Online Scheduling & Project Scheduling

Jacob Jan Paulus

Online Scheduling & Project Scheduling

by

Jacob Jan Paulus

Composition of the Graduation Committee:

prof.dr. R.J. Boucherie	(Universiteit Twente)
dr. J.L. Hurink	(Universiteit Twente)
dr. W. Kern	(Universiteit Twente)
prof.dr. C.N. Potts	(University of Southampton)
prof.dr. M. Skutella	(Technische Universität Berlin)
prof.dr. M.J. Uetz	(Universiteit Twente)
prof.dr. G.J. Woeginger	(Technische Universiteit Eindhoven)

UT / EEMCS / AM / DMMP
P.O. box 217, 7500 AE Enschede
The Netherlands

BETA, Research School for Operations Management and Logistics.
Beta Dissertation Series D115

The research in this thesis is financially supported by the Dutch government in the BSIK/BRICKS project, theme Intelligent Systems 3: Decision Support Systems for Logistic Networks and Supply Chain Optimization.

Keywords: online scheduling, competitive analysis, parallel jobs, batch scheduling, project scheduling, heuristics, deadlines, adjacent resources.

This thesis was edited with Kile and typeset with LaTeX.
Printed by Wöhrmann Print Service, Zutphen, The Netherlands.

ISBN 978-90-365-2753-8
http://dx.doi.org/10.3990/1.9789036527538

ONLINE SCHEDULING & PROJECT SCHEDULING

PROEFSCHRIFT

ter verkrijging van
de graad van doctor aan de Universiteit Twente,
op gezag van de rector magnificus,
prof.dr. H. Brinksma,
volgens besluit van het College voor Promoties
in het openbaar te verdedigen
op donderdag 22 januari 2009 om 16.45 uur

door

Jacob Jan Paulus

geboren op 13 november 1980
te Opsterland

Dit proefschrift is goedgekeurd door
prof.dr. M.J. Uetz (promotor)
dr. J.L. Hurink (assistent-promotor)

Voor Johannes en Alie

Preface

This thesis is the result of four years working together with many interesting people. Therefore, it seems unfair that only my name is on the cover of this book. I would like to pause and say a few words of thanks.

First and foremost, I have to thank Johann Hurink. Back when I was writing my Master's thesis he was the one who encouraged me to pursue a Ph.D. degree. He gave me the opportunity and freedom to work on the scheduling problems described in this thesis. The advice he gave me as a supervisor and sometimes as a mentor was of great value to me. I enjoyed the many discussions we had, both on and off subject. By enjoying each others suggestions and enthusiasm, and by debating as being peers, he gave me much more than I could possibly ask for.

Secondly, I am thankfull to the people that I had the privilege and pleasure to worked with, in particular to Marco Schutten and Walter Kern. Our stimulating discussions and their insights are of great value to me and my work. The research done together with Tom Guldemond and Leendert Kok has not only resulted in their graduation, but turned out to be the start of the research described in the last two chapters of this book.

Furthermore, I am gratefull to the people working in the group of Discrete Mathematics and Mathematical Programming at the University of Twente, both working there currently and in the past. Our joint coffee breaks and the small talk in the corridor made me feel at home.

Finally, without the continuous support and interest of loved ones, friends and family the completion of this thesis had not been possible. Thank you for reminding me that there is more to life than mathematics. Parents, you cannot begin to imagine the enormity of your contribution.

Thank you. Jacob Jan Paulus

Abstract

This thesis presents new and improved scheduling algorithms for a number of scheduling problems. The first part of the thesis deals with two online-list scheduling problems, both with the objective to minimize the makespan. In these problems, the jobs arrive one by one and the decision maker has to irrevocably schedule the arriving job before the next job becomes known. In the second part, a new solution methodology for the time-constrained project scheduling problem is developed, and a decomposition method for the time-constrained project scheduling problem with adjacent resources is presented.

