{i, \ldots, i+K_{i,m}-1\}$ to the machine-set $\{i, i+1, \ldots, m\}$ constitutes an upper bound for the initial problem, i.e. $PP(\{i, \ldots, m\}, \{i, \ldots, i + K_{i,m} - 1\}) \in \mathcal{P}(\mathcal{M}, \mathcal{J})$. ■

A generalization of Theorem 6.3.1 is provided by the next corollary.

Corollary 6.3.2
For all pairs (i_1, i_2) where $1 \leq i_1 \leq i_2 \leq m$, the partial instance $PP(\{i_1, \ldots, i_2\}, \{i_1, \ldots, i_1 + K_{i_1,i_2} - 1\})$ is an element of $\mathcal{P}(\mathcal{M}, \mathcal{J})$.

Proof
Clearly, the machine-set $\{i_1, \ldots, i_2\}$ can process at most K_{i_1,i_2} jobs in total. According to Theorem 6.3.1, there exists an optimal solution in which none of the jobs $1, \ldots, i_1 - 1$ is assigned to a machine whose index is greater than $i_1 - 1$. So, the K_{i_1,i_2} longest remaining jobs that can be processed on $\{i_1, \ldots, i_2\}$ are $\{i_1, \ldots, i_1 + K_{i_1,i_2} - 1\}$. Optimally assigning this job-set to the machine-set $\{i_1, \ldots, i_2\}$ yields an upper bound on the objective value of the initial instance, i.e. $PP(\{i_1, \ldots, i_2\}, \{i_1, \ldots, i_1 + K_{i_1,i_2} - 1\}) \in \mathcal{P}(\mathcal{M}, \mathcal{J})$. ■

Our second lifting procedure is based on the following result by Haouari and Jemmali (2008a) which proved to be effective in lifting $P||C_{min}$-upper bounds: In any feasible $P||C_{min}$-schedule there exists at least a set of i machines ($i = 1, \ldots, m$) on which in total at most

$$\mu_i(n, m) = i \lfloor n/m \rfloor + \max\{0, n - m(\lfloor n/m \rfloor + 1) + i\} \tag{6.13}$$

jobs are processed. Equation (6.13) is readily obtained when a "cardinality-balanced" schedule is considered in which the numbers of jobs assigned to the machines are as equal as possible, i.e. each machine processes either $\lfloor n/m \rfloor$ or $\lceil n/m \rceil$ jobs. However, as such a schedule might not be realizable when cardinality limits have to be taken into account, there is some potential to tighten Equation (6.13). In that regard, observe that when the first $i-1$ machines process their maximal number of $K_{1,i-1}$ jobs and the upper cardinality limit k_i of the next machine is smaller than or equal to the average number of jobs $\lfloor \frac{n-K_{1,i-1}}{m-(i-1)} \rfloor$ on the remaining machines $i, \ldots, m$, then the first i machines can process at most $K_{1,i}$ jobs instead of $\mu_i(n, m)$. It is not difficult to see that $K_{1,i}$ is smaller than (or equal

to) $\mu_i(n, m)$. We let ρ denote the maximum machine index so that the aforementioned inequality is fulfilled, i.e.

$$\rho = \underset{i=1,\dots,m}{\arg\max} \left\{ k_i \leq \left\lfloor \frac{n - K_{1,i-1}}{m - (i-1)} \right\rfloor \right\}. \tag{6.14}$$

Note that ρ is well-defined: For any instance of the k_i-PP we have $\rho \geq 1$ because $k_1 \leq \lfloor n/m \rfloor$. Then, given $k = (k_1, \dots, k_m)$, in any feasible schedule there exists at least a set of i machines ($1 \leq i \leq m$) on which in total at most

$$\bar{\mu}_i(n, m, k) = \begin{cases} K_{1,i} & \text{, if } i \leq \rho \\ K_{1,\rho} + \mu_{i-\rho}(n - K_{1,\rho}, m - \rho) & \text{, otherwise} \end{cases} \tag{6.15}$$

jobs are processed. This is a tighter version of Equation (6.13), i.e. $\bar{\mu}_i(n, m, k) \leq \mu_i(n, m)$ for all n, m, k, and i. Considering the first i machines and the longest $\bar{\mu}_i(n, m, k)$ jobs directly yields the following corollary.

Corollary 6.3.3
For all $i = 1, \dots, m$, the partial instance $PP(\{1, \dots, i\}, \{1, \dots, \bar{\mu}_i(n, m, k)\})$ is an element of $\mathcal{P}(\mathcal{M}, \mathcal{J})$.

Note that for $i \leq q$ the partial instances considered in Corollary 6.3.3 are identical with the ones in Corollary 6.3.2 for $i_1 = 1$ and $i_2 = i$.

In sum, we have identified $\mathcal{O}(m^2)$ partial instances (cf. Corollary 6.3.2 and 6.3.3) whose optimal objective values represent upper bounds on the optimal objective value of the given instance. However, as each partial instance itself represents an instance of the k_i-partitioning problem we can combine the two lifting procedures, meaning Corollary 6.3.2 can also be applied to each partial instance obtained from Corollary 6.3.3 and vice versa. This way, $\mathcal{O}(m^3)$ partial instances are received. Preliminary tests revealed that application of Corollary 6.3.3 to each partial instance obtained from Corollary 6.3.2 performs slightly better than the other way around.
Before we proceed to the development of upper bound procedures we want to emphasize once again that it is not necessary to optimally solve the identified partial instances. Instead, application of upper bound procedures to the partial instances is sufficient to potentially tighten the upper bound on C^*_{min} of the initial instance.

6.3.2 Upper bound procedures

We also derive several upper bound arguments. At first note that any upper bound for $P||C_{min}$ (cf., e.g., Haouari and Jemmali, 2008a, and Walter et al., 2016) is also valid for our k_i-partitioning problem. However, as these bounds disregard the cardinality constraints, we will not only consider bounds adapted from $P||C_{min}$ but mainly introduce new upper bounds that explicitly take into account the additional constraints on the minimum as well as maximum number of jobs that can be assigned to each machine. In what follows, we present all of our upper bounds in their most general form, i.e. their computation does not require the application of our preprocessing procedures (see Section 6.2) in advance.

We begin with two simple bounds. The first one is an immediate consequence of Criterion 6.2.2 (see Section 6.2.2). Recalling that a machine whose upper cardinality limit equals 1 should process the overall longest available job,

$$\mathrm{UB}_0 = t_{h'} \tag{6.16}$$

where $h' = \arg\max_{i=1,\ldots,m}\{k_i = 1\}$ constitutes a trivial upper bound. Note also that UB_0 is the optimal objective if there exists a feasible, but not necessarily optimal, assignment of the remaining jobs to the remaining machines so that each of these machines runs at least as long as UB_0.

Solving the continuous relaxation of $P||C_{min}$ (i.e. (6.1)–(6.3) and (6.5) replaced by $0 \leq x_{ij} \leq 1$ for $i = 1, \ldots, m$ and $j = 1, \ldots, n$) yields the second straightforward upper bound:

$$\mathrm{UB}_1 = \left\lfloor \frac{\sum_{j=1}^{n} t_j}{m} \right\rfloor. \tag{6.17}$$

Since UB_1 is easy to compute, application of our two lifting procedures (cf. Section 6.3.1) is to be recommended. If both procedures are combined as described at the end of Subsection 6.3.1, then the resulting lifted bound is

$$\widetilde{\mathrm{UB}}_1 = \min_{1 \leq i_1 \leq i_2 \leq m} \left\{ \min_{i=1,\ldots,i_2-i_1+1} \left\{ \left\lfloor \frac{\sum_{j=i_1}^{i_1+\bar{\mu}_i(K_{i_1,i_2},i_2-i_1+1,k)-1} t_j}{i} \right\rfloor \right\} \right\}. \tag{6.18}$$

Note that $\widetilde{\mathrm{UB}}_1 \leq \mathrm{UB}_1$ and, if Criterion 6.2.2 is not applied in advance, we also have that $\widetilde{\mathrm{UB}}_1 \leq \mathrm{UB}_0$.

At this point, we shall remark that one can also solve the continuous relaxation of the k_i-partitioning problem instead of its unconstrained version $P||C_{min}$. However, in order to obtain a (lifted) upper bound from the solution of the continuous relaxation it turned out to be sufficient to consider the unconstrained problem which is not only easier to solve but usually also yields the same bound.

We continue with the development of a more complex upper bound. Let $M \subset \mathcal{M}$ and $J \subset \mathcal{J}$ denote a subset of the machines and jobs, respectively, the next bound bases on the following observation. If the mean completion time of the partial problem $PP(M, J)$ is less than or equal to a valid lower bound LB (cf. Section 6.4) on the optimal objective value C^*_{min}, then the mean completion time of the residual problem $PP(\mathcal{M} \setminus M, \mathcal{J} \setminus J)$ is greater than or equal to C^*_{min} and, thus, provides an upper bound UB, i.e.

$$\frac{\sum_{j \in J} t_j}{|M|} \leq \mathrm{LB} \leq C^*_{min} \Rightarrow C^*_{min} \leq \frac{\sum_{j \in \mathcal{J} \setminus J} t_j}{m - |M|} = \mathrm{UB}. \tag{6.19}$$

To obtain a tight bound, the determination of M and J is crucial. For this purpose we

suggest to solve the following variant of a subset sum problem for each $i \in \{1, \ldots, m-1\}$:

$$\text{Minimize } Z_i = \sum_{j=1}^{n} t_j \cdot x_j^i \tag{6.20}$$

$$\text{s.t. } \sum_{j=1}^{n} t_j \cdot x_j^i \geq i \cdot \text{LB} \tag{6.21}$$

$$\sum_{j=1}^{n} x_j^i \leq \bar{\mu}_i(n, m, k) \tag{6.22}$$

$$x_j^i \in \{0, 1\} \qquad j = 1, \ldots, n. \tag{6.23}$$

Clearly, problem (6.20)–(6.23) is weakly $\mathcal{NP}$-hard. It determines a subset of the jobs whose sum of processing times is minimal (cf. (6.20)) subject to the constraints that the respective sum is not smaller than i times a given lower bound LB (cf. (6.21)) and the subset must not contain more than $\bar{\mu}_i(n, m, k)$ jobs (cf. (6.22)). Then,

$$\text{UB}_2 = \min_{i=1,\ldots,m-1} \left\{ \left\lfloor \frac{\sum_{j=1}^{n} t_j - Z_i^*}{m-i} \right\rfloor \right\} \tag{6.24}$$

constitutes an upper bound on C_{min}^* because of the following facts: As LB is a valid lower bound on C_{min}^*, in an optimal schedule each machine runs at least as long as LB. In particular, the cumulative completion time of the i machines that process in total at most $\bar{\mu}_i(n, m, k)$ jobs is at least as large as $i \cdot LB$. Recall from Corollary 6.3.3 that there always exists such a subset of the machines. The optimal objective value Z_i^* of problem (6.20)–(6.23) gives the smallest realizable cumulative completion time of the i machines. Hence, the cumulative completion time of the remaining $m-i$ machines is at most $\sum_{j=1}^{n} t_j - Z_i^*$, i.e. the average completion of these machines is at most $\lfloor (\sum_{j=1}^{n} t_j - Z_i^*)/(m-i) \rfloor$ so that $\text{UB}_2 \geq C_{min}^*$. Clearly, the better the LB the better UB_2. We refer to Section 6.4 for the computation of lower bounds.
Since solving each of the $m-1$ problems (6.20)–(6.23) requires pseudo-polynomial time, we abstained from applying our combined lifting approach to UB_2. In our experiments (see Section 6.5) we used Gurobi 6.0.3 to solve (6.20)–(6.23).

Our next upper bound is a generalization of UB_0. Let $r_1 \geq 1$ denote the index of the last machine whose upper cardinality limit equals k_1, i.e. $r_1 = \arg\max_{h=1,\ldots,m} \{k_h = k_1\}$. Then,

$$\text{UB}_3 = \begin{cases} \sum_{j=1}^{k_1-1} t_j + t_{K_{1,r_1}} & \text{, if } K_{1,r_1} = r_1 \cdot k_1, \\ \sum_{j=1}^{k_1-1} t_j & \text{, if } K_{1,r_1} < r_1 \cdot k_1 \end{cases} \tag{6.25}$$

constitutes an upper bound. The correctness of UB_3 is readily verified. At first recall that $K_{1,r_1} \leq r_1 \cdot k_1$ (cf. (6.6)) and $PP(\{1, \ldots, r_1\}, \{1, \ldots, K_{1,r_1}\}) \in \mathcal{P}(\mathcal{M}, \mathcal{J})$ (cf. Corollary 6.3.2). In case $K_{1,r_1} = r_1 \cdot k_1$, the shortest of these jobs, i.e. job K_{1,r_1}, must be assigned to one of the first r_1 machines. Thus, its assignment to a machine together with the $k_1 - 1$ longest jobs yields a valid upper bound. In the other case, i.e. $K_{1,r_1} < r_1 \cdot k_1$, there exists at least one machine which processes less than k_1 jobs.

Since UB_3 is easy to compute we suggest to lift this bound according to Corollary 6.3.2. Let $r_i = \arg\max_{h=i,\ldots,m} \{k_h = k_i\}$ for $(i = 1, \ldots, m)$, then by considering the partial

instances $PP(\{i,\ldots,r_i\},\{i,\ldots,i+K_{i,r_i}-1\})$ for $(i=1,\ldots,m)$ we arrive at the lifted upper bound $\widetilde{\mathrm{UB}}_3 = \min_{i=1,\ldots,m}\{\mathrm{UB}_3(i)\}$ where

$$\mathrm{UB}_3(i) = \begin{cases} \sum_{j=i}^{i+k_i-2} t_j + t_{i+K_{i,r_i}-1} & \text{, if } K_{i,r_i} = (r_i - i + 1)\cdot k_i, \\ \sum_{j=i}^{i+k_i-2} t_j & \text{, if } K_{i,r_i} < (r_i - i + 1)\cdot k_i. \end{cases} \tag{6.26}$$

It is readily verified that no other partial instances resulting from application of our lifting procedures (cf. Corollary 6.3.2 and 6.3.3) are able to further improve $\widetilde{\mathrm{UB}}_3$.

Our last upper bound exploits the fact that the well-known LPT algorithm is optimal when $k_i \leq 2$ for all $i = 1,\ldots,m$ (cf. also Dell'Amico and Martello, 1995). So, let $\underline{r}$ and $\bar{r}$ denote the index of the first and last machine whose upper cardinality limit equals 2, i.e. $\underline{r} = \min\{i \in \{1,\ldots,m\} : k_i = 2\}$ and $\bar{r} = \max\{i \in \{\underline{r},\ldots,m\} : k_i = 2\}$, respectively. If $\underline{r}$ exists, then

$$\mathrm{UB}_4 = \begin{cases} \min_{j=\underline{r},\ldots,\bar{r}}\{t_j + t_{2\bar{r}-j+1}\} & \text{, if } K_{\underline{r},\bar{r}} = 2(\bar{r}-\underline{r}+1) \\ \min\left\{\min_{j=\underline{r},\ldots,2\bar{r}-K_{1,\bar{r}}}\{t_j\}, \min_{j=2\bar{r}-K_{1,\bar{r}}+1,\ldots,\bar{r}}\{t_j + t_{2\bar{r}-j+1}\}\right\} & \text{, if } K_{\underline{r},\bar{r}} < 2(\bar{r}-\underline{r}+1) \end{cases} \tag{6.27}$$

represents an upper bound which is derived from the (partial) instance $PP(\{\underline{r},\ldots,\bar{r}\},\{\underline{r},\ldots,K_{1,\bar{r}}\})$. It is not difficult to see that, this time, the consideration of other partial instances resulting from Section 6.3.1 will not yield a tighter version of UB_4.

In case that the best upper bound (denoted by U^*) is given by $\widetilde{\mathrm{UB}}_1$ or UB_2 there is potential for further improvement. Both $\widetilde{\mathrm{UB}}_1$ and UB_2 are grounded on averaged machine completion times, but this does not necessarily mean that a machine can indeed finish exactly at that time. Therefore, we apply the following enhancement procedure (due to Haouari and Jemmali, 2008a) which computes the largest sum of processing times that is less than or equal to U^* by solving the subset problem (6.28)–(6.30):

$$\text{Maximize } \mathrm{UB}_5 = \sum_{j=1}^{n} t_j \cdot x_j \tag{6.28}$$

$$\text{s.t. } \sum_{j=1}^{n} t_j \cdot x_j \leq \mathrm{U}^* \tag{6.29}$$

$$x_j \in \{0,1\} \qquad j = 1,\ldots,n. \tag{6.30}$$

Clearly, we have $C^*_{min} \leq \mathrm{UB}_5 \leq \mathrm{U}^*$.

We finish this section with an example to illustrate the application of the adjustment procedures (6.7)–(6.8) and the lifting procedures (cf. Corollary 6.3.2 and 6.3.3). The effect of combining our two lifting procedures will be clarified as well.

Example 6.3.4

Given $n = 10$ jobs with processing times $p = (201, 102, 99, 86, 82, 79, 74, 73, 65, 64)$ and $m = 5$ machines with upper cardinality limits $k = (3,5,5,5,5)$ and implicit lower limits $l = (1,1,1,1,1)$. At first, we apply (6.7) to tighten the upper limits:

$$k_1 = \min\left\{\left\lfloor\frac{K_{1,5}}{5}\right\rfloor, \left\lfloor\frac{K_{1,4}}{4}\right\rfloor, \left\lfloor\frac{K_{1,3}}{3}\right\rfloor, \left\lfloor\frac{K_{1,2}}{2}\right\rfloor, \left\lfloor\frac{K_{1,1}}{1}\right\rfloor\right\} = \min\left\{\left\lfloor\frac{10}{5}\right\rfloor, \ldots, \left\lfloor\frac{3}{1}\right\rfloor\right\} = 2,$$

$$k_2 = \min\left\{\left\lfloor\frac{9}{4}\right\rfloor, \left\lfloor\frac{8}{3}\right\rfloor, \left\lfloor\frac{7}{2}\right\rfloor, \left\lfloor\frac{5}{1}\right\rfloor\right\} = 2,\ k_3 = 2,\ k_4 = 3,\ k_5 = 5.$$

The enhanced upper limits $k = (2, 2, 2, 3, 5)$ *are now used to improve the lower limits according to* (6.8)*:*

$$l_1 = \left\lceil \frac{L_{1,1}}{1} \right\rceil = \left\lceil \frac{1}{1} \right\rceil = 1,\ l_2 = \max\left\{ \left\lceil \frac{L_{1,2}}{2} \right\rceil, \left\lceil \frac{L_{2,2}}{1} \right\rceil \right\} = \max\left\{ \left\lceil \frac{2}{2} \right\rceil, \left\lceil \frac{1}{1} \right\rceil \right\} = 1,$$
$$l_3 = \max\left\{ \left\lceil \frac{3}{3} \right\rceil, \left\lceil \frac{2}{2} \right\rceil, \left\lceil \frac{1}{1} \right\rceil \right\} = 1,\ l_4 = \max\left\{ \left\lceil \frac{5}{4} \right\rceil, \left\lceil \frac{3}{3} \right\rceil, \left\lceil \frac{2}{2} \right\rceil, \left\lceil \frac{1}{1} \right\rceil \right\} = 2,\ l_5 = 2.$$

It is readily verified that a repeated application of (6.7) *using the enhanced lower limits* $l = (1, 1, 1, 2, 2)$ *cannot further reduce the upper limits.*

Computation of the upper bounds UB_1 *and* UB_3 *yields*

$$UB_1 = \left\lfloor \frac{925}{5} \right\rfloor = 185 \quad \textit{and} \quad UB_3 = t_1 + t_6 = 201 + 79 = 280.$$

Next, we reveal the effect of our lifting procedures. With regards to UB_1 *we first apply Corollary 6.3.2 and 6.3.3 individually. The respective lifted bounds are denoted by* $\widetilde{UB}_1^1 = \min_{1 \le i_1 \le i_2 \le 5}\{UB_1^1(i_1, i_2)\}$ *and* $\widetilde{UB}_1^2 = \min_{i=1,\ldots,5}\{UB_1^2(i)\}$ *where*

$$UB_1^1(i_1, i_2) = \left\lfloor \frac{\sum_{j=i_1}^{i_1+K_{i_1,i_2}-1} t_j}{i_2 - i_1 + 1} \right\rfloor,\ UB_1^2(i) = \left\lfloor \frac{\sum_{j=1}^{\bar{\mu}_i(n,m,k)} t_j}{i} \right\rfloor.$$

Afterward, we briefly present the result of the combined lifting approach (cf. last paragraph of Section 6.3.1). Table 6.1 provides detailed information on the bounds $\widetilde{UB}_1^1$, $\widetilde{UB}_1^2$, *and* $\widetilde{UB}_3$. *Regarding* $\widetilde{UB}_1^2$, *we have* $\rho = 3$ *since* $k_4 > \lfloor 4/2 \rfloor$ *(cf.* (6.14)*). As can be seen,*

i_1	i_2	K_{i_1,i_2}	UB_1^1	i_1	i_2	K_{i_1,i_2}	UB_1^1	i_1	i_2	K_{i_1,i_2}	UB_1^1	i	$\bar{\mu}_i$	UB_1^2	i	r_i	UB_3
1	1	2	303	2	2	2	201	3	4	5	210	1	2	303	1	3	280
	2	4	244		3	4	184		5	8	207	2	4	244	2	3	184
	3	6	216		4	7	198	4	4	3	247	3	6	216	3	3	185
	4	8	199		5	9	181		5	7	261	4	8	199	4	4	247
	5	10	185	3	3	2	185	5	5	5	373	5	10	185	5	5	373

Table 6.1: Upper bounds $\mathrm{UB}_1^1(i_1, i_2)$, $\mathrm{UB}_1^2(i)$, and $\mathrm{UB}_3(i)$

we obtain $\widetilde{UB}_1^1 = 181$, $\widetilde{UB}_1^2 = 185$, *and* $\widetilde{UB}_3 = 184$. *Up to this point,* $\widetilde{UB}_1^1$ *is the best upper bound. However, when we combine our two lifting procedures as done in* (6.18) *we finally receive* $\widetilde{UB}_1 = 174$ *which is due to the application of Corollary 6.3.3 to the partial instance* $PP(\{2, \ldots, 5\}, \{2, \ldots, 10\})$. *Regarding this partial instance, it is readily verified that* $\widetilde{UB}_1^2 = \min\{201, 184, 174, 181\} = 174$.

6.4 Algorithms

This section is concerned with the development of lower bounds for the k_i-partitioning problem. Specifically, we introduce two LPT-based construction algorithms (Section 6.4.1) and a dynamic programming based improvement heuristic (Section 6.4.2).

6.4.1 Construction heuristics

To construct initial solutions, we propose two modified LPT algorithms that adequately take the cardinality constraints into account. The two algorithms differ in the way they incorporate the upper and lower limits. While our first variant (dubbed LPT1) primarily concentrates on the upper limits, the second variant (dubbed LPT2) focuses on the lower limits first and observes the upper limits subsequently. Let n_i denote the current number of jobs assigned to machine i, a brief description of our two variants is given below.

LPT1: As long as the number of unassigned jobs, i.e. $n - \sum_{i=1}^{m} n_i$, is greater than the total number of jobs that are still required to satisfy all lower limits, i.e. $\sum_{i=1}^{m} \max\{l_i - n_i, 0\}$, LPT1 successively assigns the longest remaining job to the machine with the current shortest completion time among all i where $n_i < k_i$. Ties are broken in favor of the machine that has the largest difference $l_i - n_i$. Once $n - \sum_{i=1}^{m} n_i = \sum_{i=1}^{m} \max\{l_i - n_i, 0\}$, LPT1 successively assigns the longest remaining job to the machine with the current shortest completion time among all i where $n_i < l_i$ until no job remains unassigned. Now, ties are broken in favor of the machine that has the smallest difference $l_i - n_i$.

LPT2: As long as not all lower limits are satisfied, LPT2 successively assigns the longest remaining job to the machine with the current shortest completion time among all i where $n_i < l_i$. This time, ties are broken in favor of the machine that has the smallest lower limit l_i. Once $n_i = l_i$ for all i, LPT2 successively assigns the longest remaining job to the machine with the current shortest completion time among all i where $n_i < k_i$ until no job remains unassigned. Now, ties are broken in favor of the machine that has the smallest difference $k_i - n_i$.

6.4.2 A subset sum based improvement heuristic

To improve on the quality of a given solution we propose an iterative approach whose underlying idea is related to the multi-start local search method used in Haouari and Jemmali (2008a) which has proved to be effective for the unconstrained problem version $P||C_{min}$. Assuming the machines to be sorted according to non-decreasing completion times, i.e. $C_1 \leq \ldots \leq C_m$, their method iteratively selects pairs of machines $(1, h)$ for $h = m, \ldots, 2$ and solves the resulting $P2||C_{min}$ instance to optimality. Each $P2||C_{min}$ instance is reformulated as a subset sum problem as follows:

$$\text{Maximize} \quad \sum_{j \in J} t_j \cdot z_j \tag{6.31}$$

$$\text{s.t.} \quad \sum_{j \in J} t_j \cdot z_j \leq \left\lfloor \sum_{j \in J} t_j / 2 \right\rfloor \tag{6.32}$$

$$z_j \in \{0, 1\} \qquad \forall j \in J \tag{6.33}$$

where $J = J_1 \cup J_h$ is the union of jobs that are assigned either to machine 1 or h in the current solution and z_j ($j \in J$) are binary variables that take the value 1 if j is assigned to machine 1 and 0 otherwise. Once a better value for C_1 is identified, the machines are resorted and the next iteration begins. If C_1 could not be increased after considering all $m-1$ pairs, then the procedure stops. For further details we refer to Haouari and Jemmali (2008a).

In the subset sum formulation (6.31)–(6.33) for the unconstrained version, one can assume without loss of generality that machine 1 is the one whose completion is not greater than the one of machine h. However, the situation is different when cardinality constraints have to be taken into account. Here, it is not sufficient to simply add the constraint

$$\max\{l_1, |J| - k_h\} \leq \sum_{j \in J} z_j \leq \min\{k_1, |J| - l_h\} \tag{6.34}$$

to (6.31)–(6.33) and to solve the model because we can no longer assume the completion time of machine 1 to be smaller than or equal to machine's h completion time (cf. Example 6.4.1). Therefore, it is necessary to solve a second model as well where the constraint

$$\max\{l_h, |J| - k_1\} \leq \sum_{j \in J} z_j \leq \min\{k_h, |J| - l_1\} \tag{6.35}$$

is added to (6.31)–(6.33) instead of constraint (6.34) so that this second model, now, forces the completion time of machine h to be not greater than the one of machine 1. It the second model it is important to note that z_j takes the value 1 if j is assigned to machine h and 0 otherwise.

Example 6.4.1
Consider $n = 5$ jobs with processing times $p = (13, 11, 7, 5, 3)$ and $m = 2$ machines with upper and lower cardinality limits $k = l = (2, 3)$.

i	k_i	J_i	C_i
1	2	{2,3}	18
2	3	{1,4,5}	21

Table 6.2: Solution w.r.t. (6.31)–(6.33) + (6.34)

i	k_i	J_i	C_i
1	2	{1,3}	20
2	3	{2,4,5}	19

Table 6.3: Solution w.r.t (6.31)–(6.33) + (6.35)

6.4.2.1 Procedure k_i-DP for solving the case $m = 2$

To avoid solving two subset sum problems for each pair of machines, we propose the following model (6.36)–(6.40) which optimally solves k_i-partitioning problems on two parallel machines by minimizing the absolute difference Δ between the completion time $\sum_{j \in J} t_j \cdot z_j$ of machine 1 and the average completion time $\sum_{j \in J} t_j/2$ of the two machines 1 and h (cf. (6.36)–(6.38)) while observing the machine's cardinality limits (cf. (6.39)).

$$\text{Minimize } \Delta \tag{6.36}$$

$$\text{s.t. } \sum_{j \in J} t_j \cdot z_j \leq \sum_{j \in J} t_j/2 + \Delta \tag{6.37}$$

$$\sum_{j \in J} t_j \cdot z_j \geq \sum_{j \in J} t_j/2 - \Delta \tag{6.38}$$

$$\max\{l_1, |J| - k_h\} \leq \sum_{j \in J} z_j \leq \min\{k_1, |J| - l_h\} \tag{6.39}$$

$$z_j \in \{0, 1\} \qquad \forall j \in J \tag{6.40}$$

Clearly, problem (6.36)–(6.40) is $\mathcal{NP}$-hard in the ordinary sense as well. Hence, we developed a generic dynamic programming procedure (dubbed k_i-DP) to exactly solve k_i-partitioning problems on two machines. The input of the procedure is summarized in Table 6.4. Furthermore, we define a 2-dimensional binary array S of size $(n+1) \times (C+1)$ which

n	Positive integer representing the number of jobs in the two machine problem
p	Array of size n where $p[j]$ is a positive integer representing the j-th shortest processing time (i.e., the times are sorted in non-decreasing order)
C	Positive integer representing the maximum allowable completion time of the first machine
k	Positive integer representing the maximum allowable number of jobs on the first machine (cf. also right-hand side of (6.39))
l	Positive integer representing the minimum allowable number of jobs on the first machine (cf. also left-hand side of (6.39))

Table 6.4: Input of k_i-DP

stores the recursively determined information on the first machine's realizable completion times – depending on how many of the shortest jobs are considered – without exceeding the maximum allowable number of jobs on that machine (cf. Figure 6.3). More precisely, $S[j,c] = 1$ $(0 \leq j \leq n, 0 \leq c \leq C)$ if there exists a subset of the first j jobs (i.e. the shortest ones) so that the subset sum – which represents the completion time of the first machine – equals c and the subset contains at most k elements; otherwise $S[j,c] = 0$. Moreover, associated with each array element $[j,c]$ of S are (i) a binary array $S'_{j,c}$ of length $k+1$ where $S'_{j,c}[k'] = 1$ $(0 \leq k' \leq k)$ if there exists a subset of the first j jobs so that the subset sum equals c and the subset contains exactly k' elements; otherwise $S'_{j,c}[k'] = 0$, and (ii) an array $P_{j,c}$ of length $k+1$ where $P_{j,c}[k']$ $(0 \leq k' \leq k)$ stores the preceding c-value in case that $S'_{j,c}[k'] = 1$; otherwise $P_{j,c}[k'] = -1$. The arrays $S'_{j,c}$ and $P_{j,c}$ are required to observe the cardinality limits and for backtracking, respectively.
The initialization of the arrays is as follows:

$$
\begin{aligned}
&S[0,0] = 1; \quad S[0,c] = 0 \;\; (c = 1, \ldots, C) \\
&S'_{0,0}[0] = 1; \quad S'_{0,0}[k'] = 0 \;\; (k' = 1, \ldots, k); \quad S'_{0,c}[k'] = 0 \;\; (k' = 0, \ldots, k) \\
&P_{0,c}[k'] = -1 \;\; (c = 0, \ldots, C;\; k' = 0, \ldots, k).
\end{aligned}
$$

Figure 6.3 provides the pseudo code of k_i-DP and the recursion formulas to fill S as well as $S'_{j,c}$ and $P_{j,c}$ for each element $[j,c]$ of S. As can be seen, the respective arrays are filled within three nested loops using conditional statements. The outer loop iterates over the elements $p[j]$ of the array of processing times, the intermediate loop iterates over the allowable completion times c, and the inner loop iterates for each pair (j,c) over the allowable number of jobs k'. In case $c < p[j]$ (see 03.–07.), $S[j,c] = S[j-1,c]$ and all entries in the associated $S'_{j,c}$- and $P'_{j,c}$-arrays are copied from the previous row as well. In the other case, i.e. $c \geq p[j]$, it is checked which c-values can be realized – without exceeding the maximum allowable number of addends (or jobs) k – when the integer (or processing time) $p[j]$ becomes available in addition to $p[1], \ldots, p[j-1]$ (08.–26.). If c can be realized using $p[j]$ (09.–19.), $S[j,c]$ is set to one (11.) and the entries in the associated arrays $S'_{j,c}$

```
01. for j = 1 : n
02.     for c = 0 : C
03.         if c < p[j]
04.             S[j, c] = S[j − 1, c]
05.             for k' = 0 : k
06.                 S'_{j,c}[k'] = S'_{j−1,c}[k']; P_{j,c}[k'] = P_{j−1,c}[k']
07.             end
08.         else
09.             if S[j − 1, c − p[j]] == 1 and S'_{j−1,c−p[j]}[k'] == 1 (for
10.             at least one k' ∈ {0, ..., k − 1})
11.                 S[j, c] = 1
12.                 S'_{j,c}[0] = S'_{j−1,c}[0]; P_{j,c}[0] = P_{j−1,c}[0]
13.                 for k' = 1 : k
14.                     if S'_{j−1,c−p[j]}[k' − 1] == 1 and S'_{j−1,c}[k'] == 0
15.                         S'_{j,c}[k'] = 1; P_{j,c}[k'] = c − p[j]
16.                     else
17.                         S'_{j,c}[k'] = S'_{j−1,c}[k']; P_{j,c}[k'] = P_{j−1,c}[k']
18.                     end
19.                 end
20.             else
21.                 S[j, c] = S[j − 1, c]
22.                 for k' = 0 : k
23.                     S'_{j,c}[k'] = S'_{j−1,c}[k']; P_{j,c}[k'] = P_{j−1,c}[k']
24.                 end
25.             end
26.         end
27.     end
28. end
```

Figure 6.3: Dynamic programming procedure k_i-DP

and $P'_{j,c}$ are determined by checking (13.–19.) how many jobs are required to realize c when $p[j]$ is used. Only if c is realized for the first time using exactly k' ($k' = 1, \ldots, k$) out of the shortest j jobs, the respective entry in $S'_{j,c}[k']$ is set to one and $P'_{j,c}[k'] = c - p[j]$ (14.–15.). For all other k', the entries in $S'_{j,c}$ and $P'_{j,c}$ are copied from the previous row (16.–18.). The same is done if c cannot be realized using $p[j]$ (see 20.–25.). After filling the last row of S and the associated arrays $S'_{n,c}$ and $P'_{n,c}$ for all $c = 0, \ldots, C$, the optimal objective value c^* is represented by that column which fulfills $S[n, c^*] = 1$, $S'_{n,c^*}[k'] = 1$ for at least one $k' \in \{l, l+1, \ldots, k\}$, and $\left| c^* - \sum_{j=1}^{n} p[j]/2 \right|$ is minimal. The corresponding solution is obtained by backtracking through the P'-arrays.

6.4.2.2 Procedure k_i-LS for solving the general case

We use the k_i-DP procedure within our iterative local search algorithm (dubbed k_i-LS) to generate high-quality solutions for the k_i-partitioning problem on an arbitrary (but fixed)

number m of machines. Given a feasible solution, each iteration of k_i-LS begins with relabeling the machines so that $C_1 \leq \ldots \leq C_m$. Then, we successively consider the machine pair $(1, h)$ for $h = m, \ldots, 2$ and solve the corresponding two machine problem (6.36)–(6.40) via k_i-DP (where $n = |J|$ $(J = J_1 \cup J_h)$, $k = \min\{k_1, |J| - l_h\}$, $l = \max\{l_1, |J| - k_h\}$, and $C = C_h$). If $C_1 < \sum_{j \in J} t_j \cdot z_j < C_h$, then we adopt the new assignment of the jobs in J, i.e. the current solution (to the m-machine problem) is modified by setting $J_1 := \{j \in J : z_j = 1\}$, $J_h := J \setminus J_1$, and $C_i := \sum_{j \in J_i} t_j$ for $i = 1, h$. Afterward, the next iteration starts. Otherwise, h is decremented by one. The procedure is stopped if no improvement has been achieved within an iteration, i.e. after sequentially considering all $m - 1$ machine pairs $(1, h)$ $(h = m, m - 1, \ldots, 2)$.

6.5 Computational study

This section elaborates on the details of our computational study where we examine the effectiveness of our preprocessing procedures, the tightness of our upper bound procedures, and the performance of the developed heuristic algorithms. As there is no established test bed available we, first, describe how our test instances have been generated (Section 6.5.1). Then, we specify in which order the developed methods for preprocessing, bounding, and solving a given instance are executed (Section 6.5.2). Finally, we computationally examine the performance of our solution approaches and report on the relevant results in Section 6.5.3.

6.5.1 Instance generation

In order to generate test instances for our k_i-partitioning problem, we adopted the generation scheme used in Dell'Amico et al. (2006) who studied the "dual" problem version, i.e. k_i-partitioning with minimum makespan objective. Assuming $n/m \geq 2$, we consider 26 pairs of $n \in \{10, 25, 50, 100, 200\}$ and $m \in \{3, 4, 5, 10, 20, 40, 50\}$. For each of these pairs we investigate 81 different combinations of 9 processing time classes (dubbed T_j) and 9 cardinality classes (dubbed K_i). For each quadruple (n, m, T_j, K_i) 10 independent instances have been randomly generated resulting in a total number of 21060 $(= 26 \times 81 \times 10)$ instances.

Regarding the classes T_j, the processing times are independently drawn from a discrete uniform distribution on $\{t_{\min}, \ldots, t_{\max}\}$ for classes T_1–T_3, an exponential distribution with parameter λ for classes T_4–T_6 (disregarding non-positive values), and a normal distribution with mean μ and standard deviation σ for classes T_7–T_9 (disregarding non-positive values), respectively. Table 6.5 lists the corresponding parameter values. Turning to the classes K_i, the upper cardinality limits are independently drawn from a uniform distribution on $\{k_{\min}, \ldots, k_{\max}\}$ for classes K_1–K_6 while they are generated according to Figure 6.4 for classes K_7–K_9. The respective parameter values are given in Table 6.6. Instances where $\sum_{i=1}^{m} k_i < n$ have been discarded and replaced by new ones. For further information on the cardinality classes we refer to Dell'Amico et al. (2006).

	$t_{\min}$	$t_{\max}$		λ		μ	σ
T_1	10	1,000	T_4	1/25	T_7	100	33
T_2	200	1,000	T_5	1/50	T_8	100	66
T_3	500	1,000	T_6	1/100	T_9	100	100

Table 6.5: Parameters used for the processing time classes (cf. Dell'Amico and Martello, 2001, Dell'Amico et al., 2006)

	$k_{\min}$	$k_{\max}$		$k_{\min}$	$k_{\max}$		δ
K_1	$\lceil n/m \rceil - 1$	$\lceil n/m \rceil$	K_4	$\lceil n/m \rceil$	$\lceil n/m \rceil + 1$	K_7	1
K_2	$\lceil n/m \rceil - 1$	$\lceil n/m \rceil + 1$	K_5	$\lceil n/m \rceil$	$\lceil n/m \rceil + 2$	K_8	3/2
K_3	$\lceil n/m \rceil - 2$	$\lceil n/m \rceil + 2$	K_6	$\lceil n/m \rceil$	$\lceil n/m \rceil + 3$	K_9	2

Table 6.6: Parameters used for the cardinality classes (cf. Dell'Amico et al., 2006)

6.5.2 Execution scheme

The order in which we executed the developed preprocessing, bounding, and solution methods is provided by Figure 6.5. As can be seen, a great deal of effort is put into the preprocessing and bounding step (see Phase 1). With the intention to reduce the problem's size and to obtain strong upper bounds as well as a good initial solution, we iteratively apply the methods from Section 6.2, 6.3, and 6.4.1. Our improvement procedure k_i-LS is only applied if the reduced problem contains more than one machine and if there is still a gap between the current best upper and lower bound value U^* and L^* (see Phase 2). After application of k_i-LS, we compute the upper bound UB_2 (see 26.). The respective subset sum problems (cf. (6.20)–(6.23)) are solved with the help of Gurobi (version 6.0.3). Finally, we calculate UB_5 in case that the current best upper bound is neither given by $\widetilde{UB}_3$ nor UB_4 (27.–29.; cf. also paragraph on UB_5 within Section 6.3.2).

All of our methods have been implemented in C++ using the Visual C++ 2010 compiler and the tests have been carried out on a personal computer with an Intel Core i7-2600 processor, 8GB RAM, and Windows 7 Professional SP1 (64 bit).

6.5.3 Experimental results

This section reports on the results of our computational tests. Table 6.7 lists the 12 relevant performance criteria. The first group of criteria (#Ph1, #Red, $\%m_{elim}$, $\%n_{elim}$) focuses on the performance of the preprocessing and reduction procedures (see Tables 6.8 and 6.9). The second group (%GAP, MAX, #OPT, TIME) allows for a general assessment of the overall performance of our bounding and solution methods (see Tables 6.10 and 6.11) while the third group (%BESTi, %OPTi, #ImpLift, $\#<\widetilde{UB}_1$) is meant to provide a clear picture on the effectiveness of our bounding and lifting procedures (see Tables 6.12–6.15).

We start with evaluating the performance of Phase 1 (cf. Figure 6.5, steps 01.–22.). The values of the respective performance criteria (cf. first group in Table 6.7) are provided by Table 6.8 (broken down by the processing time classes) and Table 6.9 (broken down by the cardinality classes). The column #Ph1 gives the number of instances that have already

1. $Sumk = \lfloor \delta n \rfloor - 2m$
2. **for** $i = 1 : m - 1$
3. $\quad r$ = random number from uniform distribution on $\{0, \ldots, \lfloor Sumk/2 \rfloor\}$
4. $\quad k_i = 2 + r$
5. $\quad Sumk = Sumk - r$
6. **end**
7. $k_m = 2 + Sumk$

Figure 6.4: Cardinality generation method for classes K_7–K_9 (cf. Dell'Amico et al., 2006)

Criteria	Description
#Ph1	Number of instances solved in Phase 1 (cf. Figure 6.5, steps 01.–22.)
#Red	Number of instances where the problem size could be reduced
$\%m_{elim}$	Average relative number of eliminated machines (in %)
$\%n_{elim}$	Average relative number of eliminated jobs (in %)
%GAP	Average relative gap $(U^* - L^*)/U^*$ between U^* and L^* (in %)
MAX	Maximum relative gap between U^* and L^* (in %)
#OPT	Number of optimally solved instances (i.e. U^*=L^*)
TIME	Average computation time required by k_i-LS (in seconds)
%BESTi	Relative number of instances where $UB_i = U^*$ (in %)
%OPTi	Relative number of instances where $UB_i = L^*$ (in %)
#ImpLift	Number of instances where $\widetilde{UB}_i < UB_i$
#$<\widetilde{UB}_1$	Number of instances where $UB_i < \widetilde{UB}_1$ (or $\widetilde{UB}_i < \widetilde{UB}_1$)

Table 6.7: Performance criteria

	#Ph1	#Red	$\%m_{elim}$	$\%n_{elim}$
T_1	138	137	1.63	1.27
T_2	212	114	1.59	1.54
T_3	232	130	2.36	2.26
T_4	208	125	1.93	1.86
T_5	182	251	2.30	2.03
T_6	177	376	3.55	3.16
T_7	1266	331	4.43	4.20
T_8	921	435	5.80	4.97
T_9	601	544	7.01	5.74
Avg/Tot	3937	2443	3.40	3.00

Table 6.8: Performance of the reduction procedures – processing time classes

	#Ph1	#Red	$\%m_{elim}$	$\%n_{elim}$
K_1	557	122	1.11	0.82
K_2	464	276	3.41	2.00
K_3	569	536	6.46	3.73
K_4	563	183	1.73	0.86
K_5	594	165	1.51	0.71
K_6	594	158	1.41	0.65
K_7	207	374	6.01	8.55
K_8	166	331	4.80	5.33
K_9	223	298	4.15	4.39
Avg/Tot	3937	2443	3.40	3.00

Table 6.9: Performance of the reduction procedures – cardinality classes

been solved within Phase 1 and, thus, allows for an overall assessment of the effectiveness of Phase 1. The column #Red displays the number of instances for which the size could be reduced successfully by application of our reduction criteria (i.e. Criterion 6.2.2 as well as reduction procedures 1&2). In conjunction with the two other criteria $\%m_{elim}$ and $\%n_{elim}$,

Phase 1: Preprocessing, bounding, and solution generation
01. Apply Criterion 6.2.2; set $U^* = \min\{\infty, UB_0\}$
02. Apply LPT1, LPT2; set $L^* = \max\{C_{min}(LPT1), C_{min}(LPT2)\}$
03. **if** $U^* > L^*$
04. **repeat**
05. **repeat**
06. Apply (6.7), (6.8), (6.11), and (6.12)
07. **until** all k_i and l_i remain unchanged
08. Apply Criterion 6.2.2
09. Determine UB_1, UB_3, UB_4; set $U^* = \min\{U^*, UB_1, UB_3, UB_4\}$
10. **if** $U^* > L^*$
11. Apply LPT1, LPT2; set $L^* = \max\{L^*, C_{min}(LPT1), C_{min}(LPT2)\}$
12. Determine $\widetilde{UB}_1$, $\widetilde{UB}_3$; set $U^* = \min\left\{U^*, \widetilde{UB}_1, \widetilde{UB}_3\right\}$
13. **if** $U^* > L^*$
14. Apply reduction procedures 1&2 (cf. Figures 6.1, 6.2)
15. **else**
16. *Return //optimal solution found*
17. **end**
18. **else**
19. *Return //optimal solution found*
20. **end**
21. **until** no more machines and jobs can be eliminated
22. **end**

Phase 2: Solution improvement and upper bound enhancement
23. **if** $m > 1$ **and** $U^* > L^*$
24. Apply k_i-LS to the better of the two solutions obtained from LPT1 and LPT2
25. Set $L^* = C_{min}(k_i\text{-LS})$
26. Determine UB_2; set $U^* = \min\{U^*, UB_2\}$
27. **if** $U^* < \min\{\widetilde{UB}_3, UB_4\}$
28. Determine UB_5; set $U^* = UB_5$
29. **end**
30. **end**

Figure 6.5: Execution scheme

#Red allows to judge the performance of the reduction criteria.

As can be seen from the #Ph1-column, we are able to optimally solve 3937 out of the 21060 instances (i.e. almost 19%) already within Phase 1. It is worth noting that for 416 out of the 3937 instances our algorithm has already terminated after step 02. (cf. Figure 6.5) because the inequality $UB_0 \geq L^*$ was fulfilled. Most of these 416 instances belong to the classes K_2 and K_3 and satisfy $k_i < \lceil n/m \rceil$ for all i.
Phase 1 turned out to be particularly effective when the processing times are drawn from one of the three normal distributions (i.e. T_7–T_9). Here, 2788 out of the 7020 corresponding instances (i.e. almost 40%) are optimally solved within Phase 1. Regarding the cardinality

classes, K_7–K_9 appear to be the more difficult ones for our reduction procedures as here only 596 (i.e. less than 9%) instances have been solved to optimality at Phase 1.

Taking a look at the other three columns, we can state that at least one of our reduction procedures is able to successfully decrease m and/or n for 2443 instances and the overall reduction of m and n is about 3.40% and 3.00% on average, respectively. We shall also remark that $m = 2$ holds for 217 of the 2443 reduced instances. Due to the nature of our k_i-DP, these instances have later been optimally solved in Phase 2. The largest entries in the columns $\%m_{elim}$ and $\%n_{elim}$ are to be found in Table 6.9 (see classes K_7–K_9). Although the number of instances solved at Phase 1 is rather small for K_7–K_9, the average numbers of eliminated machines and jobs are ranging between 4.15% and 8.55%. This striking behavior seems to be due to the fact that the cardinality limits are more diverse when they are generated according to K_7–K_9 (cf. Figure 6.4). So, on the one hand, the elimination of machines and/or jobs in advance is fostered but, on the other hand, it is more difficult to optimally solve such instances without application of more elaborate upper and lower bound procedures as done in Phase 2.

The overall performance of our algorithmic approach and, in particular, Phase 2 is evaluated next. The respective results for each of the 26 parameter settings (n, m) are summarized in the Tables 6.10 and 6.11 – again broken down by the processing time and cardinality classes, respectively. Analogous to the results reported in Dell'Amico et al. (2006), we noticed that our results are very similar for the related classes T_1–T_3, T_4–T_6, T_7–T_9, K_1–K_3, K_4–K_6, and K_7–K_9, respectively. Therefore, we abstain from providing the results for each individual processing time and cardinality class. Instead, we group the classes accordingly so that each entry in Table 6.10 and 6.11 corresponds to 270 $(= 3 \times 9 \times 10)$ instances.

Out of the 17123 instances that remained unsolved after Phase 1, our subset sum based heuristic k_i-LS improved the best LPT-solution 15921 times and UB_5 tightened the best upper bound 696 times so that we were able to solve another 10743 instances within Phase 2. More precisely, subtracting out the number of instances that have already been solved at Phase 1, we optimally solved 4077 out of the remaining 6438 instances of class T_1–T_3, 4158 out of 6453 instances of class T_4–T_6, and 2508 out of 4232 instances of class T_7–T_9. Regarding the cardinality classes, we optimally solved 3272 out of the remaining 5430 instances of class K_1–K_3, 2944 out of 5269 instances of class K_4–K_6, and 4527 out of 6424 instances of class K_7–K_9. So, the success of Phase 2 in solving an instance is more sensitive to the cardinality class than the processing times as here the share of solved instances ranges between 55.87% and 70.47% while it ranges only between 59.26% and 64.44% for the processing time classes.

Considering both phases together, we see from the last row in Table 6.10 that we were able to solve significantly more instances with normally distributed processing times (5296 out of 7020) than with exponentially and uniformly distributed times (4725 and 4659), respectively, and at the same time less computation time was required (0.03 s on average compared to 0.26 s and 2.53 s) by k_i-LS. On the downside, the overall relative gap between U^* and L^* averages 0.70% for the classes T_7–T_9 and is therefore greater than the respective average gap (0.61% and 0.39%) for the two other groups of processing time classes. Regarding the cardinality classes, the last row in Table 6.11 reveals that we identified more optimal solutions (5123 out of 7020 compared to 4862 and 4695) when the cardinality lim-

		T_1–T_3				T_4–T_6				T_7–T_9			
n	m	%GAP	MAX	#OPT	TIME	%GAP	MAX	#OPT	TIME	%GAP	MAX	#OPT	TIME
10	3	0.81	8.65	90	0.01	1.12	11.17	63	0.00	1.36	8.67	142	0.00
10	4	2.22	24.14	109	0.01	2.66	25.36	66	0.00	2.36	22.41	138	0.00
10	5	0.88	18.64	233	0.00	1.80	17.24	188	0.00	2.50	26.67	158	0.00
25	3	0.01	0.20	212	0.12	0.04	1.28	216	0.01	0.16	4.69	223	0.00
25	4	0.04	5.55	172	0.07	0.09	4.56	205	0.01	0.34	16.04	218	0.00
25	5	0.07	1.45	99	0.06	0.12	1.70	162	0.01	0.55	12.27	190	0.00
25	10	1.92	14.56	107	0.03	2.74	9.97	46	0.00	1.64	10.26	119	0.00
50	3	0.01	1.05	200	0.62	0.04	5.77	210	0.06	0.04	1.47	237	0.01
50	4	0.01	1.21	209	0.34	0.02	0.60	216	0.03	0.04	3.90	248	0.00
50	5	0.01	0.09	209	0.29	0.05	4.27	220	0.03	0.18	8.49	232	0.00
50	10	0.07	1.32	119	0.15	0.25	6.02	187	0.02	0.53	8.93	200	0.00
50	20	1.64	8.06	113	0.08	2.52	9.03	50	0.01	1.47	7.41	125	0.00
100	3	0.01	2.14	224	3.74	0.01	0.78	218	0.67	0.03	3.16	246	0.05
100	4	0.01	0.79	220	1.49	0.02	3.28	221	0.22	0.04	2.87	240	0.02
100	5	0.00	0.02	235	1.35	0.02	1.29	211	0.16	0.06	2.84	244	0.02
100	10	0.01	0.21	222	0.71	0.02	1.45	235	0.07	0.89	17.94	212	0.01
100	20	0.04	1.81	142	0.64	0.10	6.38	218	0.07	1.00	10.53	192	0.01
100	40	1.59	7.13	112	0.27	2.46	9.51	66	0.03	1.87	9.26	107	0.00
100	50	0.76	8.94	226	0.03	1.57	11.22	155	0.01	1.95	10.11	125	0.00
200	3	0.00	0.47	203	33.31	0.00	0.15	217	1.95	0.03	3.71	251	0.26
200	4	0.01	1.04	219	6.16	0.00	0.04	225	1.01	0.02	1.39	254	0.10
200	5	0.00	0.01	230	4.91	0.01	0.85	217	0.76	0.05	2.96	244	0.10
200	10	0.00	0.01	239	2.39	0.01	0.09	238	0.32	0.11	7.23	242	0.04
200	20	0.00	0.15	231	2.96	0.01	0.46	239	0.42	0.41	11.58	236	0.03
200	40	0.04	2.12	161	3.42	0.05	4.15	243	0.35	0.19	6.73	246	0.03
200	50	0.05	1.28	123	3.72	0.17	4.55	193	0.41	0.26	4.88	227	0.03
Avg/Tot		0.39	24.14	4659	2.57	0.61	25.36	4725	0.26	0.70	26.67	5296	0.03

Table 6.10: Overall performance of U* and L* – processing time classes

its were generated according to K_7–K_9 instead of K_1–K_3 or K_4–K_6 although our reduction procedures were not that effective for the classes K_7–K_9 (cf. Table 6.9). Altogether, by means of U*=L*, we were able to identify optimal solutions for 14680 out of the 21060 test instances (i.e. almost 70%) and the overall relative gap averaged about 0.57%. While configurations with $n/m > 5$ turned out to be rather simple to solve (we found optimal solutions for 9440 out of the corresponding 11340 instances, i.e. more than 83%), the case $n/m = 2.5$ appears to be particularly difficult as we were able to solve only 1154 out of the corresponding 3240 instances (i.e. less than 36%) and the relative gap between U* and L* averaged about 2.09%. The greatest relative gaps of up to 27% were also recorded for instances with a small ratio of n to m. The conspicuous behavior of our algorithm on such instances is not that surprising. Haouari and Jemmali (2008a) and Walter et al. (2016) also report on similar observations.