The first online problem considered, is online-list scheduling of parallel jobs with the objective to minimize the makespan, and some of its special cases. In these problems there is a set of identical machines to process a set of parallel jobs. In contrast to the jobs in classical machine scheduling problems, parallel jobs require a number of machines simultaneously for their processing. For this problem, a 6.6623-competitive online algorithm and a lower bound of 2.43 on the competitive ratio of any online algorithm are presented. Both results are also applicable to the online orthogonal strip packing problem. Besides the tight lower bound of 2 on the competitive ratio of the problem with exactly two machines, also improved online algorithms for the case with exactly three machines and the semi-online case where jobs appear in non-increasing order of machine requirement are given.

The second online problem covered, is the online-list batch scheduling problem with the objective to minimize the makespan. In this problem, there is one machine with a given batch capacity that processes the jobs in a parallel batching manner. Parallel batching means that all jobs in one batch receive processing at the same time. A complete classification of the tractability of this online problem is given; with respect to the competitive ratio an optimal online algorithm for any capacity

of the batching machine is presented.

The second part of this thesis presents a new scheduling approach for scheduling with strict deadlines on the jobs. This approach is applied to the time-constrained project scheduling problem. In this problem there is a set of resources, each with its own capacity, and a set of jobs requiring a specific amount of each of the resources during its processing. Additionally, there are precedence relations between the jobs and each job has a deadline. To be able to meet the deadlines in this problem, it is possible to work in overtime or hire additional capacity in regular time or overtime. In the developed two stage heuristic, the key step lies in the first stage where partial schedules are constructed. In these partial schedules, jobs may be scheduled for a shorter duration than required. The goal is to create a schedule in which the usage of overtime and extra hiring is low. The proposed method tries to prevent a pile up of costs toward the deadlines by including all jobs partially before addressing the bottlenecks. Computational tests are preformed on modified resource-constrained project scheduling benchmark instances. Many instances are solved to optimality, and lead only to a small increase of costs if the deadline is substantially decreased.

Finally, the time-constrained project scheduling problem is considered under the addition of a new type of constraint, namely an adjacent resource constraint. An adjacent resource constraint requires that the units of the resource assigned to a job have to be adjacent. On top of that, adjacent resources are not required by single jobs, but by job groups. The additional complexity introduced by adding adjacent resources to the project scheduling problem, plays an important role both for the computational tractability and the algorithm design when solving the problem. The developed decomposition method separates the adjacent resource assignment from the rest of the scheduling problem. Once the job groups are assigned to the adjacent resource, the scheduling of the jobs can be done without considering the adjacent resource. The presented decomposition method forms a first promising approach for the time-constrained project scheduling problem with adjacent resources and may form a good basis to develop more elaborated methods.

Contents

1 Introduction

1.1 Scheduling

In what order should the products be produced? Which machines should be used to execute the different processes? When should we start the production? These type of questions are frequently asked by people who have to plan and schedule operations. Their role is to let the organization run smoothly and be cost efficient; they need to allocate scarce resources in such a way that the objectives and goals of the organization are achieved [67].

Operations Research is the discipline of applied mathematics that studies decision making as to improve processes. In Operations Research techniques like mathematical modeling, algorithms, statistics and simulation are employed to support decision making in fields, such as scheduling, routing, game theory and queuing theory. The discipline is not only broad in the fields it covers, it also ranges all the way from theory to practice.

Most problems that are treated in Operations Research are optimization type of problems. One has to find a solution that minimizes (or maximizes) some objective function taking into account a set of constraints that have to be satisfied. To illustrate the way optimization problems are stated, consider the well known Traveling Salesman Problem that was already studied in the 1800s [5]: *Given a number of cities and the distances between them, find a round-trip of minimum distance that visits each city exactly once and returns to the starting city.* Clearly, many problems from the transportation sector boil down to (a variant of) this classical problem.

Operations Research evolved during World War II, when better decision making in military logistics and training schedules became necessary. The name Operations Research stems from this era; from optimizing the military operations. After the war, the interest of academia and industry in the discipline hugely increased. Industry

adopted the developed methods to optimize their production and manufacturing processes. Nowadays good decision making in planning and scheduling is a necessity for manufactures and businesses to stay competitive.

Within Operations Research, scheduling is the branch that concerns the allocation of scarce resources to tasks over time. In industry, scheduling problems arise on all three levels of decision making, that is strategic, tactical and operational. However, scheduling is most often associated with the operational level where more details are taken into account and shorter time horizons are used.

In this thesis we design and analyze algorithms for a number of different scheduling problems. In these scheduling problems two key characteristics play an important role; either the resource is required in some specific parallel manner or there are strict deadlines. To give an example of one parallel resource requirement, consider parallel computing. Computational tasks have to be performed by a multi-processor computer, and each of these tasks requires a specific number of processors in parallel during the computation.

In Section 1.7 we give an outline of the thesis and describe for each chapter the scheduling problems discussed.

The remainder of this chapter gives an introduction to the mathematical concepts used in this thesis. This introduction is not intended to be complete, therefore we give in each section references to further reading.

1.2 Machine scheduling

Most of the theoretical research on scheduling problems can be found in the area of machine scheduling. In a machine scheduling problem we are given m machines on which n jobs have to be processed. Each machine can process exactly one job at the time. The goal is to construct a schedule as to optimize some objective function, for example to minimize the sum of the completion times of all the jobs. A solution of a machine scheduling problem, a schedule, can be described by an assignment of the jobs to the machines and the start times of the jobs, or represented by a so-called Gantt-chart. Regularly, in these charts the machines are depicted on the vertical axis and time on the horizontal axis. The jobs then correspond to blocks in the chart. See Figure 1.1 for an example Gantt-chart.

A machine scheduling problem is characterized by three aspects: the machine environment (α), the job properties (β) and the objective function (γ). Each scheduling problem has, therefore, a compact description in the form of the triplet $\alpha|\beta|\gamma$. This three field notation is introduced in [30] and extended by many other researchers. For an introduction to this notation see for example [8, 66].

To illustrate the use of the three field notation, consider the problems we deal with in Chapters 2 and 3. These problems are denoted by $Pm|\text{online}-\text{list}, m_j|C_{\max}$ and $1|\text{online}-\text{list}, p-\text{batch}, B<n|C_{\max}$, respectively. For the first problem the α-field contains Pm, stating that it is a scheduling problem with m parallel machines.

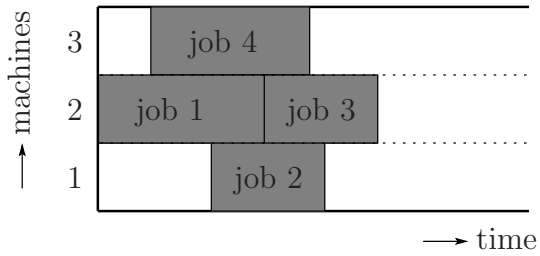


Figure 1.1: Example Gantt-chart

The β-field contains the term online − list to indicate that the jobs arrive online one by one (see Section 1.4) and m_j to indicate that the number of machines required for processing a job can be more than one and different for each job. The objective function, in the γ-field, is to minimize $C_{\max}$. This is to minimize the makespan, the completion time of the last completing job. For the second problem the value 1 in the α-field indicates that there is only one machine available to process the jobs. The terms $p-$batch and $B < n$ state that up to B jobs can be processed simultaneously in parallel to each other in a single batch and that B is smaller than the total number of jobs n (see Section 1.6.2).

1.3 Complexity of optimization problems

The optimization problems faced in Operations Research are mostly combinatorial, meaning that the solution has to be selected from a (possibly infinite) discrete set of possible solutions. In this section we formalize the concept of complexity of optimization problems. The following is based on [74].