Looking at the computational effort of our algorithm, we see that k_i-LS requires less time when processing times are small and/or the ratio of n to m is small. These cases typically result in smaller values of C (cf. Section 6.4.2) which itself strongly impacts the time requirement of k_i-LS's sub-routine k_i-DP. Clearly, the smaller C, the faster k_i-DP determines a solution. Consequently, the computation time of k_i-LS is higher for the processing time classes T_1–T_3 which generate larger processing times on average than the other time classes. Furthermore, the computation time increases when (i) n increases and m is fixed and (ii) n and m increase and n/m is fixed. In both cases the number of pairs

		K_1–K_3				K_4–K_6				K_7–K_9			
n	m	%GAP	MAX	#OPT	TIME	%GAP	MAX	#OPT	TIME	%GAP	MAX	#OPT	TIME
10	3	1.12	9.03	94	0.00	1.00	8.21	77	0.00	1.17	11.17	124	0.00
10	4	2.39	23.20	113	0.00	2.51	25.36	98	0.00	2.35	24.14	102	0.00
10	5	0.87	13.68	224	0.00	2.68	26.67	160	0.00	1.62	20.00	195	0.00
25	3	0.05	1.35	211	0.05	0.03	1.27	216	0.04	0.12	4.69	224	0.04
25	4	0.04	1.22	205	0.03	0.05	1.38	201	0.02	0.39	16.04	189	0.03
25	5	0.18	1.75	137	0.02	0.09	2.38	142	0.02	0.47	12.27	172	0.02
25	10	1.83	9.97	120	0.01	2.22	14.56	73	0.01	2.24	9.52	79	0.01
50	3	0.02	0.49	209	0.24	0.02	0.55	205	0.24	0.06	5.77	233	0.20
50	4	0.02	0.70	219	0.12	0.01	0.54	217	0.11	0.04	3.90	237	0.14
50	5	0.02	0.85	211	0.11	0.01	0.44	225	0.09	0.20	8.49	225	0.12
50	10	0.15	3.23	138	0.06	0.04	0.56	181	0.04	0.65	8.93	187	0.07
50	20	1.57	7.41	124	0.02	1.99	6.80	80	0.03	2.08	9.03	84	0.04
100	3	0.01	0.35	228	1.55	0.00	0.06	231	1.52	0.05	3.16	229	1.39
100	4	0.01	0.33	224	0.51	0.00	0.09	230	0.50	0.06	3.28	227	0.72
100	5	0.01	0.24	215	0.48	0.00	0.12	235	0.41	0.07	2.84	240	0.65
100	10	0.03	1.03	221	0.20	0.01	0.25	225	0.15	0.87	17.94	223	0.44
100	20	0.10	2.76	160	0.13	0.03	1.61	191	0.07	1.00	10.53	201	0.51
100	40	1.46	8.48	124	0.06	2.03	9.51	82	0.09	2.43	9.26	79	0.15
100	50	0.48	6.25	230	0.01	2.08	11.22	115	0.02	1.71	10.11	161	0.02
200	3	0.00	0.14	217	12.65	0.00	0.03	219	12.41	0.03	3.71	235	10.46
200	4	0.00	0.16	225	2.15	0.00	0.03	229	1.87	0.03	1.39	244	3.25
200	5	0.00	0.24	227	1.71	0.00	0.05	229	1.41	0.05	2.96	235	2.65
200	10	0.01	0.47	232	0.70	0.00	0.05	242	0.57	0.10	7.23	245	1.47
200	20	0.01	0.89	236	0.37	0.01	0.22	230	0.25	0.40	11.58	240	2.78
200	40	0.10	4.15	179	0.36	0.03	1.79	214	0.13	0.15	6.73	257	3.31
200	50	0.26	4.55	139	0.42	0.08	2.22	148	0.18	0.14	4.88	256	3.56
Avg/Tot		0.41	23.20	4862	0.84	0.57	26.67	4695	0.78	0.71	24.14	5123	1.23

Table 6.11: Overall performance of U^* and L^* – cardinality classes

of machines that has to be investigated within the local search part is increasing.

	%BESTi						%OPTi					
	UB_1	$\widetilde{UB}_1$	UB_2	UB_3	$\widetilde{UB}_3$	UB_4	UB_1	$\widetilde{UB}_1$	UB_2	UB_3	$\widetilde{UB}_3$	UB_4
T_1	82.95	85.26	39.91	13.59	20.04	9.66	44.27	46.54	24.23	13.50	19.87	9.53
T_2	78.68	82.74	36.71	16.15	26.15	14.23	45.38	49.23	23.08	15.77	24.87	13.16
T_3	67.31	74.36	34.79	20.94	39.79	26.88	48.03	55.00	23.29	20.85	39.62	26.79
T_4	78.85	87.95	41.97	15.38	32.01	19.02	53.68	61.58	30.38	15.04	29.06	16.24
T_5	85.90	90.81	47.65	11.20	20.56	10.56	52.35	56.75	32.01	10.85	19.27	9.57
T_6	85.85	90.38	47.65	10.09	18.29	9.10	51.15	55.60	30.56	9.66	17.48	8.59
T_7	78.63	98.38	54.83	5.09	11.24	6.50	61.71	81.20	44.40	4.53	9.96	5.68
T_8	80.38	96.92	52.82	5.34	11.24	6.41	55.94	72.05	38.76	4.87	9.74	5.30
T_9	81.88	94.57	54.66	4.32	10.00	6.45	51.45	63.68	36.28	4.10	9.06	5.68
Avg	80.05	89.04	45.66	11.34	21.04	12.09	51.55	60.18	31.44	11.02	19.88	11.17

Table 6.12: Performance of the upper bound procedures – processing time classes

In the last part of our computational study, we carefully evaluate the effectiveness of our upper bound as well as lifting procedures. The respective results are presented in the Tables 6.12 – 6.15. The first two tables provide for each of our upper bounds (except UB_0 and UB_5) the relative number of instances for which the respective bound is equal to U^* and L^*, respectively. The last two tables display the impact of the lifting procedures by

	%BESTi						%OPTi					
	UB_1	$\widetilde{UB}_1$	UB_2	UB_3	$\widetilde{UB}_3$	UB_4	UB_1	$\widetilde{UB}_1$	UB_2	UB_3	$\widetilde{UB}_3$	UB_4
K_1	82.65	89.79	56.41	1.62	9.02	10.04	53.89	60.90	39.40	1.58	8.68	9.70
K_2	87.48	93.46	57.14	0.81	3.21	3.21	53.42	59.27	40.26	0.73	2.48	2.48
K_3	87.18	92.82	49.83	2.61	5.09	3.03	51.97	57.31	36.62	2.48	4.66	2.74
K_4	85.13	92.56	59.96	1.75	5.00	5.43	55.13	62.31	42.31	1.67	4.27	4.74
K_5	85.34	92.65	62.01	1.07	5.17	5.68	56.75	63.85	45.30	0.94	4.79	5.30
K_6	85.68	92.86	60.09	1.37	5.21	5.64	55.38	62.26	42.52	1.28	4.62	5.04
K_7	63.29	78.80	13.55	36.41	63.42	32.09	44.49	59.06	6.79	35.51	60.21	29.66
K_8	70.98	83.55	23.16	31.20	50.56	22.95	46.54	58.59	12.22	30.64	48.63	21.50
K_9	72.69	84.87	28.85	25.26	42.65	20.73	46.41	58.08	17.56	24.36	40.60	19.40
Avg	80.05	89.04	45.66	11.34	21.04	12.09	51.55	60.18	31.44	11.02	19.88	11.17

Table 6.13: Performance of the upper bound procedures – cardinality classes

	#ImpLift		#<$\widetilde{UB}_1$		
	UB_1	UB_3	UB_2	$\widetilde{UB}_3$	UB_4
T_1	146	947	10	169	169
T_2	211	929	12	239	236
T_3	381	933	13	460	459
T_4	363	1335	1	222	221
T_5	216	1455	2	131	131
T_6	200	1528	11	112	112
T_7	486	1845	2	30	30
T_8	430	1822	2	47	47
T_9	357	1815	10	66	66
Tot	2790	12609	63	1476	1471

Table 6.14: Performance of the lifting procedures and improvement of $\widetilde{UB}_1$ – processing time classes

	#ImpLift		#<$\widetilde{UB}_1$		
	UB_1	UB_3	UB_2	$\widetilde{UB}_3$	UB_4
K_1	282	1936	7	163	162
K_2	168	1617	4	45	45
K_3	152	1175	5	38	37
K_4	217	1749	6	88	88
K_5	221	1638	3	88	88
K_6	208	1532	4	95	95
K_7	632	1059	12	406	404
K_8	472	958	14	287	287
K_9	438	945	8	266	265
Tot	2790	12609	63	1476	1471

Table 6.15: Performance of the lifting procedures and improvement of $\widetilde{UB}_1$ – cardinality classes

counting (i) the number of instances where $\widetilde{UB}_i < UB_i$ (for $i \in \{1, 3\}$) and (ii) also the number of instances where $UB_i < \widetilde{UB}_1$ (for $i \in \{2, 4\}$) and $\widetilde{UB}_3 < \widetilde{UB}_1$.

To allow for a fair comparison of the different upper bounds, we slightly modified the execution scheme in that we did not abort the algorithm as soon as an optimal solution is found but applied each upper bound except of UB_0 and UB_5 since UB_0 can only be calculated in special cases and UB_0 is also dominated by $\widetilde{UB}_1$. Determining UB_5 is also not meaningful since naturally $U^* = UB_5$ holds.

As can be seen from Tables 6.12 and 6.13, the rather simple LP-based bound UB_1 and its lifted version $\widetilde{UB}_1$ are outperforming the other bounds. In terms of numbers, we observed that $\widetilde{UB}_1 = U^*$ for 89.04% of the 21060 problem-instances and $\widetilde{UB}_1 = L^*$ for 60.18%. Considering the other, more specialized bounds and their performance with respect to the different cardinality classes it is interesting to note that UB_2 performs quite well for the classes K_1–K_6 and rather poor for K_7–K_9 whereas the opposite is the case with the three other bounds. Obviously, the poor quality of UB_3 and UB_4 for K_1–K_6 is caused by the distribution of the k_i-values. Furthermore, Tables 6.12 and 6.13 already

reveal the significant improvement of UB_1 and UB_3 by application of the combined lifting procedures. From the last two Tables 6.14 and 6.15 we see that lifting UB_1 improves the bound for a total of 2790 out of 21060 instances (i.e. 13.25%) while lifting UB_3 even improves its non-lifted version for a remarkable number of 12609 instances (i.e. 59.87%) so that application of the lifting procedures is well justified.
Although $\widetilde{UB}_1$ equals the best upper bound in almost 90%, the last two tables reveal that there exist some instances (about 7%) where $\widetilde{UB}_3$ and/or UB_4 yield a better upper bound. Most of these instances belong to the class T_3 and K_7, respectively. So, application of these two bounds is also justified whereas we found that it is not beneficial to also apply UB_2 since it cannot improve on the best upper bound value in about 99.7% although UB_2 equals the best bound in about 45%.

6.6 Conclusion

The present Chapter treats the k_i-partitioning problem with the objective of maximizing the minimum completion time subject to machine-dependent upper cardinality limits k_i on the maximum number of jobs that can be assigned to machine i. To tackle this problem we developed powerful preprocessing procedures which proved to be able to reduce the problem's size by eliminating machines and jobs in advance. One of our preprocessing steps is to exploit the implicitly given lower cardinality limits l_i in order to enhance the upper limits. We also derived several upper bound arguments and proposed effective lifting procedures which often led to even tighter bounds in our experiments. When it comes to generating high-quality solutions to the k_i-partitioning problem, we designed a powerful subset sum based improvement procedure k_i-LS whose core is built by our exact dynamic programming procedure k_i-DP that optimally solves the two-machine case. Computational tests on a large set of randomly generated instances attest to the efficacy of our solution methods: reductions were obtained for about 12% of the instances, the lifting procedures helped to tighten the best upper bound in about 62% of the cases, and the overall relative gap between the best upper and the best lower bound averages 0.57% while we optimally solved at least 70% of the instances within less than one second of computation time on average.

We see the following directions for future research. From a theoretical point of view, analyzing the worst case behavior of our upper bound procedures as well as the two LPT-based construction heuristics constitutes a challenging task. From an algorithmic point of view, we believe that further improvements in the quality of the generated solutions can be obtained by tailor-made metaheuristics or sophisticated exact algorithms. Moreover, we suggest two ideas that potentially result in even tighter bounds. The first one is to enhance the lifting procedures by explicitly using the information provided by the gaps $k_i - l_i$ between the upper and lower cardinality limits. The second idea is to combine the preprocessing and lifting procedures by applying the reduction criteria not only to the given problem instance but also to the derived partial instances.

Chapter 7

Lower bounds and algorithms for the minimum cardinality bin covering problem

Summary

This Chapter introduces the minimum cardinality bin covering problem where we are given m identical bins with capacity C and n indivisible items with integer weights w_j $(j = 1, \ldots, n)$. The objective is to minimize the number of items packed into the m bins so that the total weight of each bin is at least equal to C. We discuss reduction criteria, derive several lower bound arguments and propose construction heuristics as well as a powerful subset sum-based improvement algorithm that is even optimal when $m = 2$. Moreover, we present a tailored branch-and-bound method which is able to solve instances with up to 20 bins and several hundreds of items within a reasonable amount of time. In a comprehensive computational study on a wide range of randomly generated instances, our algorithmic approach proved to be much more effective than a commercial solver.

7.1 Introduction

7.1.1 Problem definition

In this article we address the *minimum cardinality bin covering problem* (MCBCP) which consists in determining the least number of items necessary to fill (or cover) m bins. More precisely, given $m \geq 2$ identical bins of capacity $C \in \mathbb{N}$ and a set $\mathcal{J}$ of n indivisible items with integer weights $w_j \in \mathbb{N}$ $(j = 1, \ldots, n)$ the objective is to minimize the number of items packed into the m bins so that the total weight of each bin equals at least C. Introducing binary variables x_{ij} which take the value 1 if item j is packed into bin i and 0 otherwise, a straightforward formulation of the MCBCP as an integer linear program

(ILP) consisting of objective function (7.1) subject to (7.2)–(7.4) is provided below.

$$\text{Minimize } z = \sum_{i=1}^{m} \sum_{j=1}^{n} x_{ij} \tag{7.1}$$

subject to

$$\sum_{j=1}^{n} w_j x_{ij} \geq C \qquad i = 1, \ldots, m \tag{7.2}$$

$$\sum_{i=1}^{m} x_{ij} \leq 1 \qquad j = 1, \ldots, n \tag{7.3}$$

$$x_{ij} \in \{0, 1\} \qquad i = 1, \ldots, m;\ j = 1, \ldots, n \tag{7.4}$$

Objective function (7.1) minimizes the number of items necessary to fill all m bins. Constraints (7.2) ensure that each bin is filled and constraints (7.3) guarantee that each item is assigned to at most one bin. Finally, the domains of the binary variables are set by (7.4). By reduction from 3-Partition (cf. Garey and Johnson, 1979) it is readily verified that MCBCP is $\mathcal{NP}$-hard. Throughout the Chapter, we assume the items to be labeled in such a way that $w_1 \geq w_2 \geq \ldots \geq w_n > 0$. Moreover, for economy of notation, we often identify items by their index.

As a possible application of MCBCP, consider the disposal or transportation of m different liquids (e.g., chemicals) that cannot be mixed. If at least C volume units of each liquid have to be transported and we are given n tanks of various sizes, the MCBCP is to load the m liquids into the fewest number of tanks. Clearly, the less tanks are used the more convenient the handling and the less organizational effort. Note that here, the "liquids" correspond to bins and the "tanks" (and their sizes) correspond to the items (and their weights). For the closely related *liquid loading problem* we refer to Christofides et al. (1979).

7.1.2 Related work

Problem MCBCP can be seen as the dual version of the *maximum cardinality bin packing problem* (MCBPP) which consists in determining the maximum number of indivisible items that can be packed into the m bins so that the total weight of each bin does not exceed C. The MCBPP has been widely studied in terms of upper bounds and exact solution procedures (Labbé et al., 2003, Peeters and Degraeve, 2006), worst-case performance of heuristics (Coffman et al., 1978, Coffman and Leung, 1979, Langston, 1984), probabilistic analyses (Bruno and Downey, 1985, Foster and Vohra, 1989, Rhee and Talagrand, 1993), meta heuristics (Loh et al., 2009), and innovative applications (Vijayakumar et al., 2013). In contrast to the variety of publications on MCBPP, to the best of our knowledge we are the first to study its dual version MCBCP which has recently been mentioned for the first time by Coffman et al. (2013) as a natural variant of the bin covering problem (BCP).

The BCP itself has been introduced by Assmann et al. (1984) as the dual version of the classical one-dimensional bin packing problem. So far, related work on variants of the BCP stem from, e.g., Fukunaga and Korf (2007) who considered the problem of minimizing

the total cost of the used items to cover m variable sized bins, Csirik et al. (2010) who examined the online and two semi-online versions of the problem of minimizing the total weight of the items used to cover m bins, Epstein et al. (2010) who investigated a class constrained bin covering problem where each item has a color associated with it and the goal is to cover as many bins as possible subject to the constraint that the total number of distinct colors in each bin has to be at least l, and Epstein et al. (2013) who studied the cardinality constrained bin covering problem which consists in maximizing the number of covered bins subject to the constraint that each bin must contain at least k items.

Returning to the MCBCP and its ILP formulation, it becomes obvious that MCBCP belongs to the class of mixed packing covering integer programs which are formally defined as:

$$\text{Minimize } z = c^T x \tag{7.5}$$

subject to

$$\begin{aligned} Ax &\geq a && (7.6)\\ Bx &\leq b && (7.7)\\ x &\leq d && (7.8)\\ x &\in \mathbb{Z}^N_{\geq 0} && (7.9) \end{aligned}$$

where $A \in \mathbb{R}^{M\times N}_{\geq 0}$, $B \in \mathbb{R}^{R\times N}_{\geq 0}$, $a \in \mathbb{R}^M_{>0}$, $b \in \mathbb{R}^R_{\geq 0}$, $c \in \mathbb{R}^N_{\geq 0}$, and $d \in \mathbb{R}^N_{>0}$ (cf. Kolliopoulos and Young, 2005). The constraints (7.6), (7.7), and (7.8) are called covering, packing, and multiplicity (or capacity) constraints, respectively. Note that the multiplicity constraints can also be modeled by adding (at most) N rows to B – one for each constraint $x_j \leq d_j (< \infty)$. This equivalent notation appeared in Kolliopoulos and Young (2001) under the name Covering Integer Problems (CIP) with generalized multiplicity constraints. However, it is also important to note that a packing constraint cannot be multiplied by -1 in order to be turned into a covering constraint because the problem definition presupposes non-negative data.

Setting $M = m$, $R = n$, $N = mn$, $a = (C, \ldots, C)$ (M elements), $b = (1, \ldots, 1)$ (R elements), $c = d = (1, \ldots, 1)$ (N elements), and

$$a_{\bar{i}\bar{j}} := \begin{cases} w_j & \text{for } \bar{i} = 1, \ldots, m \text{ and } \bar{j} = (\bar{i}-1)n + j \quad (j \in \{1, \ldots, n\}) \\ 0 & \text{else,} \end{cases}$$

$$b_{\bar{k}\bar{j}} := \begin{cases} 1 & \text{for } \bar{k} = 1, \ldots, n \text{ and } \bar{j} = \bar{k}, n+\bar{k}, \ldots, (m-1)n + \bar{k} \\ 0 & \text{else} \end{cases}$$

as well as $x = (x_{11}, \ldots, x_{1n}, x_{21}, \ldots, x_{2n}, \ldots, x_{m1}, \ldots, x_{mn})$ it is readily verified that MCBCP is a representative of mixed packing covering integer problems. Note that in our case the multiplicity constraints (7.8) are redundant.

A special case of CIPs with generalized multiplicity constraints is obtained when B is the $N \times N$-identity matrix, i.e. when no packing constraints are existent. This problem version (i.e. (7.5), (7.6), (7.8), and (7.9)) is called CIP with multiplicity constraints and has been studied by, e.g., Dobson (1982), Kolliopoulos and Young (2001), and Kolliopoulos (2003). When no multiplicity constraints are considered at all (i.e. (7.5),(7.6), and (7.9))

we speak of the classical CIP (see, e.g., Srinivasan, 1999). At this point it is important to note that due to the presence of the constraints (7.3) we cannot formulate the MCBCP either as a CIP or as a CIP with multiplicity constraints.

With regards to algorithms for solving CIPs with generalized multiplicity constraints (and thus the MCBCP), to the best of our knowledge, merely three approximation algorithms are to be found in the literature for which, however, only analytical results are available but no computational ones. The first algorithm stems from Kolliopoulos and Young (2001) and is based on "finely" rounding a fractional optimal solution. The authors showed that for any $\varepsilon > 0$, an integral solution $\hat{x}$ of cost $\mathcal{O}\left(\max\{1, 1/\varepsilon^2\}(1 + (\log M)/W)\right)$ times the optimum of the linear programming relaxation can be obtained in deterministic polynomial time which satisfies $A\hat{x} \geq a$ and $(B\hat{x})_r \leq \left\lceil (1+\varepsilon)b_r + \mathcal{O}\left(\min\{\varepsilon^2, 1\}\beta_r W/(\log M)\right)\right\rceil$ for all $r = 1, \ldots, R$ where β_r is the sum of coefficients at the r-th row of B and W is defined as $\min\{a_i/A_{i,j} \mid A_{i,j} > 0, i = 1, \ldots, M, j = 1, \ldots, N\}$. So, obviously, the solution returned by this algorithm cannot guarantee to meet all packing constraints. Note that we have $\beta_r = n$ for all r and $W = C/w_1$ in the MCBCP.
The other two methods are bi-criteria approximation algorithms and presented in Kolliopoulos and Young (2005). According to the authors, for any $\varepsilon \in (0, 1]$, the second algorithm finds a solution $\hat{x}$ of cost $\mathcal{O}\left(1 + ln(1 + \alpha)/\varepsilon^2\right)$ times the optimum, satisfying $A\hat{x} \geq a$, $B\hat{x} \leq (1 + \varepsilon)b + \beta$, and $\hat{x} \leq d$ where $\beta = (\beta_1, \ldots, \beta_R)$ and α is the maximum number of covering constraints that any variable appears in. Note that we have $\alpha = 1$ in the MCBCP. Again, this algorithm cannot guarantee that its returned solution meets the packing constraints. The same applies as well to the third one which, indeed, has a better (asymptotic) cost guarantee but even violates the multiplicity constraints. Therefore, we omit any further algorithmic details and refer to Kolliopoulos and Young (2005).

In contrast to the scarce literature on CIPs with generalized multiplicity constraints there exists a considerable body of literature on CIPs/CIPs with multiplicity constraints and several specific variants thereof which, for instance, differ in restrictions on the domains of the input data. However, as our MCBCP cannot be formulated as a CIP (with multiplicity constraints), we abstain from reviewing algorithms for solving CIPs (with multiplicity constraints).

Summarizing, as can be seen from the aforementioned analytical results, the three algorithms introduced in Kolliopoulos and Young (2001) and Kolliopoulos and Young (2005) are only appropriate when B has small row sums or when no packing constraints have to be taken into account. Since neither is the case with MCBCP, application of the existing methods can yield solutions where the packing constraints are violated by a large factor. Hence, there is an obvious need for suited solution procedures.

7.1.3 Contribution and Chapter structure

It is the intention of this Chapter to provide the required algorithms that are capable of generating feasible and high-quality solutions within short computation times. Therefore, we propose suited construction heuristics, a highly effective subset sum-based improvement algorithm that is even optimal when $m = 2$, and we also present an exact branch-and-bound method for which we develop tight lower bound procedures and specific dominance criteria. Our exact method is able to solve instances with up to 20 bins and several hundreds of items within a reasonable amount of time and clearly outperforms a commercial

solver.

The remainder of the Chapter is organized as follows. In Section 7.2, we describe reduction criteria and in Section 7.3 we derive several lower bound arguments. Then, Section 7.4 introduces tailor-made construction and improvement heuristics as well as a branch-and-bound algorithm, whose performances are tested and compared with the one of a commercial solver in a comprehensive computational study (Section 7.5). Finally, Section 7.6 concludes the Chapter.

7.2 Reduction criteria

In this section, we describe ideas to preprocess a given instance and to reduce its size or dimension. Without loss of generality, we assume all weights to be strictly smaller than C, i.e., $w_j < C$ $(j = 1, \ldots, n)$, because any item with $w_j \geq C$ already covers (or fills) an empty bin and thus can be assigned alone to a bin. A direct consequence of which is that in any feasible solution each bin contains at least two items. Moreover, note that if z^* is the optimal objective function value of a given instance of MCBCP, then there exists an optimal solution which contains the z^* largest items, i.e. the first z^* items in the sorted list (or, equivalently, the list's prefix of length z^*).

Within our preprocessing step, we check at first (in pseudo-polynomial time) whether there exists a subset $\mathcal{J}' \subseteq \mathcal{J}$ so that $\sum_{j \in \mathcal{J}'} w_j = C$. If no such subset exists, then C can feasibly be increased to the smallest realizable sum of weights that is greater than the original bin capacity. This will potentially lead to tighter lower bounds. Afterward, we apply the following two reduction criteria which have originally been proposed by Labbé et al. (1995) for the classical bin covering problem. As will be briefly shown next, they also apply to the MCBCP.

Criterion 7.2.1
If $w_k + w_l = C$ for some k and l, then there exists an optimal solution in which a bin contains only the two items k and l.

Proof
We distinguish three cases.

1. Consider an optimal solution which contains neither k nor l.

   Clearly, in this case each bin must contain exactly two items. Thus, there exists also an optimal solution in which a bin contains the two items k and l.

2. Consider an optimal solution which contains only one of the two items.

   Without loss of generality, assume that k is contained. Let S_k denote the set of items in the same bin as k. Then, the cardinality of S_k must be one. Hence, interchanging the respective item in S_k and l yields another optimal solution.

3. Consider an optimal solution which contains both of the two items in different bins.

   Let S_k and S_l denote the set of items in the same bin as k and l, respectively. Then, interchanging S_k and l yields a solution with the same objective function value.

■

Criterion 7.2.2
Let k be the maximum index so that $w_1 + \sum_{j=k}^{n} w_j \geq C$. If $w_1 + w_k \geq C$, then there exists an optimal solution in which a bin contains only the two items 1 and k.

Proof
Note that there exists an optimal solution containing item 1. Let S_1 denote the set of items in the same bin as item 1. If $k \notin S_1$, then S_1 must contain one of the items $2, \ldots, k-1$. Interchanging S_1 and k results in an equivalent solution. ■

In what follows, we assume that all reduction criteria have been applied and the remaining items are numbered from 1 to n.

7.3 Lower bound procedures

We also derive several lower bound arguments which help us to benchmark our heuristic solution algorithms (see Section 7.4.1 and 7.4.2) and to reduce the enumeration effort for our exact algorithm (see Section 7.4.3).

7.3.1 Combinatorial bounds

An immediate lower bound on the optimal number of items z^* is

$$L_0 = 2m \tag{7.10}$$

which follows directly from $w_j < C$ for all j. Clearly, an improved version of L_0 is

$$L_0' = \alpha m \tag{7.11}$$

where $\alpha = \min\{k : \sum_{j=1}^{k} w_j \geq C\}$ represents the minimum number of items that are required to cover a single bin.