A combinatorial optimization problem is stated as: minimize $f(x)$ with $x \in X$, where X is a discrete set derived from the input of the problem and where $f : X \to \mathbb{R}$. As an example consider the following parallel machine scheduling problem: *Given a number of identical machines and a set of jobs, find a schedule that minimizes the makespan.* The input is given by the number of machines and the set of jobs with their individual processing times. The solution set X is the set of all possible assignments of the job set to the machines. The objective function f is the makespan, the completion time of the job that completes last. An assignment to the machines is sufficient to specify a solution, since the order in which the jobs are processed on a machine does not influence the objective function.

Contrary to an optimization problem a decision problem can be answered by 'yes' or 'no'. Note that each optimization problem can be transformed into a decision problem: *For a given value r, is there an $x \in X$ such that $f(x) \leq r$?*

Complexity theory classifies decision problems with respect to the computational

effort needed to answer them. A decision problem belongs to the class NP if for a positive answer there exists a certificate such that the correctness of the positive answer can be checked in polynomial time using this certificate, i.e. the time required to check the certificate is bounded by a polynomial in the input size. The letters NP stand for non-deterministically polynomial, meaning that guessing the correct answer and a certificate leads to a polynomial time algorithm. Consider the parallel machine scheduling problem we introduced before. If for a given bound r on the makespan the answer is 'yes', then the corresponding assignment of the job set to the machines can serve as a certificate. Checking this certificate can be done in polynomial time in the input size, simply by adding the processing times of the jobs assigned to each of the machines and checking if these sums do not exceed r.

In the class P are the decision problems that can be solved in polynomial time. A problem is polynomial-time solvable if there exists an algorithm that decides for every possible input in polynomial time in the input size whether the answer is 'yes' or 'no' .

Obviously, we have P $\subseteq$ NP. For an instance of a problem in P with a positive answer, we can reconstruct the positive answer with the polynomial time algorithm, i.e. we check an empty certificate. Although it is unknown if there is a difference between the classes P and NP, it is widely believed that P $\neq$ NP. Within the class NP, the so-called NP-complete decision problems play an important role. These are the problems in NP that are at least as difficult as all other problems in NP, in the sense that every problem in NP can be reduced to them. Reducing means rewriting one problem in terms of the other problem in polynomial time. Finding a polynomial time algorithm for an NP-complete problem implies a polynomial time algorithm for all problems in NP, thus, proving that P = NP.

An optimization problem is called NP-hard if its corresponding decision problem is NP-complete.

Example. *Scheduling two parallel machines to minimize the makespan is NP-hard: We are given* 2 *machines and* n *jobs, where job* j *has processing time* p_j*. To see that this problem is NP-hard, let the corresponding decision problem be: Does there exist a schedule with makespan* $\leq \frac{1}{2}\sum_{j=1}^{n} p_j$*? It is known that the* PARTITION *problem is NP-complete [27]. This problem is formulated as follows: Given a finite set of numbers* $\{a_1, \ldots, a_t\}$*,* $a_i \in \mathbb{Z}^+$*, can the index set* $\{1, \ldots, t\}$ *be partitioned into* S_1 *and* S_2 *such that* $\sum_{i \in S_1} a_i = \sum_{i \in S_2} a_i$*? By letting* n *equal* t *and the processing times* p_i *equal* a_i*, we have reduced a general instance of the* PARTITION *problem to a specific instance of the scheduling problem, i.e. the answer is 'yes' for the scheduling instance if and only if the answer is 'yes' for the* PARTITION *instance. Thus, the decision version of minimizing the makespan on two parallel machines is at least as hard as* PARTITION *and therefore* NP*-complete. So, the optimization version of minimizing the makespan on two parallel machines is* NP*-hard.*

Under the assumption that P $\neq$ NP there exist no polynomial time algorithm for an NP-hard optimization problem. Trying to solve large instances of an NP-hard

problem by considering all possible solutions, total enumeration, takes too much time. So, the best one can hope for is near optimal solutions found by, for example, approximation schemes, constructive heuristics or local search methods [56].