Another straightforward lower bound (denoted by L_1) is obtained by solving the continuous relaxation of (7.1)–(7.4) (where (7.4) is replaced by $0 \leq x_{ij} \leq 1$ for $i = 1, \ldots, m$ and $j = 1, \ldots, n$) and rounding up the respective objective value. It is easy to observe that equality

$$L_1 = \min\{k : \sum_{j=1}^{k} w_j \geq mC\} \tag{7.12}$$

holds. Thus, L_1 can be computed in $\mathcal{O}(n)$ time assuming presorted items.

Before we proceed with the development of other combinatorial lower bounds, we briefly consider the worst-case performance of the bound $\max\{L_0', L_1\}$. As will be revealed by the following example, the worst-case performance is arbitrarily bad, i.e. the ratio $\max\{L_0', L_1\}/z^*$ can be arbitrarily close to zero for any fixed number of bins $m \geq 2$.

Example 7.3.1
Let K be a positive integer. Consider the following family of instances: $w_1 = \ldots = w_{m+1} = Km$, $w_{m+2} = \ldots = w_n = 1$ (n sufficiently large), and $C = K(m+1)$. Then, it is readily

verified that $\max\{L_0', L_1\} = \max\{2m, m+1\} = 2m$ *and* $z^* = m + 1 + (m-1)K$. *Thus, the ratio*

$$\frac{\max\{L_0', L_1\}}{z^*} = \frac{2m}{m+1+(m-1)K} = \frac{2m}{(K+1)m-(K-1)} = \frac{2}{(K-1)(1-1/m)+2}$$

converges to zero when K *tends to infinity.*

From the previous Example 7.3.1 we see that an arbitrarily large ratio q of the largest to the smallest item weight, i.e. $q = w_1/w_n$, can result in an arbitrarily bad worst-case performance of L_1. Basically, as may be imagined, the performance of the lower bound L_1 strongly depends on the ratio q, i.e. the smaller q the better the worst-case performance. The next proposition establishes a relation between q and L_1/z^* and provides a lower bound on the worst-case performance of L_1 depending on q.

Proposition 7.3.2
Let $q = \lceil w_1/w_n \rceil$, *then* $L_1/z^* > 1/(1+q)$.

Proof
The proof is quite simple. Consider the "continuous" solution obtained by packing items in non-increasing order of weights and allow them to be split between two bins. We can use this solution to construct a feasible solution to the binary program as follows. At first, each split-item is assigned to the lower-indexed bin that already contains one part of the respective item. Afterward, at most $m-1$ bins are *uncovered*, i.e., their total weight is less than C. However, we also know that the total weight of each uncovered bin is not less than $C - w_1$. Then, since $w_1/w_n \leq q$, it suffices to assign (at most) q additional items to each uncovered bin so that the total weight of each bin is at least C. Thus, the minimum number of items necessary to cover the m bins is not greater than $L_1 + (m-1)q$ and we obtain the estimation:

$$\frac{L_1}{z^*} \geq \frac{L_1}{L_1+(m-1)q} = \frac{1}{1+\frac{m-1}{L_1}q} > \frac{1}{1+q}.$$

■

We believe that the previous analysis can be tightened in order to obtain even stronger bounds on the worst-case performance of L_1 depending on the value of q. However, this is not within the scope of this Chapter but might be a topic for future research. Instead, we focus on another aspect that influences the performance of L_1. Assume that h items are of weight $\geq C/2$. In case that $h \in \{m+1, \ldots, 2m-1\}$ there is some potential to tighten L_1 since any two of the first h items already cover a bin. More precisely, the respective total weight in any such bin will be strictly greater than C (recall that the reduction criteria are assumed to be applied beforehand). Contrary to the assumption underlying L_1 the resulting excess cannot be used to fill other bins. Hence, we obtain the tightened bound

$$L_1' = \min\{k : \sum_{j=1}^{k} w_j \geq mC + E\}$$

where

$$E := \begin{cases} \sum_{j=2m-h+1}^{h} w_j - (h-m)C & \text{if } h \in \{m+1, \ldots, 2m-1\} \\ 0 & \text{else} \end{cases}$$

and $h := \max\{j : w_j \geq C/2\}$.

Next, we present a generalization of L_1 which can be computed in $\mathcal{O}(n)$ time as well.

Theorem 7.3.3
Let $Q(i) = \min\{k : \sum_{j=1}^{k} w_j \geq iC\}$. Then, $L_2(i) = Q(i) + (m-i)\lceil Q(i)/i \rceil$ is a valid lower bound on z^* for $i = 1, \ldots, m$.

Proof
Note that $Q(i)$ is a lower bound on the number of required items to cover i bins. Therefore, in any feasible solution, $Q(i)$ constitutes a lower bound on the number of items packed into the i covered bins that contain the least items. Each of the remaining $(m-i)$ bins will then contain at least $\lceil Q(i)/i \rceil$ items and the claim follows. ■

Clearly,

$$L_2 = \max_{i=1,\ldots,m} L_2(i) \tag{7.13}$$

is a valid lower bound on z^* and it is readily verified that L_2 is dominating L_1 since $L_2 \geq L_2(m) = Q(m) = L_1$. Although L_2 dominates L_1, its worst-case performance is not better than the one of L_1, i.e., the ratio L_2/z^* can be arbitrarily close to zero for any fixed $m \geq 2$. Considering the same family of instances as in Example 7.3.1, we readily obtain $L_2(i) = i + 1 + 2(m-i) = 2m + 1 - i$. Thus, $L_2 = 2m$ and the ratio L_2/z^* converges to zero when K tends to infinity (cf. Example 7.3.1).

A third bound is given by the next theorem.

Theorem 7.3.4
Let $p(i) = \min\{p : \sum_{h=i}^{i+p} w_h \geq C\} + 1$ and $P(i) = \sum_{h=1}^{i} p(h)$ for $i = 1, \ldots, m$. Then, $L_3(i) = P(i) + L_2^i$, where L_2^i denotes the L_2-bound for the reduced problem-instance on $m - i$ bins and the set of items $i + 1, \ldots, n$, is a valid lower bound on z^* for $i = 1, \ldots, m$.

Proof
Without loss of generality, assume the bins to be sorted (and labeled) in such a way that the largest item in bin i is at least as large as the largest item in bin $i + 1$ for all $i = 1, \ldots, m-1$. Then, clearly, the largest item in bin i cannot be larger than w_i. So, to cover bin i, at least $p(i)$ items are required. Hence, to cover the first i bins, at least $p(1) + \ldots + p(i) = P(i)$ items are necessary. The remaining $m - i$ bins require at least L_2^i items to be covered. ■

Consequently,

$$L_3 = \max_{i=1,\ldots,m} L_3(i) \tag{7.14}$$

is a valid lower bound on z^* and can be computed in $\mathcal{O}(mn)$ time.

Using once again the family of instances described in Example 7.3.1 it is readily verified that the worst-case performance of L_3 is arbitrarily bad, i.e., the ratio of L_3/z^* can be arbitrarily close to zero for any fixed $m \geq 2$. By hand calculation we obtain $L_3(i) = 2i + 2(m-i) = 2m$ for all $i = 1, \ldots, m$ since $P(i) = 2i$ and $L_2^i = 2(m-i)$. Note that $L_2^i(\bar{i}) = 2(m-i) + 1 - \bar{i}$ for $\bar{i} = 1, \ldots, m-i$ and all $i = 1, \ldots, m-1$. All in all, we obtain $L_3 = 2m$ and the ratio L_3/z^* converges to zero when K tends to infinity.

The following small example briefly shows that neither L_2 dominates L_3 nor L_3 dominates L_2.

Example 7.3.5
At first we treat the case $L_2 > L_3$. *Let* $C = 12$, $m \geq 2$ *fix,* $n = 2(m-1)+8$, *and assume the following weights:* $w_1 = \ldots = w_{2(m-1)} = 6$, $w_{2(m-1)+1} = w_{2(m-1)+2} = 3$, *and* $w_{2(m-1)+3} = \ldots = w_{2(m-1)+8} = 1$. *Then,* $L_2 = L_2(m) = 2(m-1)+8$ *while it is readily verified by hand calculation that* $L_3 = L_3(1) = 2(m-1)+3$.

Now, we consider the reverse case, i.e., $L_3 > L_2$. *Again, let* $C = 12$ *and* $m \geq 2$ *fix but this time we take* $n = 3(m-2)+8$ *items having the following weights:* $w_1 = 9$, $w_2 = 7$, $w_3 = \ldots = w_{3(m-2)+4} = 4$, *and* $w_{3(m-2)+5} = \ldots = w_{3(m-2)+8} = 1$. *Then, we readily obtain* $L_3 \geq L_3(m) = 3(m-1)+2 > 3(m-1)+1 = L_2(m) = L_2$.

We want to remark that the family of instances which leads to the poor worst-case performance of the developed lower bounds L_0, L_1, L_2, and L_3 is rather hypothetical. As will be demonstrated within Section 7.5.2, on randomly generated instances our combinatorial lower bounds are quite strong. Nevertheless, in the remainder of this section we are concerned with two other ideas to enhance the combinatorial bounds.

7.3.2 An enhanced lower bound based on bin covering

Next, we briefly describe how we can apply established upper bounds for the bin covering problem (BCP) in order to tighten a given lower bound value on z^*. The approach bases on the following matter of fact: If an upper bound U_{BCP} on the number of bins that can be covered by the L largest items is smaller than m, then $L+1$ is a feasible lower bound value for MCBCP. Thus, starting with the $L := \max\{L_2, L_3\}$ largest items we seek for the smallest number $L_{BCP} \in \{L, \ldots, n\}$ so that the corresponding U_{BCP}-value for the items $1, \ldots, L_{BCP}$ is greater or equal to m. Then, L_{BCP} is a feasible lower bound on z^*. In order to obtain preferably tight U_{BCP}-values, we implemented all relevant bounding procedures presented in Labbé et al. (1995) and Peeters and Degraeve (2006). For a detailed description of these bounds we refer to the respective literature.

7.3.3 Column generation lower bound

To possibly obtain even tighter bounds, we propose another approach based on column generation which, however, requires non-polynomial time and is based on a modified linear programming formulation using *patterns*. A pattern is a combination of items that can cover a bin. More formal, let $\mathcal{W}' = \{w'_1, \ldots, w'_{n'}\}$ denote the set of all pairwise different item weights and b_j $(j = 1, \ldots, n')$ the number of items of weight w'_j in $\mathcal{J}$, a pattern p is described by an integer array $(a_{1p}, \ldots, a_{n'p})$ where a_{jp} gives the number of items of weight w'_j that are used in pattern p and satisfies

$$\sum_{j=1}^{n'} a_{jp} w'_j \geq C \tag{7.15}$$

$$a_{jp} \in \mathbb{Z}_{\geq 0}. \tag{7.16}$$

We call a pattern *minimal* when using one item less results in a violation of (7.15). In what follows we only consider minimal patterns.
Let P denote the set of all minimal patterns and introduce integer variables x_p $(p \in P)$ that

count how often pattern p is used in a feasible solution, the pattern-based MCBCP-model reads as follows

$$\text{Minimize } z = \sum_{p \in P} \sum_{j=1}^{n'} a_{jp} x_p \tag{7.17}$$

subject to

$$\sum_{p \in P} x_p \geq m \tag{7.18}$$

$$\sum_{p \in P} a_{jp} x_p \leq b_j \qquad j = 1, \ldots, n' \tag{7.19}$$

$$x_p \in \mathbb{Z}_{\geq 0} \qquad p \in P. \tag{7.20}$$

As the number of feasible minimal patterns is exponential in n', it is prohibitive to enumerate all of them in advance. Instead, column generation techniques are often applied to solve problems such as the pattern-based MCBCP-model. We refer to Lübbecke and Desrosiers (2005) for a survey on selected topics in column generation.
Next, we briefly describe the basic technique for the pattern-based MCBCP and explain how to obtain a lower bound on the minimum number of items required to cover all m bins. First, we define the continuous relaxation of (7.17)–(7.20) – which is obtained by replacing (7.20) with $x_p \geq 0$ for all $p \in P$ – and we heuristically initialize the model with a reduced set of patterns $P' \subseteq P$ that provides a feasible solution. This results in the linear optimization problem

$$\text{Minimize } z_{LP} = \sum_{p \in P'} \sum_{j=1}^{n'} a_{jp} x_p \tag{7.21}$$

subject to

$$\sum_{p \in P'} x_p \geq m \tag{7.22}$$

$$\sum_{p \in P'} a_{jp} x_p \leq b_j \qquad j = 1, \ldots, n' \tag{7.23}$$

$$x_p \geq 0 \qquad p \in P' \tag{7.24}$$

which is also called the *restricted master problem* (RMP). After a solution to the RMP has been determined, let λ be the dual variable associated with constraint (7.22) and π_j be the dual variable associated with the j-th constraint of (7.23). To decide whether the solution is also optimal for $P' = P$ or whether there exits a pattern $\bar{p} \in P \setminus P'$ which could reduce the objective value of the RMP (*pricing problem*), we compute the *reduced costs* $\bar{c}_{\bar{p}} = -\lambda + \sum_{j=1}^{n'} a_{j\bar{p}}(1 + \pi_j)$ ($\bar{p} \in P \setminus P'$). Instead of explicitly pricing all candidate patterns $\bar{p} \in P \setminus P'$, we implicitly search for a pattern whose reduced costs are negative. For this purpose, we formulate and solve the following bounded knapsack problem that determines the pattern with the smallest reduced costs (*slave problem* (SP)):

$$\text{Minimize } z_{SP} = \sum_{j=1}^{n'} (1 + \pi_j) v_j \tag{7.25}$$

subject to

$$\sum_{j=1}^{n'} w'_j v_j \geq C \tag{7.26}$$

$$v_j \leq b_j \qquad j = 1, \ldots, n' \tag{7.27}$$

$$v_j \in \mathbb{Z}_{\geq 0} \qquad j = 1, \ldots, n'. \tag{7.28}$$

Note that the profits in SP correspond to the dual variables π_j, and v_j counts the number of times an item of weight w'_j is used. Let $v^* = (v_1^*, \ldots, v_{n'}^*)$ be the optimal solution to the slave problem. If $z_{SP}(v^*) < \lambda$, the pattern v^* has negative reduced costs and is inserted into P', i.e. $(1, v_1^*, \ldots, v_{n'}^*)$ is added as a new column to the RMP and the RMP is re-solved. The whole process is repeated until $z_{SP} \geq \lambda$, i.e. there exists no additional pattern that could further reduce the objective value of the RMP. Thus, the current solution is optimal for the continuous relaxation of the pattern-based MCBCP-model (7.17)–(7.20) and its rounded-up objective value $\lceil z_{LP} \rceil$ constitutes another lower bound on z^*. In the sequel, this bound is denoted by L_{CG}.

7.4 Algorithms

7.4.1 Construction heuristics

In the following, we propose three intuitive, simple heuristic algorithms to construct feasible solutions for MCBCP. All of them intend to avoid adding a large item to an almost covered bin which is also common to several construction heuristics for BCP (cf., e.g., Assmann et al., 1984, Csirik et al., 1999). Some of the BCP-heuristics – such as the *Simple* or the *Improved simple* heuristic developed by Csirik et al. (1999) – fill one or more (very) small items into an almost covered bin until the bin is covered. This clarifies why a direct application of BCP-heuristics to MCBCP might result in solutions with a rather poor quality. Therefore, to account for using preferably few items to cover a given number of bins, we suggest the three following procedures: *First Fit Decreasing* with parameter $r \in [1, 2)$ (abbreviated FFD_r), *Best Fit Decreasing* with parameter $r \in [1, 2)$ (abbreviated BFD_r), and *Lowest Fit Decreasing* (LFD). FFD_r as well as LFD have already been mentioned in Assmann et al. (1984) – however in the context of BCP. In what follows, we describe how we apply them to MCBCP.

We start with a description of FFD_r. While there are not all m bins covered, FFD_r puts the largest unpacked item into the first uncovered bin whose current level is not greater than $rC - \tilde{w}$ where $\tilde{w}$ denotes the weight of the largest unpacked item. If all uncovered bins' levels are greater than $rC - \tilde{w}$, put the item into the bin with the minimum level.

The second procedure, BFD_r, is a slight modification of FFD_r and works as follows. While there are not all m bins covered, BFD_r puts the largest unpacked item into that

uncovered bin whose current level is as small as possible among all bins with a current level ranging between $C - \tilde{w}$ and $rC - \tilde{w}$. If no such bin exists, then the item is packed into the bin with the overall minimum level.

Finally, the third procedure LFD differs from the other two in that it always puts the largest unpacked item into the bin with the minimum level until all m bins are covered. Assuming the items to be sorted in non-increasing order of their weights, the time complexity of each of the three procedures is $\mathcal{O}(n \log m)$.

7.4.2 A subset sum-based improvement heuristic

In order to improve solutions generated by our construction heuristics, we propose an improvement heuristic (dubbed SSH) that iteratively solves subset sum problems (cf. also Haouari and Jemmali, 2008a). Given a feasible solution, we, first, remove the overall smallest packed item. Note that the corresponding bin will be no longer covered. Afterward, we sort (and renumber) the bins so that $C_1 \geq \ldots \geq C_m$ (C_i denotes the level of bin i which is simply the sum of the items' weights contained in i) and we try to transform the currently infeasible solution into a feasible one that requires exactly one item less than the previous solution by re-assigning the packed items. For this purpose, we iteratively solve specific subset sum problems as follows. Starting with $i = 1$, we consider the bins i and m (for $i = 1, \ldots, k$ where $k = \max\{i : C_i > C\}$) and solve a subset sum problem according to (7.29)–(7.31) on the respective set of items $\mathcal{J}_{(i,m)} := \mathcal{J}_i \cup \mathcal{J}_m$ where $\mathcal{J}_{i'}$ denotes the set of items contained in bin i'.

$$\text{Minimize } z_i = \sum_{j \in \mathcal{J}_{(i,m)}} w_j x_j \tag{7.29}$$

subject to

$$\sum_{j \in \mathcal{J}_{(i,m)}} w_j x_j \geq C \tag{7.30}$$

$$x_j \in \{0, 1\} \qquad j \in \mathcal{J}_{(i,m)} \tag{7.31}$$

If $z_i = C_i$, i.e. considering only the two bins i and m it is impossible to increase the load of m without lowering the load of i below C, we increment i by one (unless $i = k$) and proceed with solving the respective subset sum problem for the next pair of bins. Otherwise, we set $\mathcal{J}_i = \{j \in \mathcal{J}_{(i,m)} : x_j = 1\}$, $\mathcal{J}_m = \{j \in \mathcal{J}_{(i,m)} : x_j = 0\}$, we resort the bins again according to non-increasing levels, redetermine k, and proceed with the pair of bins $(1, m)$. In the special case $C_m \geq C$, which means that an improved feasible solution (and thus an improved upper bound) has been found that requires exactly one item less than before, we additionally remove the smallest of the remaining packed items before the bins are resorted in order to check if further improvements are realizable.

Algorithm SSH either terminates if $z_k = C_k$ (after checking the pair of bins (k, m)) or a maximal number *it* of subset sum problems has been solved. Basically, good choices for the value of the parameter *it* depend on both, the number of bins m and the absolute gap between the best objective value generated by our construction heuristics and the best known lower bound. However, in our computational study (see Section 7.5), we simply

used $it = 50$ (in case of $m \leq 30$) and $it = 100$ (in case of $m = 50$) which turned out to be sufficiently large to yield strong upper bounds for almost all of the generated test problems. Moreover, we implemented a straightforward dynamic programming procedure to solve the respective subset sum problems.

We shall also remark that in case $m = 2$ our SSH-procedure turns into an exact algorithm when no limit on the maximal number it of iterations (or subset sum problems to be solved) is imposed. The proof is straightforward and therefore omitted.

7.4.3 Exact procedure

Next, we propose a tailored branch-and-bound algorithm for MCBCP whose branching strategy builds upon the one presented by Labbé et al. (1995) for the BCP. Our algorithm works as follows. At the root node, the given MCBCP-instance is preprocessed and the reduction criteria are applied (cf. Section 7.2). If the number of bins reduces to zero or one, then an optimal solution is already found or can readily be obtained by adding the largest unpacked items until the one remaining bin is finally covered. In all other cases where the number of bins is at least two in the reduced instance, we proceed with the application of FFD_r, BFD_r as well as LFD in order to obtain an upper bound U. Then, we calculate the trivial lower bounds L_0, L'_0, L_1 (and possibly L'_1) followed by application of SSH to improve on U. We take the best among the three constructed solutions as the initial one for SSH. Finally, we successively determine the remaining lower bounds L_2, L_3, L_{BCP}, and L_{CG}. To solve the RMPs and SPs within the column generation based lower bound procedure (see Sect. 7.3.3) we used Gurobi 6.0.3. Our algorithm immediately stops as soon as one of our lower bounds is equal to U, i.e. an optimal solution has already been identified at the root node. Otherwise, the branching process is initiated.

In quest of an optimal solution, we perform a depth-first search where at each level of the branching-tree the largest unpacked item is sequentially packed into at most m bins. More precisely, at level k, the current node generates at most m son-nodes by sequentially packing item k to each of the non-empty uncovered bins and – if still existent – to the lowest indexed empty bin. The order in which item k is packed into the bins depends on the ratio w_k/C. If $w_k/C \geq \theta$, then the possible bins are selected in order of increasing levels. Otherwise, the possible bins are selected in order of decreasing levels.

To obtain a local lower bound, at each node in the tree we compute L_2 and L_3 for a modified instance. The set of items in the modified instance consists of all currently unpacked items and for each non-empty uncovered bin a fictive item is added whose weight is equal to the sum of the item weights in the respective bin. The number of bins in the modified instance is $m - \bar{m}$ where $\bar{m}$ denotes the number of bins that are already covered in the partial solution that corresponds to the current node. Clearly, if L' is the value of a lower bound for the modified instance, then $L' + k - m'$ is a feasible local lower bound for the original instance where the largest k items have already been assigned and the current number of non-empty uncovered bins is m'. In case that $L' + k - m' \geq U$, the current partial solution is fathomed.

To further reduce the enumeration effort and to avoid symmetric reflections, we implemented the following straightforward dominance criteria:

- If at least two uncovered bins have the same current level, then only one of them is

considered for the assignment of the current item.

- Let $\tilde{C}_i$ denote the current level of bin i and $\tilde{I} = \left\{ i : C - w_k \leq \tilde{C}_i < C \right\}$. If $|\tilde{I}| > 1$, then the current item k is assigned only to the bin $i \in \tilde{I}$ with lowest level.
- If $w_k = w_{k-1}$ and item $k-1$ has been assigned to bin i', then, depending on the ratio w_k/C, item k is either assigned only to those bins i where $\tilde{C}_i \geq \tilde{C}_{i'} - w_{k-1}$ (case $w_k/C \geq \theta$) or $\tilde{C}_i \leq \tilde{C}_{i'} - w_{k-1}$ (case $w_k/C < \theta$).
- Let z be the best known objective function value. If after the assignment of the largest $z - m + i$ ($i = 0, \ldots, m-1$) items less than $i+1$ bins are covered, the partial solution can be fathomed.

We conducted some preliminary tests in order to experimentally determine the best values for the parameters

- r out of the set $\{1.50, 1.20, 1.15, 1.10, 1.05, 1.03, 1.01, 1.005, 1.001\}$ and
- θ out of the set $\{0, 0.05, 0.1, 0.15, 0.2, \ldots, 0.6\}$.

Based on the outcome of the tests we decided on $r = 1.005$ for FFD, $r = 1.01$ for BFD, and $\theta = 0.4$.

7.5 Computational study

This section elaborates on our computational tests where we examine the overall performance of our branch-and-bound algorithm (cf. Section 7.4.3) including the reduction criteria from Section 7.2, the lower bound arguments derived in Section 7.3, and the heuristics developed in Section 7.4.1 and 7.4.2.

7.5.1 Instance generation

As there is no established test bed available for MCBCP, we randomly generated test instances that comprise a wide range of settings, i.e. small-, medium-, and large-sized instances as well as various item weights drawn from different probability distributions. The test cases are classified according to three parameters: (i) number of bins m, (ii) bin capacity C, and (iii) probability distribution d from which the item weights are independently drawn. We investigated the following $7 \times 8 \times 6 = 336$ different combinations of the three parameters:

(i) $m \in \{2, 5, 10, 15, 20, 30, 50\}$

(ii) $C \in \{100, 120, 150, 200, 300, 500, 750, 1000\}$

(iii) d:

- discrete uniform distribution in (1) $[1, 99]$ and (2) $[20, 99]$
- cut-off normal distribution with (3) $\mu = 25$, $\sigma = 10$ and (4) $\mu = 50$, $\sigma = 25$

- Weibull distribution with (5) $k = 1.5$, $\lambda = 50$ and (6) $k = 2.5$ and $\lambda = 75$.

For each triple (m, C, d), 20 independent instances were randomly generated, i.e. we receive a total number of 6720 instances for our computational study. With regards to d, the parameters of the respective probability distributions have been chosen in such a way that for most of the 336 (m, C, d)-triples the average item weight is not too small, i.e. not smaller than about $C/10$. In a preliminary study we observed that, typically, instances with a small average item weight – which implies that many items are necessary to cover a bin – are quite easy to solve (even by means of simple heuristics).

To guarantee the existence of a feasible solution to each test instance, the item weights have been generated in $m + 1$ iterations. Within each iteration we independently draw random numbers w_j (> 0) from the respective probability distribution until the condition $\sum w_j \geq C$ is met. Note that m iterations would already be sufficient to ensure a feasible solution but we decided to generate the item weights in $m + 1$ rounds just to make sure that not almost all items will be needed to cover the m bins and to increase the solution space.

We have implemented our algorithms in C++ using the Visual C++ 2010 compiler. All tests have been carried out on a personal computer with an Intel Core i7-2600 processor, 8GB RAM, and Windows 7 Professional SP1 (64 bit). The maximal computation time per instance was set to 300 seconds.

7.5.2 Results

In what follows, we provide detailed results on the computational performance of our solution approach and we also compare them with those of a commercial solver. In order to assess the overall effectiveness of our reduction criteria as well as our lower and upper bound procedures, as a first performance criterion we recorded for each (m, C, d)-triple the number of instances that have already been solved at the root node.