This section gives only a short (informal) introduction to a much more elaborate field of theoretical computer science. There are many more complexity classes, some with only subtle differences like the difference between strongly and weakly NP-hard. The interested reader is referred to [27, 74] for more details.

1.4 Online optimization problems

For optimization problems it is often assumed that all information about the problem is known beforehand. However, this assumption is not always valid. Dropping this assumption gave cause to study so-called online optimization problems, problems where information is revealed piece by piece to the decision maker [6, 24]. The Ski-Rental Problem is a classical example of such an online optimization problem.

Example. *The Ski-Rental Problem: A girl goes skiing for a number of days, she does not know how many days she is going to ski, she might suddenly change her mind and switch to snow boarding. Assume that when she stops skiing she will never ski again and the value of used skis is zero. The cost to rent skis for a day is* 10 *Euros, the cost to buy the skis is* 100 *Euros. What should she do to minimize the expenditure?*

In online scheduling a sequence of jobs $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n)$ is revealed step by step. An online algorithm has to deal with the jobs before knowing the entire sequence of jobs. The characteristics of future jobs, for example their processing times, remain unknown until revealed. Even the length of the sequence is unknown to the online algorithm. Solving the problem with all information beforehand available is called the offline problem and leads to the optimal offline solution (as in Section 1.3). Not knowing the entire sequence of jobs makes it in general impossible for the online algorithm to construct an optimal solution.

There are several online paradigms, each with their own way of revealing the sequence of jobs, see e.g. [70].

The online-list model. The jobs of the sequence are revealed one by one to the online algorithm. As soon as a job with all its characteristics is revealed, the online algorithm has to irrevocably schedule this job, i.e. determine the start time of this job. After this, the next job is revealed. The online algorithm is free to choose any available time slot for processing the job. As a consequence, the start time of the current job can be later than that of successive jobs in the sequence. The offline solution is not restricted in any way by the order in which the jobs are presented.

The online-time model. The jobs of the sequence are revealed over time. Typically jobs have a release date and they are revealed to the online algorithm at their release date. The online algorithm must base its behavior at time t on the jobs

released before t, i.e. at time t the constructed schedule is already executed up to time t and can therefore not be changed. The online algorithm is not restricted to the order in which the jobs are revealed, i.e. it can delay a jobs and take scheduling decisions for later released jobs before scheduling this job. In the offline problem all job characteristics, including release dates, are known in advance, and an offline algorithm only has to make sure that no job is scheduled to start before its release date. There are two variants of the online-time model. If the processing time of a job becomes known to the online algorithm at the release date of the job, then the model is called clairvoyant. If the processing time is unknown until the job actually finished processing, the model is called non-clairvoyant.

The above described paradigms are not all the paradigms considered in the literature, but are by far the most commonly studied. For example, in other models the jobs arrive according to there precedence relations, i.e. a job becomes known if the processing of all its predecessors is completed, or jobs have to be scheduled in a predefined interval, i.e. jobs are either scheduled in this predefined interval or rejected from the schedule.

An online problem is called semi-online if there is some a priori knowledge about the input sequences. For example, the jobs in the sequence are known to be ordered by their processing times, the processing times are restricted, or the optimal offline objective value is known.

To evaluate the performance of an online algorithm competitive analysis is used [76]. Let $C_A(\sigma)$ denote the objective value of the schedule produced by an online Algorithm A and let $C^*(\sigma)$ denote the objective value of the optimal offline schedule, for a sequence σ.

Definition 1.1. *An online algorithm A is said to be ρ-competitive if $C_A(\sigma) \leq \rho \cdot C^*(\sigma) + \alpha$ for any sequence σ and a fixed constant α.*

For most online scheduling problems, especially for all that are considered in this thesis, the additive term α can be removed since by scaling the processing times of the jobs the objective value can be made arbitrary large compared to α.