The respective results are contained in Table 7.1 where we also give the maximum absolute gap between the best upper bound (denoted by U) and the best lower bound (denoted by L) at the root node if greater than 0 (cf. numbers in brackets). The entries in Table 7.1 clearly indicate that the effort spent at the root to preprocess a given instance and to compute lower as well as upper bounds is very well justified as we have already solved 6465 out of 6720 test instances, i.e. about 96.2%, at the root node. In other words, only 255 instances remained unsolved at the root node and required branching. As can be seen from Table 7.1, these 255 instances are spread over 74 of the 336 (m, C, d)-triples among which $(50, 100, 3)$ is the triple with the fewest number of instances solved at the root node with only 7 closely followed by $(30, 100, 3)$ and $(30, 150, 5)$ with 8 each. In contrast, the 262 triples for which all 5240 instances were solved at the root node include among others all triples where $m = 2$ (recall that SSH optimally solves such instances when it is sufficiently large) and all triples where $C \in \{500, 750, 1000\}$. It is not surprising that all instances with large-sized bins, i.e. $C \geq 500$, have already been solved at the root node. According to the distributions of the item weights, these instances require on average about 10 or even more items per bin and are, therefore, rather simple (cf. remark within Section 7.5.1).

d	m	C												**Total**	
		100		120		150		200		300		500	750	1000		
1	2	20		20		20		20		20		20	20	20	**160**	
	5	20		20		20		20		19	(1)	20	20	20	**159**	**(1)**
	10	20		20		19	(1)	20		19	(1)	20	20	20	**158**	**(1)**
	15	20		20		18	(2)	18	(1)	20		20	20	20	**156**	**(2)**
	20	20		20		18	(1)	18	(1)	20		20	20	20	**156**	**(1)**
	30	20		20		17	(2)	18	(5)	20		20	20	20	**155**	**(5)**
	50	20		20		17	(2)	14	(2)	20		20	20	20	**151**	**(2)**
Total		**140**		**140**		**129**	**(2)**	**128**	**(5)**	**138**	**(1)**	**140**	**140**	**140**	**1095**	**(5)**
2	2	20		20		20		20		20		20	20	20	**160**	
	5	20		20		20		20		20		20	20	20	**160**	
	10	20		20		20		19	(1)	19	(1)	20	20	20	**158**	**(1)**
	15	20		20		19	(1)	17	(1)	20		20	20	20	**156**	**(1)**
	20	20		19	(1)	19	(1)	20		20		20	20	20	**158**	**(1)**
	30	20		20		18	(1)	20		20		20	20	20	**158**	**(1)**
	50	20		20		19	(1)	17	(3)	20		20	20	20	**156**	**(3)**
Total		**140**		**139**	**(1)**	**135**	**(1)**	**133**	**(3)**	**139**	**(1)**	**140**	**140**	**140**	**1106**	**(3)**
3	2	20		20		20		20		20		20	20	20	**160**	
	5	16	(1)	14	(1)	18	(1)	20		20		20	20	20	**148**	**(1)**
	10	17	(1)	19	(1)	20		20		20		20	20	20	**156**	**(1)**
	15	11	(3)	19	(1)	20		20		20		20	20	20	**150**	**(3)**
	20	14	(3)	20		20		20		20		20	20	20	**154**	**(3)**
	30	8	(6)	19	(1)	20		20		20		20	20	20	**147**	**(6)**
	50	7	(8)	19	(1)	20		20		20		20	20	20	**146**	**(8)**
Total		**93**	**(8)**	**130**	**(1)**	**138**	**(1)**	**140**		**140**		**140**	**140**	**140**	**1061**	**(8)**
4	2	20		20		20		20		20		20	20	20	**160**	
	5	20		20		20		19	(1)	18	(1)	20	20	20	**157**	**(1)**
	10	20		20		19	(1)	15	(3)	20		20	20	20	**154**	**(3)**
	15	20		20		19	(1)	13	(1)	20		20	20	20	**152**	**(1)**
	20	20		18	(1)	19	(1)	16	(2)	20		20	20	20	**153**	**(2)**
	30	20		18	(1)	20		15	(4)	20		20	20	20	**153**	**(4)**
	50	20		20		20		11	(4)	20		20	20	20	**151**	**(4)**
Total		**140**		**136**	**(1)**	**137**	**(1)**	**109**	**(4)**	**138**	**(1)**	**140**	**140**	**140**	**1080**	**(4)**
5	2	20		20		20		20		20		20	20	20	**160**	
	5	20		20		20		20		17	(1)	20	20	20	**157**	**(1)**
	10	20		19	(1)	15	(4)	11	(1)	20		20	20	20	**145**	**(4)**
	15	20		18	(2)	15	(1)	14	(1)	19	(1)	20	20	20	**146**	**(2)**
	20	17	(1)	18	(3)	12	(2)	12	(1)	20		20	20	20	**139**	**(3)**
	30	19	(1)	16	(3)	8	(3)	15	(4)	20		20	20	20	**138**	**(4)**
	50	19	(1)	18	(5)	11	(4)	12	(3)	20		20	20	20	**140**	**(5)**
Total		**135**	**(1)**	**129**	**(5)**	**101**	**(4)**	**104**	**(4)**	**136**	**(1)**	**140**	**140**	**140**	**1025**	**(5)**
6	2	20		20		20		20		20		20	20	20	**160**	
	5	20		20		20		20		16	(1)	20	20	20	**156**	**(1)**
	10	20		20		20		19	(1)	18	(1)	20	20	20	**157**	**(1)**
	15	20		20		20		19	(1)	16	(1)	20	20	20	**155**	**(1)**
	20	20		20		19	(1)	17	(1)	18	(1)	20	20	20	**154**	**(1)**
	30	20		20		19	(1)	20		20		20	20	20	**159**	**(1)**
	50	20		20		20		17	(3)	20		20	20	20	**157**	**(3)**
Total		**140**		**140**		**138**	**(1)**	**132**	**(3)**	**128**	**(1)**	**140**	**140**	**140**	**1098**	**(3)**

Table 7.1: Number of instances solved at the root node (in brackets: maximum absolute gap between U and L if greater than 0)

As can also be seen from Table 7.1, the maximal absolute gap between U and L was 8 (see again triple $(50, 100, 3)$). However, Table 7.2 reveals that such a huge gap occurred very rarely. More precisely, only for 19 out of the 6720 instances we observed a gap that

was greater than 3. If the gap is greater than 0 at all, then, in most cases the gap is just 1. Table 7.2 gives a concise summary of the frequency distribution of the absolute gap between U and L at the root node. The overall ratio U/L averaged about 1.0012 (considering all 6720 instances) and about 1.0316 (considering only the 255 instances where $U > L$), respectively.

$U - L$	0	1	2	3	4	5	6	7	8	≥ 9
#	6465	181	26	29	8	5	1	2	3	0

Table 7.2: Frequency distribution of the gap between U and L at the root node

The overall performance of our branch-and-bound algorithm is assessed by the next performance criteria: average total computation time in seconds (see Table 7.3) and average number of generated branch-and-bound nodes (see Table 7.4). For each (m, C, d)-triple, averages are taken over all instances that were optimally solved within the given runtime of 300 seconds. Note that this introduces a bias in cases where not all 20 instances have been solved. Therefore, if greater than zero, we also provide the number of instances that remained unsolved (cf. numbers in brackets in Table 7.3). Complementing the other two criteria (time and nodes), the number of unsolved instances (US) is a meaningful indicator of the difficulty of finding an optimal solution.
Looking at the average computation times we see that, except for a few triples, optimal solutions are usually identified within a few milliseconds. Clearly, the short computation times are also a result of our fast and powerful bounding techniques (cf. also Table 7.1). Average times of more than 10 seconds occur only for the six triples $(30, 150, 5)$, $(15, 200, 5)$, $(20, 150, 5)$, $(20, 200, 1)$, $(15, 200, 4)$, and $(20, 150, 4)$. However, this does not automatically mean that these triples are the most difficult ones. Consider, e.g., the triples $(50, 100, 3)$ and $(30, 100, 3)$ where the average computation time is only 3 milliseconds each, but 13 and 12 instances remained unsolved, respectively.
Recalling that 255 out of 6720 instances required branching, our branch-and-bound algorithm was able to solve 127 of them within the given runtime. Broken down by the three parameters that characterize our instances, we recorded that small-sized instances with up to 10 bins are easy to solve via branching. Here, all 55 instances that required branching have been successfully solved by our branch-and-bound algorithm. When the number of bins increases, it is not surprising that instances become more difficult: while still 56 out of the 91 branched instances were optimally solved for $m \in \{15, 20\}$, only 16 out of the 109 branched instances for $m \in \{30, 50\}$ were optimally solved. Regarding the bin capacity C, the most difficult instances occurred when $C = 100$ and $C = 200$ where we found optimal solutions only for 15 out of the 52 and 37 out of the 94 branched instances, respectively. With respect to the third parameter d, our algorithm was able to solve 22 of the 39 "uniform" instances that remained unsolved at the root note, 39 of the 99 Gaussian-based instances, and 66 of the 117 Weibull-based instances.

Taking a brief look at the average number of generated branch-and-bound nodes (see Table 7.4), it is not surprising that the results are interrelated with the ones presented in the first two tables. While most of the solved instances required no branching, obvious peaks in the average number of generated nodes can be recognized for the seven triples $(30, 150, 5)$, $(20, 150, 5)$, $(20, 150, 4)$, $(30, 120, 5)$, $(15, 200, 5)$, $(15, 200, 4)$, and $(20, 200, 1)$

d	m	C														**Avg.**	**(US)**
		100		120		150		200		300		500		750	1000		
1	2	0.001		0.002		0.002		0.002		0.002		0.002		0.002	0.002	**0.002**	
	5	0.002		0.002		0.005		0.001		0.021		0.002		0.002	0.002	**0.005**	
	10	0.002		0.003		0.016		0.002		0.041		0.002		0.002	0.002	**0.009**	
	15	0.002		0.002		0.684		0.660		0.002		0.002		0.002	0.003	**0.170**	
	20	0.002		0.002		0.543		11.941	(1)	0.003		0.002		0.003	0.003	**1.497**	**(1)**
	30	0.002		0.001		6.754	(1)	0.003	(2)	0.002		0.003		0.003	0.003	**0.819**	**(3)**
	50	0.001		0.001		0.359	(3)	0.002	(6)	0.003		0.004		0.004	0.004	**0.043**	**(9)**
Avg.	**(US)**	**0.001**		**0.002**		**1.172**	**(4)**	**1.834**	**(9)**	**0.011**		**0.002**		**0.003**	**0.003**	0.364	(13)
2	2	0.001		0.001		0.002		0.001		0.002		0.002		0.002	0.002	**0.001**	
	5	0.002		0.002		0.003		0.001		0.001		0.002		0.002	0.002	**0.002**	
	10	0.001		0.002		0.012		0.018		3.337		0.002		0.002	0.002	**0.422**	
	15	0.001		0.005		0.016		0.725		0.003		0.002		0.002	0.002	**0.095**	
	20	0.001		0.006		0.290		0.002		0.002		0.002		0.002	0.003	**0.039**	
	30	0.001		0.001		0.442		0.001		0.003		0.002		0.003	0.003	**0.057**	
	50	0.001		0.001		0.157	(1)	0.002	(3)	0.004		0.004		0.004	0.004	**0.022**	**(4)**
Avg.	**(US)**	**0.001**		**0.003**		**0.132**	**(1)**	**0.109**	**(3)**	**0.479**		**0.002**		**0.002**	**0.003**	0.091	(4)
3	2	0.002		0.002		0.001		0.002		0.002		0.002		0.003	0.003	**0.002**	
	5	0.016		0.026		0.021		0.002		0.002		0.003		0.003	0.003	**0.010**	
	10	0.417		0.008		0.002		0.002		0.003		0.004		0.003	0.033	**0.055**	
	15	1.383	(6)	3.711		0.002		0.002		0.003		0.003		0.003	0.004	**0.610**	**(6)**
	20	0.002	(6)	0.000		0.002		0.003		0.004		0.003		0.005	0.004	**0.003**	**(6)**
	30	0.003	(12)	0.002	(1)	0.002		0.002		0.003		0.003		0.004	0.004	**0.003**	**(13)**
	50	0.003	(13)	0.002	(1)	0.003		0.002		0.003		0.004		0.005	0.006	**0.003**	**(14)**
Avg.	**(US)**	**0.273**	**(37)**	**0.544**	**(2)**	**0.005**		**0.002**		**0.003**		**0.003**		**0.004**	**0.004**	0.098	(39)
4	2	0.001		0.000		0.001		0.002		0.001		0.002		0.002	0.003	**0.001**	
	5	0.001		0.000		0.002		0.007		0.056		0.002		0.002	0.003	**0.009**	
	10	0.001		0.002		0.033		2.334		0.003		0.003		0.003	0.003	**0.298**	
	15	0.001		0.004		0.016		11.020	(4)	0.004		0.003		0.003	0.003	**1.135**	**(4)**
	20	0.001		0.020		10.862		4.956	(3)	0.003		0.004		0.004	0.003	**1.925**	**(3)**
	30	0.001		0.017		0.016		0.003	(5)	0.002		0.002		0.003	0.004	**0.006**	**(5)**
	50	0.002		0.012		0.059		0.003	(9)	0.004		0.003		0.004	0.005	**0.012**	**(9)**
Avg.	**(US)**	**0.001**		**0.008**		**1.570**		**2.584**	**(21)**	**0.011**		**0.003**		**0.003**	**0.003**	0.483	(21)
5	2	0.001		0.001		0.002		0.001		0.001		0.001		0.001	0.001	**0.001**	
	5	0.002		0.002		0.006		0.018		0.080		0.002		0.002	0.002	**0.014**	
	10	0.005		0.015		0.079		2.142		0.002		0.002		0.002	0.002	**0.281**	
	15	0.003		0.015		1.464		13.845	(1)	0.002	(1)	0.002		0.002	0.002	**1.853**	**(2)**
	20	0.057		0.043		13.616	(2)	5.801	(6)	0.000		0.002		0.002	0.002	**2.161**	**(8)**
	30	0.010		3.495		34.927	(10)	0.249	(5)	0.003		0.003		0.004	0.003	**2.920**	**(15)**
	50	0.024		0.037	(1)	0.034	(9)	0.003	(8)	0.003		0.004		0.004	0.004	**0.013**	**(18)**
Avg.	**(US)**	**0.015**		**0.519**	**(1)**	**5.258**	**(21)**	**3.261**	**(20)**	**0.013**	**(1)**	**0.002**		**0.002**	**0.002**	1.016	(43)
6	2	0.001		0.001		0.001		0.001		0.002		0.001		0.001	0.001	**0.001**	
	5	0.002		0.002		0.003		0.003		0.013		0.002		0.002	0.002	**0.003**	
	10	0.001		0.001		0.004		0.007		0.061		0.002		0.002	0.002	**0.010**	
	15	0.002		0.002		0.003		0.044		0.381	(2)	0.002		0.002	0.002	**0.050**	**(2)**
	20	0.002		0.002		0.004		6.437	(1)	0.002	(2)	0.002		0.002	0.002	**0.781**	**(3)**
	30	0.002		0.001		0.017		0.349		0.002		0.002		0.003	0.003	**0.048**	
	50	0.001		0.001		0.017		0.146	(3)	0.003		0.004		0.004	0.004	**0.020**	**(3)**
Avg.	**(US)**	**0.001**		**0.001**		**0.007**		**0.977**	**(4)**	**0.063**	**(4)**	**0.002**		**0.002**	**0.002**	0.129	(8)

Table 7.3: Average computation time in seconds (in brackets: number of unsolved instances if greater than 0)

where the averages are each about 10^6 or even greater. Once again, we want to emphasize that there are some other triples as well for which quite difficult instances exist (cf., e.g., the triples $(50, 100, 3)$ and $(30, 100, 3)$ where 7 and 8 instances were solved at the root node and the other 13 and 12 instances remained unsolved after the time limit, respectively).

To summarize the computational results reported in Tables 7.1–7.4, one of our major findings is that instances where the average number of items per bin ranges between 3 and about 5 are much more difficult to solve than instances where on average 10 or even more items are necessary to cover a bin. This observation is well in line with what is

known for related problems such as the classical one-dimensional bin packing problem, its dual version or even parallel machine scheduling problems where instances with about 3 items/jobs per bin/machine are considered to be particularly difficult (cf., e.g., Fleszar and Charalambous, 2011, Labbé et al., 1995, Haouari and Jemmali, 2008a).

To complement the data in Table 7.1 and to provide a clearer picture of the effect and usefulness of the developed reduction criteria, lower and upper bound procedures ("initial bounds", U_{SSH}, L_2/L_3, L_{BCP}, L_{CG}) as well as the implemented item-oriented depth-first branching scheme, for each of these components we also count the number of instances that were solved after their computation/execution. Table 7.5 contains the overall results (cf. row labeled as Total) and three blocks where the total results are broken down by the distribution of the item weights d, the number of bins m, and the bin capacity C. Before we discuss the main results it is important to note that the entries in Table 7.5 strongly depend on the order in which the individual components are applied (see beginning of Section 7.4.3 for the implemented order to which the order of the columns in Table 7.5 correspond from left to right).

As can be seen from Table 7.5, 185 instances have already been solved by application of our reduction criteria (cf. Section 7.2). Unsurprisingly, all but one of them belong to triples where $m = 2$. The results also indicate that the reduction criteria are more powerful for small-sized bins ($C \leq 150$) and when the average number of items per bin is about 2–3. Apart from the pure number of instances that have already been solved by application of the reduction criteria, we observed that the criteria helped to reduce the solution space for a total number of 2080 instances.

Continuing with the influence of the initial bounds – by which we subsume the application of the three construction heuristics FFD_r, BFD_r, and LFD and the computation of the trivial lower bounds L_0, L'_0, L_1, and L'_1 – we see that 2219 instances were solved after their computation. Many of them belong to triples where $m \in \{2, 5\}$ or $C \leq 200$ so that the initial bounds appear to be sufficiently strong to solve rather small-sized instances.

The application of our improvement heuristic SSH turned out to be very successful. With its help we were not only able to improve the upper bound value for 4155 out of the remaining 4316 instances, but also to verify – in conjunction with the trivial lower bounds – optimal solutions for 3385 of them. It is interesting to see that among the 3385 instances there are all 2092 instances with $C \geq 500$ that remained unsolved thus far. Thus, not only SSH works well on instances with large-sized bins but also our initial lower bounds.

After application of SSH only 931 instances remained unsolved. By means of our more elaborate lower bound procedures we were able to solve 676 of them. More precisely, 390 instances have been solved after computing the lower bounds L_2 and L_3, 41 instances via the bin covering based lower bound, and 245 via the computationally more expensive column generation lower bound. This justifies their application at the root node in addition to the trivial lower bounds. In particular, we believe the column generation bound to be very strong because we also observed that L_{CG} equals z^* for every solved instance that required branching, i.e. all these instances have the integer round-up property (IRUP). We refer to Baum and Trotter (1981) who introduced this notation and to Scheithauer and Terno (1995) who defined the modified integer round-up property (MIRUP). Nevertheless, due to the computational complexity of the column generation bound, it would not be reasonable to refrain from applying the simpler and faster combinatorial lower bounds in

d	m	C								**Avg.**
		100	120	150	200	300	500	750	1000	
1	2	1	1	1	1	1	1	1	1	**1**
	5	1	1	1	1	4	1	1	1	**1**
	10	1	1	25	1	2631	1	1	1	**333**
	15	1	1	125854	38611	1	1	1	1	**20559**
	20	1	1	63889	983713	1	1	1	1	**125588**
	30	1	1	814871	1	1	1	1	1	**98616**
	50	1	1	1	1	1	1	1	1	**1**
Avg.		**1**	**1**	**141750**	**148571**	**377**	**1**	**1**	**1**	**35045**
2	2	1	1	1	1	1	1	1	1	**1**
	5	1	1	1	1	1	1	1	1	**1**
	10	1	1	1	297	344816	1	1	1	**43140**
	15	1	1	253	61099	1	1	1	1	**7670**
	20	1	126	22942	1	1	1	1	1	**2884**
	30	1	1	38017	1	1	1	1	1	**4753**
	50	1	1	1	1	1	1	1	1	**1**
Avg.		**1**	**19**	**8808**	**8964**	**49260**	**1**	**1**	**1**	**8380**
3	2	1	1	1	1	1	1	1	1	**1**
	5	24	108	1889	1	1	1	1	1	**253**
	10	87362	1087	1	1	1	1	1	1	**11057**
	15	102107	419247	1	1	1	1	1	1	**63731**
	20	1	1	1	1	1	1	1	1	**1**
	30	1	1	1	1	1	1	1	1	**1**
	50	1	1	1	1	1	1	1	1	**1**
Avg.		**30847**	**60934**	**271**	**1**	**1**	**1**	**1**	**1**	**10754**
4	2	1	1	1	1	1	1	1	1	**1**
	5	1	1	1	2	138	1	1	1	**18**
	10	1	1	3915	533242	1	1	1	1	**67145**
	15	1	1	135	1108408	1	1	1	1	**113701**
	20	1	9475	1507045	319076	1	1	1	1	**227737**
	30	1	1727	1	1	1	1	1	1	**224**
	50	1	1	1	1	1	1	1	1	**1**
Avg.		**1**	**1601**	**215871**	**284233**	**21**	**1**	**1**	**1**	**58483**
5	2	1	1	1	1	1	1	1	1	**1**
	5	1	1	1	1	10933	1	1	1	**1368**
	10	1	3	3096	242612	1	1	1	1	**30714**
	15	1	572	129130	1180434	1	1	1	1	**158370**
	20	8788	3589	2312674	617179	1	1	1	1	**332344**
	30	5	1307884	7807694	1	1	1	1	1	**718861**
	50	1194	1623	1	1	1	1	1	1	**386**
Avg.		**1427**	**189006**	**1028148**	**299342**	**1574**	**1**	**1**	**1**	**171738**
6	2	1	1	1	1	1	1	1	1	**1**
	5	1	1	1	1	5	1	1	1	**1**
	10	1	1	1	37	2228	1	1	1	**284**
	15	1	1	1	1720	19686	1	1	1	**2461**
	20	1	1	37	498940	1	1	1	1	**60387**
	30	1	1	301	1	1	1	1	1	**39**
	50	1	1	1	1	1	1	1	1	**1**
Avg.		**1**	**1**	**49**	**69964**	**2934**	**1**	**1**	**1**	**8922**

Table 7.4: Average number of generated branch-and-bound nodes

		Reduction Criteria	Initial Bounds	U_{SSH}	L_2/L_3	L_{BCP}	L_{CG}	B&B solved	B&B unsolved
Total		**185 (2.8%)**	**2219 (33.0%)**	**3385 (50.4%)**	**390 (5.8%)**	**41 (0.6%)**	**245 (3.6%)**	**127 (1.9%)**	**128 (1.9%)**
d	1	28	474	493	28	16	56	12	13
	2	44	496	475	31	11	49	10	4
	3	0	191	858	7	2	3	20	39
	4	30	368	538	105	5	34	19	21
	5	25	256	550	118	5	71	52	43
	6	58	434	471	101	2	32	14	8
m	2	184	622	129	19	5	1	0	0
	5	1	418	437	49	14	18	23	0
	10	0	285	537	61	3	42	32	0
	15	0	248	551	64	2	50	31	14
	20	0	224	574	64	6	46	25	21
	30	0	215	578	70	9	38	14	36
	50	0	207	579	63	2	50	2	57
C	100	81	605	61	29	1	11	15	37
	120	70	527	102	74	2	39	23	3
	150	30	212	150	206	22	158	36	26
	200	4	307	304	80	15	36	37	57
	300	0	140	676	1	1	1	16	5
	500	0	130	710	0	0	0	0	0
	750	0	147	693	0	0	0	0	0
	1000	0	151	689	0	0	0	0	0

Table 7.5: Number of instances solved at the root node (by reduction criteria or lower/upper bound procedures) or by branch-and-bound and number of unsolved instances

advance.

Lastly, we examined the performance of a commercial solver when applied to the MCBCP as formulated by (7.1)–(7.4). We decided for Gurobi to serve as the benchmark for our algorithm since Gurobi is the state-of-the-art solver in mathematical programming. In this last part of our computational study, we compare the performance of Gurobi and our algorithm (labeled as B&B) with respect to the two criteria number of solved instances and average computation time. The corresponding results are presented in a compact way in Table 7.6 that consists of three horizontal blocks – one for each of our three parameters d, m, and C. We remark that, for each of the two solution approaches, we averaged the computation time only over those instances that have been solved before reaching the time limit of 300 seconds. Moreover, Gurobi was allowed to use all four cores in parallel whereas our algorithm ran as a single thread. The results in Table 7.6 clearly indicate that our algorithm is superior to the commercial solver Gurobi for all investigated parameter settings. Considering the number of unsolved instances, Gurobi was not able to solve a total number of 1494 instances within the given time limit whereas only 128 instances remained unsolved after application of our exact branch-and-bound algorithm. Again, the number of bins m has a significant influence on the performance of the solver. Both, the average computation time as well as the number of unsolved instances strictly increases when m increases. In particular, instances with more than 10 bins already appear to be very hard to solve for Gurobi. Regarding the bin capacity C, we observe obvious peaks

in the number of unsolved instances when $C \in \{200, 300, 500\}$ while the distribution d of the item weights seems to have only a mild influence on Gurobi's performance. All in all, Gurobi often required more than 10 seconds on average to solve an instance while our branch-and-bound algorithm never required more than about 1.4 seconds on average.

	d	1	2	3	4	5	6		
Gurobi	Time	12.999	9.222	14.984	15.768	19.103	8.981		
	US	270	197	243	261	279	244		
B&B	Time	0.364	0.091	0.098	0.483	1.016	0.129		
	US	13	4	39	21	43	8		
	m	2	5	10	15	20	30	50	
Gurobi	Time	0.003	0.395	12.940	14.584	16.711	27.465	47.006	
	US	0	0	68	213	307	386	520	
B&B	Time	0.002	0.007	0.179	0.649	1.063	0.617	0.019	
	US	0	0	0	14	21	36	57	
	C	100	120	150	200	300	500	750	1000
Gurobi	Time	3.816	6.628	13.988	15.502	18.410	15.011	19.992	18.273
	US	60	80	164	224	382	283	161	140
B&B	Time	0.039	0.178	1.259	1.388	0.097	0.003	0.003	0.003
	US	37	3	26	57	5	0	0	0

Table 7.6: Comparison between Gurobi and our B&B-algorithm

7.6 Conclusions

The Chapter introduced the minimum cardinality bin covering problem (MCBCP) which is a variant of the bin covering problem (BCP) and can be seen as the dual of the maximum cardinality bin packing problem. We introduced several combinatorial lower bounds, analyzed their worst-case performance, and derived enhanced bounding techniques. Furthermore, we proposed three construction heuristics, a powerful subset sum-based improvement heuristic as well as an exact branch-and-bound procedure. The computational performance of our approach on a large set of 6720 randomly generated instances is convincing. On the one hand, due to our powerful bounding techniques, we are able to solve the vast majority of instances already at the root node. On the other hand, if branching is required, then our exact algorithm often identifies an optimal solution within a reasonable amount of time when the number of bins is not too large.

With regards to future research on MCBCP we identify the following interesting topics. In order to quickly find high-quality or even optimal solutions for instances with many small-sized bins and, on average, rather large items in relation to the bin capacity, fast and efficient meta heuristics are required and the implementation of a bin-oriented branching scheme might be beneficial as well. Furthermore, at least from a theoretical point of view, the development of combinatorial lower bounds with a provable worst-case performance greater than 0 and the analysis of the worst-case performance of the construction heuristics presented in Section 7.4.1 seem to be challenging tasks. Concerning the pattern-based

formulation of the MCBCP and the column generation lower bound we see three particular research directions: a continuing study on the IRUP/MIRUP conjecture, the development of an enhanced and accelerated column generation procedure, e.g. based on dual cuts (cf. Valério de Carvalho, 2005), and the transformation of a fractional RMP-solution into a feasible integer solution, e.g. via rounding and repair heuristics. Eventually, similar as Friesen and Langston (1987) did for the bin packing problem, studying other variants of the BCP with respect to the optimal number of items packed seems to be also interesting and might lead to further insights.