The concept behind the competitive ratio can be seen as the analysis of a two-person zero-sum game. On the one hand we have the online player, playing the online algorithm, and on the other hand we have the adversary constructing and revealing the job sequence. The objective of the online player is to minimize the competitive ratio, while the adversary strives to maximize the competitive ratio. This game has an equilibrium, the value of the game, which is called the competitive ratio of the online problem, i.e. $\min_A \sup_\sigma \{C_A(\sigma)/C^*(\sigma)\}$. So, whenever we want to show a lower bound on the competitive ratio we place ourself in the position of the adversary and construct a set of sequences for which no online player can achieve a good competitive ratio. If we want to show an upper bound on the competitive ratio we have to design an algorithm which has a guaranteed performance no matter what instance the adversary presents us with. An online algorithm is called optimal if it attains the best possible competitive ratio. See [6] or more information on the game

theoretic foundations of competitive analysis.

Example. *The Ski-Rental Problem (cont.): The optimal offline algorithm for the Ski-Rental Problem will buy the skis at day one if the girl is going to ski for* 10 *days or more, and will rent otherwise. Let* N *be the number of days the girl goes skiing, this number is unknown to the online algorithm until the day the girl decides to stop. The offline cost equals* $\min\{100, 10 \cdot N\}$.

A generic online algorithm rents the skis for the first i *days and buys them on day* $i+1$. *The resulting cost are* $10 \cdot N$ *if* $i \geq N$ *and* $10 \cdot i + 100$ *if* $i < N$. *To minimize the competitive ratio, the girl should buy the skis after* 9 *days of renting, i.e.* $i = 9$. *This policy is* 1.9*-competitive and optimal. This can be seen by considering the situation when the girl buys the skis on the last day, i.e. the adversary lets* $N = i + 1$. *If either* $i < 9$ *or* $i > 9$ *the competitive ratio is larger than* 1.9.

The scheduling problem mentioned earlier in Section 1.3 of minimizing the makespan on parallel machines is one of the classical machine scheduling problems. This problem was already studied in the '60s by Graham [28, 29]. In these papers, Graham presents the analysis of his famous list scheduling algorithm. The jobs are in a list and the algorithm takes the next job from the list and schedules it as early as possible on the available machines, i.e. it schedules the jobs greedily. This algorithm is an online algorithm, because when a job is scheduled the remainder of the job list is not considered. The list scheduling algorithm is probably the first online algorithm in the field of scheduling, although back at that time the concept of online algorithms was unknown. To illustrate the competitive analysis behind this problem consider the following example.

Example. *Scheduling parallel machines to minimize the makespan: We are give* m *machines and a sequence* σ *of* n *jobs, where job* j *has processing time* p_j. *The objective is to minimize the makespan of the schedule. The list scheduling algorithm includes the jobs one by one as given in the sequence, each one as early as possible in the schedule. It is straightforward to show that the competitive ratio of the list scheduling algorithm is* $2 - \frac{1}{m}$ *[28].*

To prove that $2 - \frac{1}{m}$ *is an upper bound on the competitive ratio of the list scheduling algorithm, assume w.l.o.g. that job* n *is the last completing job and it starts at time* s. *By definition of the algorithm, all machines are busy in* $[0, s]$ *and job* n *is not processed in this interval. The optimal offline makespan for processing all* n *jobs is at best* $\frac{1}{m} \sum_{j=1}^{n} p_j$ *and the optimal offline makespan is also no less than* p_n. *So,*

$$C_{\text{LIST}}(\sigma) = s + p_n \leq \frac{1}{m} \sum_{j=1}^{n-1} p_j + p_n \leq \left(2 - \frac{1}{m}\right) C^*(\sigma) \ .$$

To prove that $2 - \frac{1}{m}$ *is a lower bound on the competitive ratio of the list scheduling algorithm, consider the following sequence of jobs: There are* $m(m-1)+1$ *jobs in the list, the first* $m(m-1)$ *have processing time* 1 *and the last job has processing time*

m. The optimal offline solution has makespan m, and the makespan of the schedule created by list scheduling is $2m - 1$.

Both results together show that the bound $2 - \frac{1}{m}$ is tight for list scheduling.

Competitive analysis is truly a worst-case analysis on the performance of an online algorithm. The malicious adversary knows exactly the behavior of the online algorithm and this leads to quite strong lower bounds on the performance. To overcome the "unrealistic" power of the adversary some models allow randomization in the online algorithm. The adversary knows the random process the online algorithm uses but does not know the outcome of that process. Such an oblivious adversary has to commit to a sequence before the online algorithm starts. A randomized algorithm is ρ-competitive if its solution is in expectation within a factor of ρ times the optimal offline solution. For more on randomization and examples, see [6, 70].

In the literature there are other ways discussed to make the game more fair toward the online player. For example, by resource augmentations the online algorithm has an faster machine or an additional machine to process the jobs [70].

1.5 Project scheduling

The modeling of more general scheduling problems is done in what is called project scheduling. One of the basic problems in this area is the Resource-Constrained Project Scheduling Problem (RCPSP). It is formulated as follows. A project consists of a set of jobs $\{0, 1, \ldots, n, n+1\}$ and a set of resources $\{1, \ldots, K\}$, where jobs 0 and $n + 1$ are dummy jobs representing the start and completion of the project, respectively. Each job j has a processing time of p_j, and during its non-preemptive processing it requires q_{jk} units of resource k. Each resource k has only a limited availability R_k. Furthermore, there are temporal relations between the jobs; job i is a predecessor of job j if job i is required to complete before the start of job j. These precedence relations can be expressed by an acyclic network on the jobs. A project instance can therefore be represented by a so-called Activity-on-Node network, see Figure 1.2 for an example project and the Gantt-chart of a corresponding solution.

The project scheduling model is richer than the machine scheduling model in the sense that jobs can require more than one resource for their processing, i.e. they require a specific amount of each of the available resources during their processing. Therefore, machine scheduling problems are special cases of project scheduling problems. Due to the richness of the model it captures many aspects of practical scheduling problems that are found in diverse industries such as construction engineering, make-to-order assembly, software development, etc. [9].

The richness of the model and the introduction of many real life aspects makes these project scheduling problems hard to solve, not only theoretically. This hardness is reflected in the inapproximability of the resource-constrained project scheduling problem, as there is a reduction from vertex coloring in graphs [73]. While vertex coloring is not approximable to within any constant bound, many machine scheduling

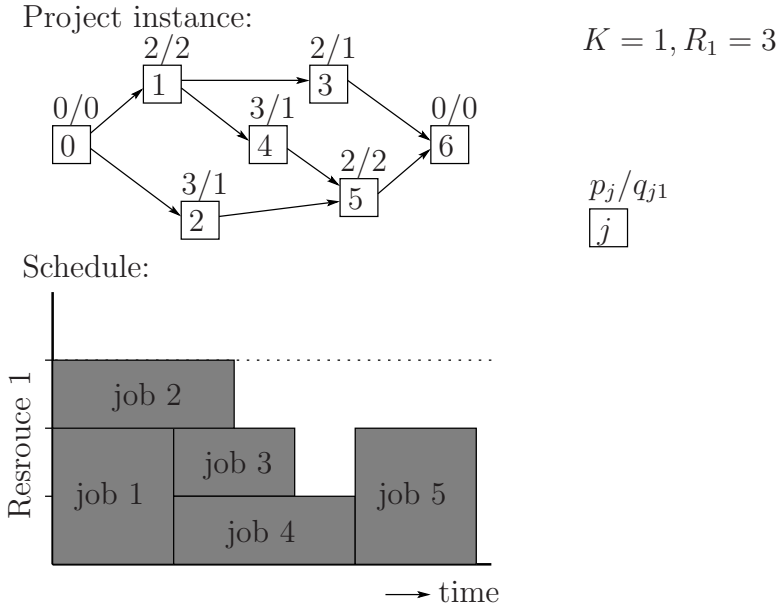


Figure 1.2: Example Project

problems are. Due to the complexity of these project scheduling problem exact methods fail on large instances, but there are many constructive heuristics and local search techniques available to find good quality solutions. For an overview of these solution approaches the reader is referred to [46, 49, 50]. For further information about the different model aspects available in project scheduling we refer to [9, 18, 36].