Chapter 8

Conclusion of the thesis

In the present doctoral thesis an overview on short-term planning tasks for workload distribution, that are related to machine scheduling, multiprocessor scheduling and bin-realted problems was given. To be more precise, we analyzed several variants of the identical parallel machine scheduling problem as well as a specific bin covering problem from a rather technical point of view. We described the importance of researches on these topics by identifying research gaps and by showing the practical relevance of such problem variants, which received only little attention, yet. We introduced the minimum cardinality bin covering problem which is a "dual" of $P||C_{min}$ and demonstrated that it has significant practical relevance as it can be transformed into the liquid loading problem (cf. Christofides et al., 1979). The considered problems cover a large share of short-time planning tasks as they can be used to measure the efficiency as well as the balance of a system.

In order to give a basic overview, first a detailed problem description as well as a comprehensive review on the existing literature is given for each of the considered optimization problems. Based on this, we are able to describe the necessity for further researches with regards to issues such as structural patterns, bounding and solution approaches.

The main contribution of the thesis is the so called "path-concept", which enables us to characterize potentially (uniquely) optimal schedules, e.g. for $P||C_{max}$ and $P||C_{min}$ (cf. Chapter 3 and 4) but also for other similarly structured discrete optimization problems. Based on a concept similar to inverse optimization, we identified a subset of all possible m^n schedules with certain properties, so that they have the potential to become (uniquely) optimal independently from the actual processing times of the jobs. Based on this knowledge, a shrunken solution space and therefore a reduced complexity can be achieved. As the modern hardware technologies are really beneficial for the implementation of complex structured ideas and for the handling of an increasing amount of information, we decided to transform the concept into several dominance criteria for depth-first branch-and-bound algorithms and intensively analyzed their performance within Chapters 3 and 4. The approach turned out to perform very well, especially when the ratio of n to m is rather small.

For future research on structural patterns we see a lot of potential, as the "path-concept" might also have an impact on many other similarly structured problems and has still some potential for extensions by incorporating more problem specific data. Although

the concept in its basic form is designed for depth-first enumeration trees, where jobs are assigned in non-increasing order with respect to their processing times, it is also applicable to other search strategies.

With respect to the analysis of structural patterns, also several preprocessing and lifting methods were proposed (cf. Chapter 6) which are capable of reducing the solution space, e.g. by removing jobs as well as machines without changing the optimal solution. It might be a fruitful approach to combine some of these methods as well as to apply them to the "dual" problem $P||C_{min}$.

Besides the analysis of solution patterns and the implementation of state-of-the-art branch-and-bound procedures, the definition of novel strategies for the determination of bounds as well as heuristics (cf. Chapter 4, 6 and 7) is another core element of the thesis. The proposed bounding strategies are based on different methodological approaches like column generation, the duality to other problems or theoretical characteristics of the respective problems. For the design of constructive as well as improvement heuristics we also decided for a diversified proceeding, by adapting existing and by defining new methods, e.g. based on dynamic programming. The impact of both, lower and upper bounds, is shown by the results of the respective computational studies.

To allow for a comprehensive analysis of practically relevant instances, we also defined benchmark data-sets for k_i-partitioning and MCBCP which might be a prerequisite for further researches on these topics. Finally, the thesis gave an overview on the coherence between machine scheduling problems and bin-covering as well as bin-packing (cf. Chapter 2 – 7). As it turned out, a sophisticated exploitation of "dualities" leads to tight lower and upper bounds.

In general we can conclude, that despite its simple problem inherent structure, the analysis of the identical parallel machine scheduling problem is still well justified. Although the scope of research projects changed within the last decades towards the definition of meta-heuristics (cf. Chapter 2), we think that the knowledge on structural patterns is still very limited and a profound analysis of solution properties might therefore be an essential prerequisite for the determination of novel, strong solution approaches.

Appendices

Appendix A

Appendix for Effective solution space limitation for the identical parallel machine scheduling problem

A.1 Proof of Lemma 3.2.6

At first, recall that with the two-machine path-modification of $P_S^{(i_1,i_3)}$ upward moves can only be shifted backwards. In regard of $P_S^{(i_1,i_2)}$, swapping the jobs k on i_1 and l on i_3 implies that:

$$P_{\tilde{S}}^{(i_1,i_2)}(j) = \begin{cases} P_S^{(i_1,i_2)}(j), & \text{if } j = 1,\ldots,k-1, \\ P_S^{(i_1,i_2)}(j)-1, & \text{if } j = k,\ldots,l-1, \\ P_S^{(i_1,i_2)}(j), & \text{if } j = l,\ldots,n. \end{cases}$$

As this interrelation holds for each single swap during the (i_1, i_3)-path-modification, it follows immediately that the fulfillment of the path-condition of (i_1, i_2) cannot be violated. Indeed, the corresponding (i_1, i_2)-path of the resulting schedule may now take negative values even earlier. ■

A.2 Proof of Lemma 3.2.7

Assume that the number of jobs assigned to the machines i_1, i_2 and i_3 are r, s and q, respectively. Furthermore, let a_r, b_s and c_q denote the corresponding indices of the last job assigned to each of the three machines. Additionally, by b_k we denote the job with the k-th smallest index on machine i_2 in S. Then, we distinguish two cases for q.

1. $q = 1$.

   Due to the assumptions on the three considered paths, in this case we have that $r = 1$ and $s > 1$. According to Sect. 3.2.2.1 this implies that we need one shift and at most one swap during the path-modification of (i_2, i_3). Clearly, as the number of jobs assigned to machine i_2 is still at least one and the number of jobs assigned to i_3 is two after the modification of the (i_2, i_3)-path, both the resulting (i_1, i_2)-path and the (i_1, i_3)-path fulfill their path-condition.

2. $q > 1$.

   Clearly, this implies $s \geq q$ since $P_S^{(i_2,i_3)}(j) \geq 0$ for all $j = 1, \ldots, n$. Let j_1 denote the first position where $P_S^{(i_1,i_2)}$ takes a negative value. Analogously, define j_2 as the first position where $P_S^{(i_1,i_3)}$ takes a negative value. It is readily verified that $j_1 \leq b_q$. Furthermore, we can conclude $j_1 < j_2$ since $P_S^{(i_1,i_2)}(j) \leq P_S^{(i_1,i_3)}(j)$ for all positions $j = 1, \ldots, n$. Recall from Sect. 3.2.2.1 that the path-modification of $P_S^{(i_2,i_3)}$ will result in a schedule $\bar{S}$ where the $q-1$ longest jobs on machine i_2, i.e., $b_1, \ldots, b_{q-1}$, are the same as in schedule S. Moreover, note that job c_q is then assigned to machine i_2. Then, two subcases depending on the relation between j_1 and b_q can occur.

   (a) $j_1 \leq b_{q-1}$.

   This case is trivial because $b_1, \ldots, b_{q-1}$ remain the $q-1$ longest jobs on i_2. Thus, we have that $P_{\bar{S}}^{(i_1,i_2)}(j) = P_S^{(i_1,i_2)}(j)$ for $j \leq b_{q-1}$ and, in particular, $P_{\bar{S}}^{(i_1,i_2)}(j_1) = -1$.

   (b) $j_1 = b_q$.

   First, note that this subcase implies $j_2 = c_q$ and we can further conclude that exactly $q-1$ jobs on machine i_1 in S have an index no greater than $c_q - 1$. As the path-modification of $P_S^{(i_2,i_3)}$ swaps at least the jobs b_q and c_q while not affecting the assignment of the jobs $b_1, \ldots, b_{q-1}$, the indices of at least q jobs on machine i_2 in $\bar{S}$ are not greater than c_q. Thus, we have that $P_{\bar{S}}^{(i_1,i_2)}(c_q) \leq P_S^{(i_1,i_3)}(c_q) = -1$.

   Finally, for both subcases it is readily verified that $P_{\bar{S}}^{(i_1,i_3)}(j) \leq P_S^{(i_1,i_3)}(j)$ for all $j = 1, \ldots, n$ as (i) jobs on machine i_1 were not affected and (ii) some downward moves occur earlier than before. ■

A.3 Compatibility issues of dominance criteria

Criterion A.3.1
If job $k+1$ is the next one to be assigned and there are l machines with equal current completion time $C_{i_1}^k = \ldots = C_{i_l}^k$, then, only the machine with smallest index has to be considered for the assignment of $k+1$.

As long as the machines are empty, i.e., $C_{i_1}^k = \ldots = C_{i_l}^k = 0$, the criterion is in line with the non-permuted schedule representation (cf. Sect. 3.2.1) and can be applied without any modification. However, in case that $C_{i_1}^k = \ldots = C_{i_l}^k > 0$ an adaptation is required. Assume $i_1 < \ldots < i_l$ and let $L = (i_l, i_{l-1}, \ldots, i_1)$ be a sorted list of the respective machine-indices. Then, we distinguish two sub-cases. In case that all path-conditions are already fulfilled, it suffices to assign job $k+1$ to the first machine in L. In the other case, we successively and separately assign job $k+1$ to each of the machines in L (starting from left to right) as long as at least one node in the corresponding sub-tree is fathomed by the path-related dominance criteria.

Criterion A.3.2
If two consecutive jobs $k+1$ and $k+2$ have the same processing time, and job $k+1$ is assigned to M_i, only machines j fulfilling $C_j^k \geq C_i^k$ have to be considered for the assignment of job $k+2$.

This criterion is compatible in its original version with the path-related criteria as will be shown in the following. Assume (i) the next h jobs $k+1, k+2, \ldots, k+h$ to have equal processing times and (ii) $h_i \in \{0, 1, \ldots, h\}$ of them will be assigned to machine i ($i = 1, \ldots, m$). Note that h_i only denotes the number of jobs but not which jobs are exactly meant. Then, the idea is to show that for any fixed m-tuple $(h_1, \ldots, h_m)$ the respective assignment of these h jobs to the m machines according to Crit. A.3.2 induces a maximal number of machine-pairs that fulfill their path-condition compared to any feasible (partial) schedule that can be obtained by assigning h_i jobs out of $k+1, \ldots, k+h$ to M_i ($i = 1, \ldots, m$). In other words, if a pair $P_S^{(i_1,i_2)}$ does currently not fulfill its path-condition by application of Crit. A.3.2 after the assignment of the first $k+h$ jobs, the path-condition would not be fulfilled by any other schedule with respect to $(h_1, \ldots, h_m)$.
So, let i_1 and i_2 ($i_1 < i_2$) be two arbitrary machines for which the respective path-condition is currently not fulfilled for sure and where $h_{i_1}, h_{i_2} > 0$ (all other cases are trivial). Then, we distinguish the following two main cases:

1. $C_{i_1}^k = C_{i_2}^k$: If the machines are still empty, Crit. A.3.1 is applied in its original version. Otherwise, i_1 does not receive a job as long as the path-condition is not fulfilled.

2. $C_{i_1}^k > C_{i_2}^k$: We distinguish two sub-cases depending on h_{i_2}:

   (a) $h_{i_2} > P_S^{(i_1,i_2)}(k)$: According to Crit. A.3.2, machine i_1 cannot receive one of its h_{i_1} jobs until the current completion time of i_2 is at least as large as the completion time of i_1 or all h_{i_2} jobs are assigned to i_2. In either case, the corresponding path-condition will be fulfilled for sure.

   (b) $h_{i_2} \leq P_S^{(i_1,i_2)}(k)$: Here, the fulfillment of the respective path-condition cannot be achieved after the assignment of the first $k+h$ jobs independently from Crit. A.3.2.

Obviously, for any fixed m-tuple $(h_1, \ldots, h_m)$ it suffices to separately consider the pairs (i_1, i_2) since any job-assignment to another machine $i \notin \{i_1, i_2\}$ results in horizontal moves of the (i_1, i_2)-path and thus the previous argumentation remains valid.

Criterion A.3.3
If at level $k+1$ the number of remaining jobs $n-k$ is less than m, only the $n-k$ machines with smallest current completion times C_i^k have to be considered for the assignment of job $k+1$.

This criterion requires just a slight modification. Instead of considering only the $n-k$ machines with smallest current completion times, we also consider all machines with equal or longer running completion time that do currently not fulfill all their path-conditions for sure.

Criterion A.3.4
At level $n-2$, only two possible assignments have to be considered in order to optimally complete the partial schedule:

(i) At each level, the jobs $n-2$, $n-1$, and n are assigned to the machine with smallest current completion time.

(ii) At level $n-2$, job $n-2$ is assigned to the machine with second smallest current completion time, and the jobs $n-1$ and n are assigned to the machine with smallest completion time at level $n-2$.

As it is certainly not meaningful to fathom an almost complete partial schedule, we apply Crit. A.3.4 in its original version and neglect at this point the demand for solutions that fulfill all path-conditions.

Appendix B

Appendix for Improved approaches to the exact solution of the machine covering problem

Tables B.1 and B.2 contain detailed results for each of the 390 uniform and 390 non-uniform benchmark instances, respectively. Each row corresponds to a triple $(n, m, Interval)$ and each column to one of the 10 instances per triple. For each instance we record the best found objective function value (labeled as "Best") as well as the best found upper bound value (labeled as "UB"). Bold entries indicate optimal values.

				No.									
n	m	Interval		0	1	2	3	4	5	6	7	8	9
10	5	$[1, 10^2]$	Best	**87**	**75**	**101**	**100**	**68**	**95**	**93**	**89**	**100**	**85**
			UB	92	76	101	100	68	95	93	90	100	85
		$[1, 10^3]$	Best	**1149**	**898**	**844**	**1046**	**631**	**805**	**545**	**1066**	**1112**	**939**
			UB	1149	957	846	1046	631	805	545	1066	1112	939
		$[1, 10^4]$	Best	**11493**	**10634**	**8813**	**6763**	**9454**	**8970**	**9943**	**10016**	**9968**	**10223**
			UB	11493	10634	8813	6823	9454	9806	9943	10016	10878	10948
50	5	$[1, 10^2]$	Best	**514**	**559**	**451**	**571**	**539**	**496**	**540**	**535**	**471**	**497**
			UB	514	559	451	571	539	496	540	535	471	497
		$[1, 10^3]$	Best	**4950**	**5810**	**4624**	**4854**	**4409**	**5072**	**5148**	**5007**	**5485**	**5334**
			UB	4950	5810	4624	4854	4409	5072	5148	5007	5485	5334
		$[1, 10^4]$	Best	**55927**	**53504**	**50056**	**45688**	**55528**	**51621**	**45106**	**41226**	**51986**	**42169**
			UB	55927	53504	50056	45688	55528	51621	45106	41226	51986	42169
	10	$[1, 10^2]$	Best	**241**	**226**	**209**	**271**	**251**	**276**	**256**	**229**	**234**	**234**
			UB	241	226	209	271	251	276	256	229	234	234
		$[1, 10^3]$	Best	2546	2308	2497	**2176**	**2612**	2403	**2417**	**2795**	2283	**2456**
			UB	2547	2309	2498	2176	2612	2404	2417	2795	2284	2456
		$[1, 10^4]$	Best	26660	26227	23667	27758	27202	25289	23383	25476	32262	26701
			UB	26661	26233	23673	27764	27204	25295	23387	25478	32265	26706
	25	$[1, 10^2]$	Best	**97**	**96**	**95**	**99**	**99**	**100**	**94**	**86**	**93**	**100**
			UB	102	97	99	99	99	110	94	86	94	107
		$[1, 10^3]$	Best	863	**997**	**953**	**989**	**951**	**907**	921	**940**	**942**	908
			UB	899	1073	977	999	1107	929	938	1059	950	936
		$[1, 10^4]$	Best	**10355**	9013	7807	7390	**9311**	**9802**	8267	**9856**	**8819**	9079
			UB	11004	9306	7884	7483	9572	10789	8537	10674	9361	9298
100	5	$[1, 10^2]$	Best	**921**	**951**	**972**	**1050**	**1008**	**1082**	**1044**	**976**	**1063**	**1004**
			UB	921	951	972	1050	1008	1082	1044	976	1063	1004
		$[1, 10^3]$	Best	**10472**	**10858**	**10028**	**10236**	**9070**	**9234**	**10537**	**10133**	**10270**	**9001**
			UB	10472	10858	10028	10236	9070	9234	10537	10133	10270	9001
		$[1, 10^4]$	Best	**105081**	**91044**	**91927**	**96921**	**91927**	**96925**	**98933**	**106194**	**110073**	**105081**
			UB	105081	91044	91927	96921	91927	96925	98933	106194	110073	105081
	10	$[1, 10^2]$	Best	**545**	**441**	**507**	**492**	**517**	**529**	**515**	**514**	**493**	**460**
			UB	545	441	507	492	517	529	515	514	493	460
		$[1, 10^3]$	Best	**5335**	**5416**	**5011**	**5005**	**5183**	**4722**	**4984**	**5034**	**5503**	**5016**
			UB	5335	5416	5011	5005	5183	4722	4984	5034	5503	5016
		$[1, 10^4]$	Best	**46655**	**46158**	**52285**	**51413**	**52540**	**56580**	**45522**	**52421**	**48460**	**53151**
			UB	46655	46158	52285	51413	52540	56580	45522	52421	48460	53151
	25	$[1, 10^2]$	Best	**194**	**194**	**198**	**199**	**190**	**218**	**193**	**213**	**204**	**202**
			UB	194	194	198	199	190	218	193	213	204	202
		$[1, 10^3]$	Best	2089	1934	1940	1950	1790	2125	2192	1877	1894	2195
			UB	2091	1935	1941	1952	1792	2127	2193	1878	1895	2197
		$[1, 10^4]$	Best	21161	17181	21561	20830	20557	20682	20014	19260	20591	19111
			UB	21168	17196	21571	20841	20567	20694	20020	19271	20597	19123
500	5	$[1, 10^2]$	Best	**5106**	**4970**	**5085**	**4887**	**4925**	**5220**	**5152**	**4840**	**4917**	**5004**
			UB	5106	4970	5085	4887	4925	5220	5152	4840	4917	5004
		$[1, 10^3]$	Best	**49375**	**49881**	**49375**	**52902**	**51401**	**52852**	**49558**	**50613**	**47180**	**48867**
			UB	49375	49881	49375	52902	51401	52852	49558	50613	47180	48867
		$[1, 10^4]$	Best	**504251**	**500430**	**513001**	**485975**	**501999**	**492923**	**508864**	**488853**	**494248**	**505127**
			UB	504251	500430	513001	485975	501999	492923	508864	488853	494248	505127
	10	$[1, 10^2]$	Best	**2519**	**2450**	**2537**	**2392**	**2504**	**2541**	**2533**	**2512**	**2568**	**2543**
			UB	2519	2450	2537	2392	2504	2541	2533	2512	2568	2543
		$[1, 10^3]$	Best	**24921**	**24640**	**24255**	**24218**	**25308**	**25086**	**24206**	**25992**	**25876**	**24875**
			UB	24921	24640	24255	24218	25308	25086	24206	25992	25876	24875
		$[1, 10^4]$	Best	**265229**	**263643**	**256023**	**244038**	**249160**	**261390**	**239883**	**257333**	**241659**	**254780**
			UB	265229	263643	256023	244038	249160	261390	239883	257333	241659	254780
	25	$[1, 10^2]$	Best	**1015**	**950**	**997**	**994**	**1012**	**974**	**990**	**999**	**952**	**987**
			UB	1015	950	997	994	1012	974	990	999	952	987
		$[1, 10^3]$	Best	**9628**	**9924**	**10107**	**9988**	**10224**	**10474**	**10227**	**9729**	**10128**	**9683**
			UB	9628	9924	10107	9988	10224	10474	10227	9729	10128	9683
		$[1, 10^4]$	Best	**99712**	**101630**	**96835**	**96084**	**96221**	**102001**	**102493**	**100419**	**99712**	**97049**
			UB	99712	101630	96835	96084	96221	102001	102493	100419	99712	97049
1000	5	$[1, 10^2]$	Best	**9989**	**10262**	**10072**	**10105**	**9903**	**9782**	**10048**	**9860**	**10064**	**9826**
			UB	9989	10262	10072	10105	9903	9782	10048	9860	10064	9826
		$[1, 10^3]$	Best	**102391**	**98203**	**98085**	**101873**	**99749**	**100072**	**98725**	**99387**	**101817**	**99329**
			UB	102391	98203	98085	101873	99749	100072	98725	99387	101817	99329
		$[1, 10^4]$	Best	**1001418**	**993453**	**1012868**	**1011507**	**988262**	**968878**	**996506**	**1006553**	**999716**	**1012238**
			UB	1001418	993453	1012868	1011507	988262	968878	996506	1006553	999716	1012238
	10	$[1, 10^2]$	Best	**4836**	**5020**	**5109**	**4925**	**5118**	**4884**	**4954**	**5152**	**4854**	**4885**
			UB	4836	5020	5109	4925	5118	4884	4954	5152	4854	4885
		$[1, 10^3]$	Best	**50317**	**48373**	**48582**	**50546**	**50555**	**49107**	**49300**	**50101**	**50232**	**49954**
			UB	50317	48373	48582	50546	50555	49107	49300	50101	50232	49954
		$[1, 10^4]$	Best	**495462**	**482122**	**494773**	**502895**	**483590**	**488311**	**492996**	**511505**	**506910**	**512238**
			UB	495462	482122	494773	502895	483590	488311	492996	511505	506910	512238
	25	$[1, 10^2]$	Best	**2024**	**1930**	**2026**	**2023**	**2007**	**1958**	**2032**	**1965**	**2024**	**2047**
			UB	2024	1930	2026	2023	2007	1958	2032	1965	2024	2047
		$[1, 10^3]$	Best	**20305**	**20100**	**19821**	**20462**	**19360**	**20021**	**20556**	**20319**	**20122**	**19119**
			UB	20305	20100	19821	20462	19360	20021	20556	20319	20122	19119
		$[1, 10^4]$	Best	**202497**	**198820**	**205820**	**202422**	**197545**	**203788**	**195468**	**201132**	**202464**	**200507**
			UB	202497	198820	205820	202422	197545	203788	195468	201132	202464	200507

Table B.1: Detailed results on all 390 uniform instances

				No.									
n	m	Interval		0	1	2	3	4	5	6	7	8	9
10	5	$[1, 10^2]$	Best	**100**	**115**	**103**	**104**	**110**	**113**	**102**	**119**	**113**	**107**
			UB	100	115	103	104	110	113	102	119	113	107
		$[1, 10^3]$	Best	**992**	**1135**	**1030**	**1035**	**1093**	**1121**	**1006**	**1187**	**1124**	**1066**
			UB	992	1135	1030	1035	1093	1121	1006	1187	1124	1066
		$[1, 10^4]$	Best	**9913**	**11335**	**10290**	**10351**	**10921**	**11204**	**10055**	**11862**	**11230**	**10653**
			UB	9913	11335	10290	10351	10921	11204	10055	11862	11230	10653
50	5	$[1, 10^2]$	Best	902	912	905	893	898	901	906	901	909	914
			UB	935	931	931	927	925	932	935	931	941	931
		$[1, 10^3]$	Best	8984	9068	9004	8901	8942	8987	9032	8956	9051	9100
			UB	9348	9307	9316	9277	9265	9328	9348	9312	9406	9321
		$[1, 10^4]$	Best	89811	90647	90001	88978	89389	89836	90290	89511	90473	90953
			UB	93483	93084	93162	92781	92647	93283	93477	93118	94053	93214
	10	$[1, 10^2]$	Best	398	413	410	412	419	405	414	410	410	402
			UB	466	465	468	468	465	463	469	465	464	465
		$[1, 10^3]$	Best	3969	4103	4084	4102	4181	4034	4120	4084	4082	4005
			UB	4663	4660	4677	4678	4654	4638	4691	4654	4649	4660
		$[1, 10^4]$	Best	39661	41002	40811	40991	41800	40314	41176	40820	40796	40026
			UB	46626	46602	46776	46777	46548	46382	46907	46547	46495	46604
	25	$[1, 10^2]$	Best	**104**	**117**	**105**	**108**	**101**	**103**	**109**	**113**	**118**	**119**
			UB	104	117	105	108	101	103	109	113	118	119
		$[1, 10^3]$	Best	**1039**	**1162**	**1044**	**1076**	**1006**	**1028**	**1085**	**1125**	**1176**	**1183**
			UB	1039	1162	1044	1076	1006	1028	1085	1125	1176	1183
		$[1, 10^4]$	Best	**10378**	**11614**	**10432**	**10751**	**10054**	**10266**	**10843**	**11244**	**11750**	**11825**
			UB	10378	11614	10432	10751	10054	10266	10843	11244	11750	11825
100	5	$[1, 10^2]$	Best	**1873**	**1861**	**1863**	**1864**	**1873**	**1870**	**1862**	**1869**	**1867**	**1866**
			UB	1873	1861	1863	1864	1873	1870	1862	1869	1867	1866
		$[1, 10^3]$	Best	**18718**	**18616**	**18641**	**18654**	18727	**18700**	**18625**	**18690**	**18667**	**18656**
			UB	18718	18616	18641	18654	18728	18700	18625	18690	18667	18656
		$[1, 10^4]$	Best	**187165**	**186177**	**186406**	**186542**	**187273**	**187000**	**186260**	**186915**	**186668**	**186557**
			UB	187165	186177	186406	186542	187273	187000	186260	186915	186668	186557
	10	$[1, 10^2]$	Best	911	898	908	904	910	901	898	905	896	895
			UB	934	933	936	930	934	933	939	936	930	933
		$[1, 10^3]$	Best	9081	8959	9039	9010	9078	8963	8955	9016	8937	8925
			UB	9339	9338	9356	9304	9344	9329	9386	9360	9301	9333
		$[1, 10^4]$	Best	90767	89548	90354	90068	90754	89582	89513	90114	89331	89213
			UB	93390	93386	93568	93040	93441	93293	93860	93596	93016	93328
	25	$[1, 10^2]$	Best	304	**312**	303	312	305	318	**314**	**306**	**313**	301
			UB	373	312	370	374	371	370	314	306	313	370
		$[1, 10^3]$	Best	3030	3111	3022	3110	3042	3169	3136	3047	3127	3002
			UB	3730	3734	3706	3739	3721	3703	3741	3734	3748	3711
		$[1, 10^4]$	Best	30288	31091	30209	31086	30410	31671	31345	30454	31260	29995
			UB	37310	37338	37071	37398	37216	37043	37416	37343	37481	37116
500	5	$[1, 10^2]$	Best	**9406**	**9419**	**9398**	**9400**	**9380**	**9407**	**9380**	**9395**	**9371**	**9391**
			UB	9406	9419	9398	9400	9380	9407	9380	9395	9371	9391
		$[1, 10^3]$	Best	**94044**	**94178**	**93985**	**94003**	**93820**	**94064**	**93822**	**93937**	**93761**	**93884**
			UB	94044	94178	93985	94003	93820	94064	93822	93937	93761	93884
		$[1, 10^4]$	Best	**940444**	**941762**	**939851**	**940029**	**938230**	**940641**	**938208**	**939385**	**937623**	**938833**
			UB	940444	941762	939851	940029	938230	940641	938208	939385	937623	938833
	10	$[1, 10^2]$	Best	**4702**	**4698**	**4685**	**4700**	**4695**	**4705**	**4703**	**4706**	**4694**	**4705**
			UB	4702	4698	4685	4700	4695	4705	4703	4706	4694	4705
		$[1, 10^3]$	Best	**47012**	**46979**	**46863**	**46998**	**46953**	**47061**	**47022**	**47055**	**46943**	**47060**
			UB	47012	46979	46863	46998	46953	47061	47022	47055	46943	47060
		$[1, 10^4]$	Best	**470117**	**469780**	**468636**	**469993**	**469537**	**470608**	**470225**	**470546**	**469430**	**470592**
			UB	470117	469780	468636	469993	469537	470608	470225	470546	469430	470592
	25	$[1, 10^2]$	Best	**1877**	**1877**	**1875**	**1878**	**1881**	**1881**	**1880**	**1883**	**1883**	**1878**
			UB	1877	1877	1875	1878	1881	1881	1880	1883	1883	1878
		$[1, 10^3]$	Best	**18775**	**18769**	**18755**	**18782**	**18810**	**18805**	**18804**	**18824**	**18830**	**18783**
			UB	18775	18769	18755	18782	18810	18805	18804	18824	18830	18783
		$[1, 10^4]$	Best	**187751**	**187692**	**187560**	**187824**	**188102**	**188060**	**188047**	**188245**	**188291**	**187832**
			UB	187751	187692	187560	187824	188102	188060	188047	188245	188291	187832
1000	5	$[1, 10^2]$	Best	**18801**	**18804**	**18801**	**18821**	**18812**	**18824**	**18807**	**18818**	**18820**	**18805**
			UB	18801	18804	18801	18821	18812	18824	18807	18818	18820	18805
		$[1, 10^3]$	Best	**188043**	**188038**	**188013**	**188211**	**188106**	**188206**	**188081**	**188167**	**188169**	**188061**
			UB	188043	188038	188013	188211	188106	188206	188081	188167	188169	188061
		$[1, 10^4]$	Best	**1880395**	**1880345**	**1880141**	**1882117**	**1881050**	**1882024**	**1880834**	**1881686**	**1881697**	**1880627**
			UB	1880395	1880345	1880141	1882117	1881050	1882024	1880834	1881686	1881697	1880627
	10	$[1, 10^2]$	Best	**9409**	**9421**	**9402**	**9396**	**9408**	**9404**	**9388**	**9408**	**9398**	**9406**
			UB	9409	9421	9402	9396	9408	9404	9388	9408	9398	9406
		$[1, 10^3]$	Best	**94087**	**94190**	**94012**	**93957**	**94091**	**94041**	**93926**	**94081**	**93993**	**94061**
			UB	94087	94190	94012	93957	94091	94041	93926	94081	93993	94061
		$[1, 10^4]$	Best	**940881**	**941899**	**940139**	**939562**	**940924**	**940407**	**939260**	**940806**	**939921**	**940600**
			UB	940881	941899	940139	939562	940924	940407	939260	940806	939921	940600
	25	$[1, 10^2]$	Best	**3766**	**3758**	**3762**	**3766**	**3759**	**3766**	**3764**	**3759**	**3760**	**3762**
			UB	3766	3758	3762	3766	3759	3766	3764	3759	3760	3762
		$[1, 10^3]$	Best	**37655**	**37585**	**37631**	**37660**	**37595**	**37658**	**37645**	**37598**	**37603**	**37622**
			UB	37655	37585	37631	37660	37595	37658	37645	37598	37603	37622
		$[1, 10^4]$	Best	**376546**	**375852**	**376315**	**376605**	**375956**	**376584**	**376449**	**375988**	**376037**	**376224**
			UB	376546	375852	376315	376605	375956	376584	376449	375988	376037	376224

Table B.2: Detailed results on all 390 non-uniform instances

Appendix C

German summary

In modernen Industrieunternehmen stellt die Produktionsplanung ein Kernelement dar, um die Performance eines Produktionssystems messbar machen zu können und mögliche Optimierungspotentiale in Bezug auf Effizienzgewinne aufzuzeigen. Die Planungsebenen lassen sich hierbei in Abhängigkeit ihrer zeitlichen Reichweite in strategische, taktische und operative Komponenten aufteilen (siehe Domschke et al., 1997). Vor allem Entscheidungen mit einem langfristigen Horizont sind in der Regel sehr komplex, da diese von einer dynamisch veränderlichen Umwelt und vielen weiteren stochastischen Einflüssen abhängig sind und einen weitreichenden Einfluss auf nachgelagerte Planungsebenen haben. Im Gegensatz dazu sind operative Aufgaben durch ihre häufig deterministische Datenbasis sehr gut durch die Anwendung von Methoden des Operations Research handhabbar.

Die vorliegende Dissertation zum Thema "Identische Parallel-Maschinen-Probleme: Struktureigenschaften, Schrankenberechnung and Lösungsmöglichkeiten" hat theoretische Analysen und Lösungsmethoden für einen Teilbereich der operativen Produktionsprozessplanung zum Inhalt. Es werden Probleme zur Verteilung von Arbeitslasten auf einer Menge von identischen Entitäten betrachtet. Probleme dieser Art treten typischerweise im Bereich der Maschinenbelegung (engl. Scheduling) auf, wo Aufträge mit individueller Laufzeit auf parallele, identische Maschinen zugeordnet werden. Vor allem im Kontext von Mass Customization sind häufig Problemstellungen aufzufinden, wo vergleichsweise geringe Mengen von individualisierten Produkten im Zuge einer Massenproduktion hergestellt werden. Die Vielzahl der möglichen Problemvarianten unterscheidet sich hinsichtlich der Definition und Ausgestaltung von Aufträgen und Maschinen sowie in Bezug auf die konkrete Zielstellung. So existieren sowohl Szenarien, in denen die Anzahl der verwendeten Maschinen fixiert ist, als auch andere in denen diese Anzahl minimiert werden soll. Simultan dazu kann das Ziel darin bestehen, die komplette Arbeitslast zu verteilen oder aufgrund von zusätzlichen Restriktionen die Anzahl der eingeplanten Aufträge zu maximieren.

Der bekannteste Vertreter auf dem Gebiet der identischen, parallelen Maschinen ist das Problem der Makespan-Minimierung $P||C_{max}$, wobei n Aufträge auf m Maschinen zu verteilen sind. Die zugrunde liegende Zielstellung, den spätesten Fertigstellungszeitpunkt $C_{max} = \max\{C_1, \ldots, C_i, \ldots C_m\}$ aller Maschinen zu minimieren, wobei C_i den Fertigstellungszeitpunkt von Maschine i bezeichnet, korreliert dabei unmittelbar mit der Effizienz eines Produktionssystems. Trotz seiner relativ einfachen Struktur, ist es eines der meist-

beforschten kombinatorischen Optimierungsproblem der letzten 50 Jahre (siehe z.B. McNaughton, 1959, Blazewicz, 1987, Lawler et al., 1993, Dell'Amico und Martello, 1995, Moktotoff, 2001, Walter und Lawrinenko, 2016). Das Gegenstück zu $P||C_{max}$, zu welchem jedoch deutlich weniger wissenschaftliche Veröffentlichungen existieren (siehe z.B. Woeginger, 1997, Haouari und Jemmali, 2008a, Walter et al., 2016), bildet das sogenannte Machine-Covering Problem $P||C_{min}$ bei welchem die Aufgabenstellung darin besteht, den frühesten Fertigstellungszeitpunkt $C_{min} = \min\{C_1, \ldots, C_i, \ldots C_m\}$ aller Maschinen zu maximieren. Probleme dieser Art treten beispielsweise im Zuge der fairen Aufteilung von Investitionsprojekten auf verschiedene Regionen auf (siehe Haouari und Jemmali, 2008b).

Für die beiden genannten Problemstellungen geht lediglich der Fertigstellungszeitpunkt einer einzigen Maschine in die Zielfunktion ein. Wenn jedoch zum Zwecke einer gleichmäßigen Abnutzung eine Ausbalancierung der Maschinen angestrebt wird, so ist die Betrachtung von alternativen Zielfunktionen sinnvoll. Ein einfaches Kriterium ist beispielsweise durch die Minimierung zwischen C_{max} und C_{min} Maschine gegeben (siehe Karmarkar und Karp, 1982, Coffman und Langston, 1984). Eine explizite Einbeziehung aller Fertigstellungszeitpunkte ist durch die Methode der kleinsten Quadrate gegeben, in welcher $\sum_{i=1}^{m}(C_i - \mu)^2$ minimiert wird und $\mu = \sum_{i=1}^{m} C_i/m$ den Mittelwert der Fertigstellungszeitpunkte darstellt (siehe z.B. Chandra und Wong, 1975, Alon et al., 1998, Walter und Lawrinenko, 2014, Schwerdfeger und Walter, 2016).

Wenn die Definition der Arbeitslastverteilung dahingehend geändert wird, dass die Anzahl der eingesetzen Entitäten optimiert werden soll, so resultieren weitere bekannte kombinatorische Optimierungsprobleme. Als ein typischer Vertreter lässt sich das Bin-Packing Problem benennen, bei welchem die Zielsetzung darin besteht, n Gegenstände auf eine möglichst geringe Anzahl von Behälter mit Kapazität C zu verteilen. Eine umfassende Literaturübersicht für das Bin-Packing Problem ist beispielsweise in Delorme et al. (2016) zu finden. Bin-Packing ist außerdem dual zum $P||C_{max}$-Problem was bedeutet, dass Bin-Packing Instanzen zu äquivalenten $P||C_{max}$ Instanzen transformiert werden können, indem die Bin-Kapazität C zu einer äquivalenten maximalen Produktionsdauer C transformiert wird. Hierbei besteht die Aufgabenstellung dann darin zu entscheiden, ob m Maschinen ausreichen um die komplette Arbeitslast unter dieser Beschränkung zu verteilen. Dieser Zusammenhang kann beispielsweise dazu genutzt werden um schärfere Schranken und bessere Lösungsmethoden für beide Problemvarianten zu definieren. In einer ähnlichen Weise ist das Bin-Covering Problem dual zu $P||C_{min}$. Hierbei besteht bei der Verteilung der Gegenstände die Aufgabe darin, für möglichst viele Behälter die Mindestkapazität C zu decken. Probleme dieser Art treten bei der Verpackung von Gütern mit individuellen Gewichten auf, wobei Pakete mit einem bestimmten Mindestgewicht resultieren sollen (z.B. bei Rispentomaten).

Begründet auf der Tatsache, dass seit fast 20 Jahren kein Survey mehr zum identischen Parallel-Maschinen-Problem veröffentlicht wurde, wird in Kapitel 2 der Arbeit eine umfassende Literaturübersicht bezüglich Schrankenberechnungen sowie heuristischen und exakten Lösungsverfahren vorgenommen. Dabei wird auf bekannte Probleme wie $P||C_{max}$, Workload Balancing und $P||\sum w_j C_j$ eingegangen (Kapitel 2.2). Zudem werden Veröf-

fentlichungen aus der Artificial Intelligence Community analysiert, wo Probleme dieser Art typischerweise als Number Partitioning bezeichnet werden (siehe z.B. Korf, 1998, Mertens, 2006, Moffitt, 2013). Die Forschungsaspekte der Bereiche Operations Research und Artificial Intelligence werden anschließend tabellarisch gegenübergestellt, wobei auf die individuellen Schwerpunkte eingegangen wird.

Die in der Arbeit betrachteten Optimierungsprobleme besitzen prinzipiell eine sehr einfache mathematische Struktur, dennoch ist in der Regel nur sehr wenig über Eigenschaften optimaler Lösungen bekannt. In Kapitel 3 der Arbeit wird deshalb das sogenannte Pfad-Konzept erläutert, durch welches universell gültige Eigenschaften von optimalen Lösungen identifiziert werden. Basierend auf einem Konzept der inversen Optimierung liegt die Idee hierbei darin begründet, unabhängig von den konkreten Auftragsdauern eine Untermenge aller möglichen Maschinenbelegungspläne zu ermitteln, in welcher mindestens eine optimale Lösung liegt. Das Pfad-Konzept wird in diesem Kapitel für das $P||C_{max}$-Problem definiert, kann jedoch auch für viele ähnlich strukturierte kombinatorische Optimierungsprobleme angewendet werden. Zunächst wird das Optimalitätsprinzip in Kapitel 3.2.2.1 für den Zwei-Maschinen-Fall erläutert und dann in Kapitel 3.2.2.2 für den generellen Fall ($m \geq 3$) überführt. Anschließend erfolgt die Transformation des Pfad-Konzeptes in eine Reihe von Dominanzkriterien (Kapitel 3.3), deren Effektivität in einem Branch-and-Bound Algorithmus untersucht werden (Kapitel 3.4 und 3.5). Dabei stellt sich heraus, das vor allem für Instanzen mit einem Quotienten $\frac{n}{m} < 3$ der Lösungsaufwand durch das Pfad-Konzept stark reduziert werden kann. Abschließend erfolgt ein Vergleich mit den exakten Lösungsverfahren von Dell'Amico und Martello (1995), Dell'Amico et al. (2008) und Haouari und Jemmali (2008b).

In Kapitel 4 der Arbeit wird eine theoretische Analyse des Machine-Covering Problems $P||C_{min}$ durchgeführt. Dazu werden in Kapitel 4.2 zunächst das Pfad-Konzept und dessen Anwendbarkeit für die Problemstellung erläutert. Anschließend werden neue Dominanzkriterien vorgestellt, die das Pfad-Konzept um die Komponente der Auftragsdauern erweitern (Kapitel 4.3). Daraufhin werden neue Methoden zur Berechnung von unteren und oberen Schranken präsentiert, die beispielsweise den Zusammenhang zum Bin-Covering Problem ausnutzen (Kapitel 4.4.1.4), aber zum anderen auch auf Lösungs- und Struktureigenschaften von $P||C_{min}$ beruhen (Kapitel 4.4.1.5). Simultan zu $P||C_{max}$ werden die gewonnenen Erkenntnisse dann in einem Branch-and-Bound Verfahren empirisch untersucht (Kapitel 4.5). Vor allem die Bin-Covering basierten oberen Schranken zeigen dabei eine hohe Effizienz. In Bezug auf den Schwierigkeitsgrad von Instanzen ergeben sich ähnliche Resultate wie beim zuvor betrachteten $P||C_{max}$-Problem.

Der Themenbereich Workload-Balancing wird im darauffolgenden Kapitel untersucht. Hierbei wird auf den Zusammenhang zwischen optimalen Lösungen für $P||C_{max}$ und $P||$NSSWD eingegangen, wobei das NSSWD-Kriterium eine normierte Variante der kleinsten Quadrate Methode darstellt. Der Aufhänger dieses Kapitel ist ein falsches Resultat aus Ho et al. (2009), welche behaupten dass ein NSSWD-optimaler Maschinenbelegungsplan stets auch C_{max} optimal ist. Es erfolgt eine Erläuterung für die Inkorrektheit dieses Resultates. Zudem wird ein Gegenbeweis in Form einer Beispielinstanz für beliebige m Werte einge-

führt (Kapitel 5.1). Eine vertiefende Untersuchung der Korrelation beider Zielfunktion erfolgt anschließend in Kapitel 5.2. Dabei stellt sich heraus, dass $P||$NSSWD optimale Schedules generell auch gut für $P||C_{max}$ geeignet sind. Das umgekehrte Resultat gilt allerdings nicht zwangsläufig.

In Kapitel 6 wird eine weitere Variante des Machine-Covering Problems betrachtet. Hierbei wird die Anzahl der möglichen Aufträge auf jeder Maschine durch einen individuellen Höchstwert k_i beschränkt. Das sogenannte k_i-Partitioning Problem tritt beispielsweise im Bereich flexibler Fertigungszellen auf. In Kapitel 6.2 werden Reduktionskriterien vorgestellt, welche dazu genutzt werden können, um den Lösungsraum einzuschränken in dem Kardinalitäten verschärft, sowie Maschinen und Aufträge entfernt werden ohne dabei die optimale Lösung zu verändern. Auch die Definition von neuartigen oberen Schranken (Kapitel 6.3) sowie heuristischer Eröffnungs- und Verbesserungsverfahren (Kapitel 6.4) ist ein wesentlicher Beitrag dieses Abschnittes. Abschließend wird die Performance der vorgestellten Prozeduren in einer neuartigen experimentellen Studie untersucht. Die generell geringen Abstände zwischen unteren und oberen Schranken lassen dabei auf eine hohe Effizienz der vorgestellten Prozeduren schließen.

In Kapitel 7 wird eine duale Problemvariante für $P||C_{min}$ vorgestellt. Beim sogenannten Minimum-Cardinality Bin-Covering (MCBCP) Problem besteht die Aufgabenstellung darin, m Behälter mit Kapazität C durch die Zuweisung von einer möglichsten geringen Anzahl von Gegenständen zu decken. Das Problem tritt beispielsweise beim Transport von verschiedenen Chemikalien, welche nicht gemischt werden dürfen, auf. Neben einer umfangreichen Literatureinordnung (Kapitel 7.1) werden technische Resultate in Form von Reduktionskriterien (Kapitel 7.2) und unteren Schranken (Kapitel 7.3) hergeleitet. Zudem werden heuristische Eröffnungsverfahren für das MCBCP adaptiert (Kapitel 7.4.1) und es wird eine Subset-Sum basierte Verbesserungsheuristik eingeführt (Kapitel 7.4.2). Basierend auf den vorher eingeführten Komponenten wird anschließend ein Branch-and-Bound Algorithmus definiert (Kapitel 7.4.3). Eine hohe Effizienz der vorgestellten Methoden wird in der abschließenden experimentellen Studie (Kapitel 7.5) aufgezeigt. Es können hierbei über 96% der Instanzen durch die Kombination aus Reduktionskriterien und Schrankenberechnung optimal gelöst werden. Zudem wird die Überlegenheit des Branch-and-Bound Verfahrens gegenüber dem Standardsolver Gurobi offensichtlich. Abschließend werden in Kapitel 8 die wesentlichen Resultate zusammengefasst und basierend auf bestehenden Forschungslücken, mögliche Ansätze für weitere Forschungspotentiale dargestellt. Durch die Generierung von neuartigen Lösungsmethoden in Kombination mit der Analyse von Lösungsstrukturen und der Definition von effizienten Schranken, wird in dieser Dissertation ein wesentlicher Beitrag zur Lösung von praxisrelevanten Instanzen für verschiedene kombinatorische Problemstellungen geleistet.

Appendix D

Ehrenwörtliche Erklärung

Hiermit erkläre ich,

1. dass mir die geltende Promotionsordnung bekannt ist;
2. dass ich die Dissertation selbst angefertigt, keine Textabschnitte eines Dritten oder eigener Prüfungsarbeiten ohne Kennzeichnung übernommen und alle von mir benutzten Hilfsmittel, persönlichen Mitteilungen und Quellen in meiner Arbeit angegeben habe;
3. dass ich bei der Auswahl und Auswertung des Materials sowie bei der Herstellung des Manuskriptes keine unzulässige Hilfe in Anspruch genommen habe;
4. dass ich nicht die Hilfe eines Promotionsberaters in Anspruch genommen habe und dass Dritte weder unmittelbar noch mittelbar geldwerte Leistungen von mir für Arbeiten erhalten haben, die im Zusammenhang mit dem Inhalt der vorgelegten Dissertation stehen;
5. dass ich die Dissertation noch nicht als Prüfungsarbeit für eine staatliche oder andere wissenschaftliche Prüfung eingereicht habe;
6. dass ich nicht die gleiche, eine in wesentlichen Teilen ähnliche oder eine andere Abhandlung bei einer anderen Hochschule bzw. anderen Fakultät als Dissertation eingereicht habe.

Appendix E

Curriculum vitae

Personal data

Date of birth:	July, 16th, 1986
Place of birth:	Saalfeld, Germany
Marital status:	married
Citizenship:	German

Work

since 10/2016 — **Business analyst, Fastlane Automotive Berlin**

System analysis and determination of pricing models

Education

04/2012 – 9/2016 — **Scientific assistant, Friedrich-Schiller-University Jena**

Work as scientific assistant at the chair for ABWL and Management Science at the facility of economics of the Friedrich-Schiller-University in Jena

PhD study on the topic: Identical Parallel Machine Scheduling Problems: Structural patterns, bounding techniques and solution procedures

Supervision of Bachelor-, Master- and Seminar theses

Experience as lecturer in Revenue Management, Introduction to programming, Project Management & Scheduling, Revenue Management, Decision Making and Computational Logistics and software course (Management Science and Introduction to MS Project)

09/2006 – 11/2011	**Study in Business Mathematics, Friedrich-Schiller-University Jena**
	Graduated with the title *Dipl.-Math. oec.*
07/2005 – 04/2006	**Community service, Arbeiterwohlfahrt Saalfeld**
08/1997 – 06/2005	**High school degree (Abitur), Heinrich-Boell-Gymnasium Saalfeld**
08/1993 – 06/1997	**Primary school, Saalfeld**

Scientific work

Published

02/2017	*Walter, R.; Lawrinenko, A.: Lower bounds and algorithms for the minimum cardinality bin covering problem. European Journal of Operational Research 256, 392–403.*
04/2016	*Walter, R.; Wirth, M.; Lawrinenko, A.: Improved approaches to the exact solution of the machine covering problem. Journal of Scheduling, DOI 10.1007/s10951-016-0477-x*
10/2014	*Walter, R.; Lawrinenko, A.: A note on minimizing the normalized sum of squared workload deviations on m parallel processors. Computers & Industrial Engineering, Vol. 75, 257-259*

Work in progress

Lawrinenko, A.: A survey on the identical parallel machine scheduling problem – Bounding techniques, approximation results, and solution approaches

Walter, R.; Lawrinenko, A.: Effective solution space limitation for the identical parallel machine scheduling problem

Lawrinenko, A.; Schwerdfeger, S.; Walter, R.: Reduction criteria, upper bounds, and a dynamic programming based heuristic for the k_i-partitioning problem

Additional experiences

02/2012 – 03/2012	Research assistant at the chair for ABWL and Management Science
11/2011 – 12/2011	Research assistant at the chair for ABWL and Management Science
04/2006 – 09/2010	Periodical work at Trumpf Medical in Saalfeld in different organization units, e.g. IT

Bibliography

[1] Ahuja, R. K.; Orlin, J. B. (2001): Inverse optimization. Operations Research 49, 771–783.

[2] Alidaee, B; Glover, F.; Kochenberger, G. A.; Rego, C. (2005): A new modeling and solution approach for the number partitioning problem. Journal of Applied Mathematics and Decision Sciences 9, 113–121.

[3] Alon, N.; Azar, Y.; Woeginger, G. J.; Yadid, T. (1998): Approximation schemes for scheduling on parallel machines. Journal of Scheduling 1, 55–66.

[4] Alvim, A. C. F.; Ribeiro, C. C. (2004): A hybrid bin-packing heuristic to multiprocessor scheduling. Lecture Notes in Computer Science 3059, 1–13.

[5] Alvim, A. C. F.; Ribeiro, C. C.; Glover, F.; Aloise, D. J. (2004): A hybrid improvement heuristic for the one-dimensional bin packing problem. Journal of Heuristics 10, 205–229.

[6] Argüello, M. F.; Feo, T. A.; Goldschmidt, O. (1996): Randomized methods for the number partitioning problem. Computers & Operations Research 23, 103–111.

[7] Assmann, S. F.; Johnson, D. S.; Kleitman, D. J. (1984): On a dual version of the one-dimensional bin packing problem. Journal of Algorithms 5, 502–525.

[8] Azar, Y.; Epstein, L. (1998): On-line machine covering. Journal of Scheduling 1, 67–77.

[9] Azizoglu, M.; Kirca, O. (1999): On the minimization of total weighted flow time with identical and uniform parallel machines. European Journal of Operational Research 113, 91–100.

[10] Babel, L.; Kellerer, H.; Kotov, V. (1998): The k-partitioning problem. Mathematical Methods of Operations Research 47, 59–82.

[11] Baker, K. R.; Merten, A. G. (1973): Scheduling with parallel processors and linear delay costs. Naval Research Logistics Quarterly 20, 793–804.

[12] Barnes, J. W.; Brennan, J. J. (1977): An improved algorithm for scheduling jobs on identical machines. AIIE Transactions 9, 25–31.

[13] Baum, S.; Trotter , L. E. Jr. (1981): Integer rounding for polymatroid and branching optimization problems. SIAM Journal on Algebraic and Discrete Methods 2, 416–425.

[14] Belouadah, H.; Potts, C. N. (1994): Scheduling identical parallel machines to minimize the total weighted completion time. Discrete Applied Mathematics 48, 201–218.

[15] Berretta, R.; Cotta, C.; Moscato, P. (2004): Enhancing the performance of memetic algorithms by using a matching-based recombination algorithm. In: Resende, M. G. C.; Sousa, J. P.; Viana, A. (Eds.): Metaheuristics: Computer Decision-Making. Kluwer Academic Publishers, Norwell, MA, USA, 65–90.

[16] Blazewicz, J. (1987): Selected topics in scheduling theory. Annals of Discrete Mathematics 31, 1–60.

[17] Brucker, P.; Shakhlevich, N. V. (2009): Inverse scheduling with maximum lateness objective. Journal of Scheduling 12, 475–488.

[18] Brucker, P.; Shakhlevich, N. V. (2011): Inverse scheduling: two-machine flow-shop problem. Journal of Scheduling 14, 239–256.

[19] Bruno, J.; Coffman Jr., E. G.; Sethi, R. (1974): Scheduling independent tasks to reduce mean finishing time. Communications of the ACM 17, 382–387.

[20] Bruno, J. L.; Downey, P. J. (1985): Probabilistic bounds for dual bin-packing. Acta Informatica 22, 333–345.

[21] Cai, S.-Y. (2007): Semi-online machine covering. Asia-Pacific Journal of Operational Research 24, 373–382.

[22] Chandra, A. K.; Wong, C. K. (1975): Worst-case analysis of a placement algorithm related to storage allocation. SIAM Journal on Computing 4, 249–263.

[23] Chen, J.; Pan, Q. K.; Wang, L.; Li, J. Q. (2012): A hybrid dynamic harmony search algorithm for identical parallel machines scheduling. Engineering Optimization 44, 209–224.

[24] Chen, S. P.; He, Y.; Lin, G. (2002): 3-Partitioning problems for maximizing the minimum load. Journal of Combinatorial Optimization 6, 67–80.

[25] Chen, S. P.; He, Y.; Yao, E.-Y. (1996): Three-partitioning containing kernels: Complexity and heuristic. Computing 57, 255–271.

[26] Cheng, T. C. E.; Liu, Z. (2004): Parallel machine scheduling to minimize the sum of quadratic completion times. IIE Transactions 36, 11–17.

[27] Cheng, T. C. E.; Sin, C. C. S. (1990): A state-of-the-art review of parallel-machine scheduling research. European Journal of Operational Research 47, 271–292.

[28] Chiaselotti, G.; Gualtieri, M. I.; Pietramala, P. (2010): Minimizing the makespan in nonpreemptive parallel machine scheduling problem. Journal of Mathematical Modelling and Algorithms in Operations Research 9, 39–51.

[29] Christofides, N.; Mingozzi, A.; Toth, P. (1979): Loading problems. In: Christofides, N.; Mingozzi, A.; Toth, P.; Sandi, C. (Eds.): Combinatorial Optimization. John Wiley and Sons, New York.

[30] Coffman Jr., E. G.; Csirik, J.; Galambos, G.; Martello, S.; Vigo, D. (2013): Bin packing approximation algorithms: Survey and classification. In: Pardalos, P. M.; Du, D.-Z.; Graham, R. L. (Eds.): Handbook of Combinatorial Optimization. 2nd edition, Springer, New York, 455–531.

[31] Coffman Jr., E. G.; Garey, M. R.; Johnson, D. S. (1978): An application of bin-packing to multiprocessor scheduling. SIAM Journal on Computing 7, 1–17.

[32] Coffman Jr., E. G.; Gilbert, E. N. (1985): On the expected performance of list scheduling. Operations Research 33, 548–561.

[33] Coffman Jr., E. G.; Langston, M. A. (1984): A performance guarantee for the greedy set-partitioning algorithm. Acta Informatica 21, 409–415.

[34] Coffman, E. G.; Leung, J. Y-T. (1979): Combinatorial analysis of an efficient algorithm for processor and storage allocation. SIAM Journal on Computing 8, 202–217.

[35] Coffman, E. G.; Leung, J. Y-T.; Ting, D. W. (1978): Bin packing: maximizing the number of pieces packed. Acta Informatica 9, 263–271.

[36] Conway, R. W.; Maxwell, W. L.; Miller, L. W. (1967): Theory of scheduling. Addison–Wesley, Reading.

[37] Cossari, A.; Ho, J. C.; Paletta, G.; Ruiz-Torres, A. J. (2012): A new heuristic for workload balancing on identical parallel machines and a statistical perspective on the workload balancing criteria. Computers & Operations Research 39, 1382–1393.

[38] Cossari, A.; Ho, J. C.; Paletta, G.; Ruiz-Torres, A. J. (2013): Minimizing workload balancing criteria on identical parallel machines. Journal of Industrial and Production Engineering 30, 160–172.

[39] Csirik, J.; Epstein, L.; Imreh, C.; Levin, A. (2010): On the sum minimization version of the online bin covering problem. Discrete Applied Mathematics 158, 1381–1393.

[40] Csirik, J.; Frenk, J. B. G.; Labbé, M.; Zhang, S. (1999): Two simple algorithms for bin covering. Acta Cybernetica 14, 13–25.

[41] Csirik, J.; Kellerer, H.; Woeginger, G. J. (1992): The exact LPT-bound for maximizing the minimum completion time. Operations Research Letters 11, 281–287.

[42] Davidović, T.; Šelmić, M.; Teodorović, D.; Ramljak, D. (2012): Bee colony optimization for scheduling independent tasks to identical processors. Journal of Heuristics 18, 549–569.

[43] Dell'Amico, M.; Iori, M.; Martello, S. (2004): Heuristic algorithms and scatter search for the cardinality constrained $P||C_{max}$ problem. Journal of Heuristics 10, 169–204.

[44] Dell'Amico, M.; Iori, M.; Martello, S.; Monaci, M. (2006): Lower bounds and heuristics for the k_i-partitioning problem. European Journal of Operational Research 171, 725–742.

[45] Dell'Amico, M.; Iori, M.; Martello, S.; Monaci, M. (2008): Heuristic and exact algorithms for the identical parallel machine scheduling problem. INFORMS Journal on Computing 20, 333–344.

[46] Dell'Amico, M.; Martello, S. (1995): Optimal scheduling of tasks on identical parallel processors. ORSA Journal on Computing 7, 191–200.

[47] Dell'Amico, M.; Martello, S. (2001): Bounds for the cardinality constrained $P||C_{max}$ problem. Journal of Scheduling 4, 123–138.

[48] Dell'Amico, M.; Martello, S. (2005): A note on exact algorithms for the identical parallel machine scheduling problem. European Journal of Operational Research 160, 576–578.

[49] Della Croce, F.; Koulamas, C. (2012): A note on minimizing the sum of quadratic completion times on two identical parallel machines. Information Processing Letters 112, 738–742.

[50] Delorme, M.; Iori, M.; Martello, S. (2016): Bin packing and cutting stock problems: Mathematical models and exact algorithms. European Journal of Operational Research 255, 1–20.

[51] Deuermeyer, B. L.; Friesen, D. K.; Langston, M. A. (1982): Scheduling to maximize the minimum processor finish time in a multiprocessor system. SIAM Journal on Algebraic and Discrete Methods 3, 190–196.

[52] Dobson, G. (1982): Worst-case analysis of greedy heuristics for integer programming with nonnegative data. Mathematics of Operations Research 7, 515–531.

[53] Domschke W.; Scholl, A.; Voß, S. (1997): Produktionsplanung: Ablauforganisatorische Aspekte. 2nd edition, Springer, Berlin.

[54] Eastman, W. L.; Even, S.; Isaacs, I. M. (1964): Bounds for the optimal scheduling of n jobs on m processors. Management Science 11, 268–279.

[55] Ebenlendr, T.; Noga, J.; Sgall, J.; Woeginger, G. J. (2006): A note on semi-online machine covering. In: Erlebach, T.; Persiano, G. (Eds.): Approximation and Online Algorithms, Lecture Notes in Computer Science 3879, 110–118. Springer, Berlin.

[56] Elmaghraby, S. E.; Park, S. H. (1974): Scheduling jobs on a number of identical machines. AIIE Transactions 6, 1–13.

[57] Epstein, L.; Imreh, C.; Levin, A. (2010): Class constrained bin covering. Theory of Computing Systems 46, 246–260.

[58] Epstein, L.; Imreh, C.; Levin, A. (2013): Bin covering with cardinality constraints. Discrete Applied Mathematics 161, 1975–1987.

[59] Epstein, L.; Levin, A.; van Stee, R. (2011): Max-min online allocations with a reordering buffer. SIAM Journal on Discrete Mathematics 25, 1230–1250.

[60] Epstein, L.; Sgall, J. (2004): Approximation schemes for scheduling on uniformly related and identical parallel machines. Algorithmica 39, 43–57.

[61] Fatemi-Ghomi, S. M. T.; Jolai-Ghazvini, F. (1998): A pairwise interchange algorithm for parallel machine scheduling. Production Planning & Control 9, 685–689.

[62] Fekete, S. P.; Schepers, J. (2001): New classes of fast lower bounds for bin packing problems. Mathematical Programming 91, 11–31.

[63] Fleszar, K.; Charalambous, C. (2011): Average-weight-controlled bin-oriented heuristics for the one-dimensional bin-packing problem. European Journal of Operational Research 210, 176–184.

[64] Foster, D. P.; Vohra, R. V. (1989): Probabilistic analysis of a heuristics for the dual bin packing problem. Information Processing Letters 31, 287–290.

[65] França, P. M.; Gendreau, M.; Laporte, G.; Müller, F. M. (1994): A composite heuristic for the identical parallel machine scheduling problem with minimum makespan objective. Computers & Operations Research 21, 205–210.

[66] Frenk, J. B. G.; Rinnooy Kan, A. H. G. (1986): The rate of convergence to optimality of the LPT rule. Discrete Applied Mathematics 14, 187–197.

[67] Frenk, J. B. G.; Rinnooy Kan, A. H. G. (1987): The asymptotic optimality of the LPT rule. Mathematics of Operations Research 12, 241–254.

[68] Finn, G.; Horowitz, E. (1979): A linear time approximation algorithm for multiprocessor scheduling. BIT Numerical Mathematics 19, 312–320.

[69] Frangioni, A.; Necciari, E.; Scutellà, M. G. (2004): A multi-exchange neighborhood for minimum makespan parallel machine scheduling problems. Journal of Combinatorial Optimization 8, 195–220.

[70] Friesen, D. K. (1984): Tighter bounds for the Multifit processor scheduling algorithm. SIAM Journal on Computing 13, 170–181.

[71] Friesen, D. K.; Deuermeyer, B. L. (1981): Analysis of greedy solutions for a replacement part sequencing problem. Mathematics of Operations Research 6, 74–87.

[72] Friesen, D. K.; Langston, M. A. (1986): Evaluation of a Multifit-based scheduling algorithm. Journal of Algorithms 7, 35–59.

[73] Friesen, D. K.; Langston, M. A. (1987): Bin packing: On optimizing the number of pieces packed. BIT Numerical Mathematics 27, 148–156.

[74] Fukunaga, A. S.; Korf, R. E. (2007): Bin completion algorithms for multicontainer packing, knapsack, and covering problems. Journal of Artificial Intelligence Research 28, 393–429.

[75] Garey, M. R.; Johnson, D. S. (1979): Computers and intractability: a guide to the theory of NP-completeness. W. H. Freeman & Co., San Francisco.

[76] Gent, I. P.; Walsh, T. (1998): Analysis of heuristics for number partitioning. Computational Intelligence 14, 430–451.

[77] Goldberg, R. R.; Shapiro, J. (1999): A tight upper bound for the k-partition problem on ideal sets. Operations Research Letters 24, 165–173.

[78] Goldberg, R. R.; Shapiro, J. (2000): Partitioning under the L_p norm. European Journal of Operational Research 123, 585–592.

[79] Graham, R. L. (1966): Bounds for certain multiprocessing anomalies. Bell System Technical Journal 45, 1563–1581.

[80] Graham, R. L. (1969): Bounds on multiprocessing timing anomalies. SIAM Journal on Applied Mathematics 17, 416–429.

[81] Graham, R. L.; Lawler, E. L.; Lenstra, J. K.; Rinnooy Kan, A. H. G. (1979): Optimization and approximation in deterministic sequencing and scheduling: A survey. Annals of Discrete Mathematics 5, 287–326.

[82] Gualtieri, M. I.; Paletta, G.; Pietramala, P. (2008): A new nlogn algorithm for the identical parallel machine scheduling problem. International Journal of Contemporary Mathematical Sciences 3, 25–36.

[83] Gualtieri, M. I.; Pietramala, P.; Rossi, F (2009): Heuristic algorithms for scheduling jobs on identical parallel machines via measures of spread, International Journal of Applied Mathematics 39, 100–107.

[84] Gupta, J. N. D.; Ruiz-Torres, A. J. (2001): A LISTFIT heuristic for minimizing makespan on identical parallel machines. Production Planning & Control 12, 28–36.

[85] Haouari, M.; Gharbi, A.; Jemmali, M. (2006): Tight bounds for the identical parallel machine-scheduling problem. International Transactions in Operational Research 13, 529–548.

[86] Haouari, M.; Jemmali, M. (2008a): Maximizing the minimum completion time on parallel machines. 4OR: A Quarterly Journal of Operations Research 6, 375–392.

[87] Haouari, M.; Jemmali, M. (2008b): Tight bounds for the identical parallel machine-scheduling problem: part II. International Transactions in Operational Research 15, 19–34.

[88] He, Y.; Tan, Z. Y. (2002): Ordinal on-line scheduling for maximizing the minimum machine completion time. Journal of Combinatorial Optimization 6, 199–206.

[89] He, Y.; Tan, Z.; Zhu, J.; Yao, E. (2003): k-Partitioning problems for maximizing the minimum load. Computers & Mathematics with Applications 46, 1671–1681.

[90] Heuberger, C. (2004): Inverse combinatorial optimization: A survey on problems, methods, and results. Journal of Combinatorial Optimization 8, 329–361.

[91] Ho, J. C.; Tseng,T. -L. B.; Ruiz-Torres, A. J.; López, F. J. (2009): Minimizing the normalized sum of square for workload deviations on m parallel processors. Computers & Industrial Engineering 56, 186–192.

[92] Ho, J. C.; Wong, J. S. (1995): Makespan minimization for m parallel identical processors. Naval Research Logistics 42, 935–948.

[93] Hochbaum, D. S.; Shmoys, D. B. (1987): Using dual approximation algorithms for scheduling problems: practical and theoretical results. Journal of the ACM 34, 144–162.

[94] Horowitz, E.; Sahni, S. (1974): Computing partitions with applications to the knapsack problem. Journal of the ACM 21, 277–292.

[95] Hübscher, R.; Glover, F. (1994): Applying tabu search with influential diversification to multiprocessor scheduling. Computers & Operations Research 21, 877–884.

[96] Johnson, D. S.; Aragon, C. R.; McGeoch, L. A.; Schevon, C. (1991): Optimization by simulated annealing: An experimental evaluation: Part II. Graph coloring and number partitioning. Operations Research 39, 378–406.

[97] Karmarkar, N.; Karp, R. M. (1982): The differencing method of set partitioning. Technical Report No. UCB/CSD 82/113, University of California, Berkeley.

[98] Karmarkar, N.; Karp, R. M.; Lueker, G. S.; Odlyzko, A. M. (1986): Probabilistic analysis of optimum partitioning. Journal of Applied Probability 23, 626–645.

[99] Kashan, A. H.; Karimi, B. (2009): A discrete particle swarm optimization algorithm for scheduling parallel machines. Computers & Industrial Engineering 56, 216–223.

[100] Kawaguchi, T.; Kyan, S. (1986): Worst case bound of an LRF schedule for the mean weighted flow-time problem. SIAM Journal on Computing 15, 1119–1129.

[101] Kellerer, H.; Kotov, V. (1999): A 7/6-approximation algorithm for 3-partitioning and its application to multiprocessor scheduling. Information Systems and Operational Research 37, 48–56.

[102] Kellerer, H.; Kotov, V. (2011): A 3/2-approximation algorithm for k_i-partitioning. Operations Research Letters 39, 359–362.

[103] Kellerer, H.; Woeginger, G. J. (1993a): A greedy-heuristic for 3-partitioning with similar elements. Computing 50, 271–278.

[104] Kellerer, H.; Woeginger, G. J. (1993b): A tight bound for 3-partitioning. Discrete Applied Mathematics 45, 249–259.

[105] Kolliopoulos, S. G. (2003): Approximating covering integer programs with multiplicity constraints. Discrete Applied Mathematics 129, 461–473.

[106] Kolliopoulos, S. G.; Young, N. E. (2001): Tight approximation results for general covering integer programs. Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science (IEEE 42), 522–528.

[107] Kolliopoulos, S. G.; Young, N. E. (2005): Approximation algorithms for covering/packing integer programs. Journal of Computer and System Sciences 71, 495–505.

[108] Korf, R. E. (1998): A complete anytime algorithm for number partitioning. Artificial Intelligence 106, 181–203.

[109] Korf, R. E. (2009): Multi-way number partitioning. In Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009), 538–543.

[110] Korf, R. E. (2011): A hybrid recursive multi-way number partitioning algorithm. In Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011), 591–596.

[111] Koulamas, C. (2005): Inverse scheduling with controllable job parameters. International Journal of Services and Operations Management 1, 35–43.

[112] Koulamas, C.; Kyparisis, G. J. (2008): An improved delayed-start LPT algorithm for a partition problem on two identical parallel machines. European Journal of Operational Research 187, 660–666.

[113] Kumar, N.; Shanker, K. (2001): Comparing the effectiveness of workload balancing objectives in FMS loading. International Journal of Production Research 39, 843–871.

[114] Labbé, M.; Laporte, G.; Martello, S. (1995): An exact algorithm for the dual bin packing problem. Operations Research Letters 17, 9–18.

[115] Labbé, M.; Laporte, G.; Martello, S. (2003): Upper bounds and algorithms for the maximum cardinality bin packing problem. European Journal of Operational Research 149, 490–498.

[116] Langston, M. A. (1982): Improved 0/1 interchange scheduling. BIT Numerical Mathematics 22, 282–290.

[117] Langston, M. A. (1984): Performance of heuristics for a computer resource allocation problem. SIAM Journal on Algebraic Discrete Methods 5, 154–161.

[118] Lawler, E. L.; Lenstra, J. K.; Rinnooy Kan, A. H. G.; Shmoys, D. B. (1993): Sequencing and scheduling: algorithms and complexity. In Graves, S. C.; Rinnooy Kan, A. H. G.; Zipkin, P. H. (Eds.): Logistics of Production and Inventory, vol. 4 of Handbooks in Operations Research and Management Science. North-Holland, Amsterdam, 445–522.

[119] Lawrinenko, A.; Schwerdfeger, S.; Walter, R. (2016): Reduction criteria, upper bounds, and a dynamic programming based heuristic for the k_i-partitioning problem. Working Paper.

[120] Lee, C. Y.; Massey, J. D. (1988): Multiprocessor scheduling: combining LPT and Multifit. Discrete Applied Mathematics 20, 233–242.

[121] Lee, C. Y.; Uzsoy, R. (1992): A new dynamic programming algorithm for the parallel machines total weighted completion time problem. Operations Research Letters 11, 73–75.

[122] Lee, W. C.; Wu, C. C.; Chen, P. (2006): A simulated annealing approach to makespan minimization on identical parallel machines. The International Journal of Advanced Manufacturing Technology 31, 328–334.

[123] Leung, J. Y. T.; Wei, W. D. (1995): Tighter bounds on a heuristic for a partition problem. Information Processing Letters 56, 51–57.

[124] Loh, K.-H.; Golden, B.; Wasil, E. (2009): Solving the maximum cardinality bin packing problem with a weight annealing-based algorithm. In: Chinneck, J. W.; Kristjansson, B.; Saltzman, M. J. (Eds.): Operations Research and Cyber-Infrastructure. Springer, New York, 147–164.

[125] Lübbecke, M. E.; Desrosiers, J. (2005): Selected topics in column generation. Operations Research 53, 1007–1023.

[126] Lueker, G. S. (1987): A note on the average-case behavior of a simple differencing method for partitioning. Operations Research Letters 6, 285–287.

[127] Luo, R.-Z.; Sun, S.-J. (2005): Semi on-line scheduling problem for maximizing the minimum machine completion time on m identical machines. Journal of Shanghai University 9, 99–102.

[128] Martello, S.; Toth, P. (1984): A mixture of dynamic programming and branch-and-bound for the subset sum problem. Management Science 30, 765–771.

[129] McNaughton, R. (1959): Scheduling with deadlines and loss functions. Management Science 6, 1–12.

[130] Mertens, S. (1998): Phase transition in the number partitioning problem. Physical Review Letters 81, 4281–4284.

[131] Mertens, S. (1999): A complete anytime algorithm for balanced number partitioning. Preprint xxx.lanl.gov/abs/cs.DS/9903011.

[132] Mertens, S. (2006): The easiest hard problem: Number partitioning. In: Percus, A. G.; Istrate, G.; Moore, C. (Eds.): Computational Complexity and Statistical Physics. Oxford University Press, New York, 125–139.

[133] Michiels, W.; Aarts, E.; Korst, J.; van Leeuwen, J.; Spieksma, F. C. R. (2012): Computer-assisted proof of performance ratios for the differencing method. Discrete Optimization 9, 1–16.

[134] Michiels, W.; Korst, J.; Aarts, E.; van Leeuwen, J. (2007): Performance ratios of the Karmarkar-Karp differencing method. Journal of Combinatorial Optimization 13, 19–32.

[135] Min, L.; Cheng, W. (1999): A genetic algorithm for minimizing the makespan in the case of scheduling identical parallel machines. Artificial Intelligence in Engineering 13, 399–403.

[136] Moffitt, M. D. (2013): Search strategies for optimal multi-way number partitioning. In Proceedings of the 23rd international Joint Conference on Artificial Intelligence (IJCAI 13), 623–629.

[137] Mokotoff, E. (2001): Parallel machine scheduling problems: A survey. Asia Pacific Journal of Operational Research 18, 193–242.

[138] Mokotoff, E. (2004): An exact algorithm for the identical parallel machine scheduling problem. European Journal of Operational Research 152, 758–769.

[139] Ouazene, Y.; Yalaoui, F.; Chehade, H.; Yalaoui, A. (2014) Workload balancing in identical parallel machine scheduling using a mathematical programming method. International Journal of Computational Intelligence Systems 7, 58–67.

[140] Paletta, G.; Pietramala, P. (2007): A new approximation algorithm for the nonpreemptive scheduling of independent jobs on identical parallel processors. SIAM Journal on Discrete Mathematics 21, 313–328.

[141] Paletta, G.; Ruiz-Torres, A. J. (2015): Partial solutions and Multifit algorithm for multiprocessor scheduling. Journal of Mathematical Modelling and Algorithms in Operations Research 14, 125–143.

[142] Paletta, G.; Vocaturo, F. (2011): A composite algorithm for multiprocessor scheduling. Journal of Heuristics 17, 281–301.

[143] Pedroso, J. P.; Kubo, M. (2010): Heuristics and exact methods for number partitioning. European Journal of Operational Research 202, 73–81.

[144] Peeters, M.; Degraeve, Z. (2006): Branch-and-price algorithms for the dual bin packing and maximum cardinality bin packing problem. European Journal of Operational Research 170, 416–439.

[145] Rajakumar, S.; Arunachalam, V. P.; Selladurai, V. (2004): Workflow balancing strategies in parallel machine scheduling. The International Journal of Advanced Manufacturing Technology 23, 366–374.

[146] Rhee, W. T.; Talagrand, M. (1993): Dual bin packing with items of random sizes. Mathematical Programming 58, 229–242.

[147] Riera, J.; Alcaide, D.; Sicilia, J. (1996): Approximate algorithms for the $P||C_{max}$ problem. Top 4, 345–359.

[148] Rothkopf, M. H. (1966): Scheduling independent tasks on parallel processors. Management Science 12, 437–447.

[149] Ruml, W.; Ngo, J. T.; Marks, J.; Shieber, S. M. (1996): Easily searched encodings for number partitioning. Journal of Optimization Theory and Applications 89, 251–291.

[150] Sahni, S. (1976): Algorithms for scheduling independent tasks. Journal of the ACM 23, 116–127.

[151] Sarin, S. C.; Ahn, S.; Bishop, A. B. (1988): An improved branching scheme for the branch and bound procedure of scheduling n jobs on m parallel machines to minimize total weighted flowtime. International Journal of Production Research 26, 1183–1191.

[152] Scheithauer, G.; Terno, J. (1995): The modified integer round-up property of the one-dimensional cutting stock problem. European Journal of Operational Research 84, 562–571.

[153] Schreiber, E. L.; Korf, R. E. (2014): Cached iterative weakening for optimal multiway number partitioning. In Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI 2014), 2738–2744.

[154] Schroeppel, R.; Shamir, A. (1981): A $T = \mathcal{O}(2^{n/2}), S = \mathcal{O}(2^{n/4})$ algorithm for certain NP-complete problems. SIAM Journal on Computing 10, 456–464.

[155] Schulz, A. (1996): Scheduling to minimize total weighted completion time: performance guarantees of LP-based heuristics and lower bounds. Integer Programming and Combinatorial Optimization 1084, 301–315.

[156] Schwerdfeger, S.; Walter, R. (2016): A fast and effective subset sum based improvement procedure for workload balancing on identical parallel machines. Computers & Operations Research 73, 84–91.

[157] Skutella, M.; Woeginger, G. J. (2000): A PTAS for minimizing the total weighted completion time on identical parallel machines. Mathematics of Operations Research 25, 63–75.

[158] Srinivasan, A. (1999): Improved approximation guarantees for packing and covering integer programs. SIAM Journal on Computing 29, 648–670.

[159] Tan, Z.; Wu, Y. (2007): Optimal semi-online algorithms for machine covering. Theoretical Computer Science 372, 69–80.

[160] Tang, L.; Luo, J. (2006): A new ILS algorithm for parallel machine scheduling problems. Journal of Intelligent Manufacturing 17, 609–619.

[161] Tang, L.; Xuan, H.; Liu, J. (2006): A new lagrangian relaxation algorithm for hybrid flowshop scheduling to minimize total weighted completion time. Computers & Operations Research 33, 3344–3359.

[162] Tasi, L.-H. (1995): The modified differencing method for the set partitioning problem with cardinality constraints. Discrete Applied Mathematics 63, 175–180.

[163] Thesen, A. (1998): Design and evaluation of tabu search algorithms for multiprocessor scheduling. Journal of Heuristics 4, 141–160

[164] Tsai, L.-H. (1992): Asymptotic analysis of an algorithm for balanced parallel processor scheduling. SIAM Journal on Computing 21, 59–64.

[165] Valério de Carvalho, J. M. (2005): Using extra dual cuts to accelerate column generation. INFORMS Journal on Computing 17, 175–182.

[166] van den Akker, J. M.; Hoogeveen, J. A.; van de Velde, S. L. (1999): Parallel machine scheduling by column generation. Operations Research 47, 862–872.

[167] Vijayakumar, B.; Parikh, P. J.; Scott, R.; Barnes, A.; Gallimore, J. (2013): A dual bin-packing approach to scheduling surgical cases at a publicly-funded hospital. European Journal of Operational Research 224, 583–591.

[168] Walter, R. (2013): Comparing the minimum completion times of two longest-first scheduling-heuristics. Central European Journal of Operations Research 21, 125–139.

[169] Walter, R. (2015): A note on minimizing the sum of squares of machine completion times on two identical parallel machines. Central European Journal of Operations Research. DOI: 10.1007/s10100-015-0429-0.

[170] Walter, R.; Lawrinenko, A. (2014): A note on minimizing the normalized sum of squared workload deviations on m parallel processors. Computers & Industrial Engineering 75, 257–259.

[171] Walter, R.; Lawrinenko, A. (2016): Effective solution space limitation for the identical parallel machine scheduling problem. Working Paper.

[172] Walter, R.; Lawrinenko, A. (2017): Lower bounds and algorithms for the minimum cardinality bin covering problem. European Journal of Operational Research 256, 392–403.

[173] Walter, R.; Wirth, M.; Lawrinenko, A. (2016): Improved approaches to the exact solution of the machine covering problem. Journal of Scheduling (online first), DOI 10.1007/s10951-016-0477-x.

[174] Webster, S. T. (1992): New bounds for the identical parallel processor weighted flow time problem. Management Science 38, 124–136.

[175] Webster, S. T. (1996): A general lower bound for the makespan problem. European Journal of Operational Research 89, 516–524.

[176] Woeginger, G. J. (1997): A polynomial-time approximation scheme for maximizing the minimum machine completion time. Operations Research Letters 20, 149–154.

[177] Woeginger, G. J. (2005): A comment on scheduling two parallel machines with capacity constraints. Discrete Optimization 2, 269–272.

[178] Xu, J.; Nagi, R. (2013): Identical parallel machine scheduling to minimise makespan and total weighted completion time: a column generation approach. International Journal of Production Research 51, 7091–7104.

[179] Yakir, B. (1996): The differencing algorithm LDM for partitioning: a proof of a conjecture of Karmarkar and Karp. Mathematics of Operations Research 21, 85–99.

[180] Yue, M. (1990): On the exact upper bound for the Multifit processor scheduling algorithm. Annals of Operations Research 24, 233–259.

[181] Zhang, C.; Wang, G.; Liu, X.; Liu, J. (2009): Approximating scheduling machines with capacity constraints. In: Deng, X.; Hopcroft, J. E.; Xue, J. (Eds.): Frontiers in Algorithmics, Lecture Notes in Computer Science 5598, Springer, Berlin, 283–292.

[182] Zhang, J.; Mouratidis, K.; Pang, H. H. (2011): Heuristic algorithms for balanced multi-way number partitioning. In Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011), 693–698.