This thesis is concerned with the Time-Constrained Project Scheduling Problem (TCPSP). Contrary to the resource-constrained problem, in the time-constrained variant the jobs have deadlines that are strict. In order to meet these deadlines different ways to speed up the project are given, e.g. by working in overtime or hiring additional resource capacity. These options are costly but in practice often not avoidable. The question arises how much, when, and what kind of extra capacity should be employed to meet the deadlines against minimum cost. Although in practice deadlines occur often in projects, the Time-Constrained Project Scheduling Problem has been considered less frequently in the literature.

1.6 Specific parallel resource requirements

This thesis mainly concerns scheduling problems that contain some specific form of parallel resource requirement. In this section we introduce the different forms of

parallelism that are considered in this thesis.

1.6.1 Parallel jobs

As an extension of classical machine scheduling problems, parallel jobs are jobs that require not 1 machine for processing but a number of machines simultaneously. A job j is therefore not only characterized by its processing time p_j but also by the number of machines m_j it requires for processing.

Scheduling problems with parallel jobs can be found in applications like parallel computing and memory allocation. In parallel computing a large number of processors work together to process the total work load. To obtain a high performance in these systems it is crucial to divide the jobs in the right way among the machines. Whenever a job is divided over a number of processors, communication between the processors is in some models necessary. In these cases there is an underlying network facilitating this communication. If this network is a complete graph we are dealing with a PRAM (Parallel Random Accessing Machine) but the network topology can also be a line or a mesh [75].

In the literature the concept of parallel jobs is known by many different names, such as *parallel tasks*, *parallelizable tasks*, *multiprocessor tasks*, *multiple-job-on-one-processor*, and *1-job-on-r-processors*. Due to this diversity, in some literature the machine requirement m_j of a job is called the width or the size of a job, and in stead of m_j sometimes the term $size_j$ or simply s_j is used to denote the parallel machine requirement of job j.

There are different models for the flexibility of parallel jobs. For the problem described above, which we deal with in this thesis, the jobs are called rigid. In the rigid model the number of machines needed for each job is precisely given. Jobs are called moldable if the scheduler can determine (within limits) the number of machines assigned to each job. The processing time reduces when more machines are assigned. For malleable jobs the number of machines assigned to a job may even change during the processing of the job. For an extensive overview of the results for the different flexibility models and network topologies, we refer to [22].

1.6.2 Batching machine

Batching machines are machines that process sets of jobs simultaneously. These sets are so-called batches. The machine receives, processes and releases the jobs of a batch together. In this thesis, we consider what is called parallel batching. The processing of the jobs in a batch starts and end with the processing of the batch, even if the required processing time of a job is less. Therefore, the processing time of a batch is given by the maximum processing time of the jobs in that batch.

This model of batching finds applications in, for example, scheduling burn-in ovens for circuit boards. There is one oven, the batching machine, that can accommodate a large number of circuit boards that need to be heated. A circuit board

needs to be heated for a minimum amount of time, and staying longer in the oven than required is no problem. However, during the heating process we cannot remove parts of the ovens content.

For more on parallel batching problems and other types of batching, see [8].

1.6.3 Adjacent resources

In the setting of project scheduling, we consider adjacent resources. An adjacent resource is a special type of resource of which the resource units are somehow topologically ordered, e.g. they are ordered on a line. When units are assigned to a job, these units have to be next to each other; they have to be adjacent. Reassignment of resource units during the jobs processing is not allowed.

These type of adjacency requirements are found in many practical settings. If the adjacent resource is the only resource and its topology is a line, the scheduling problem can be seen as a two dimensional packing problem. Examples of this can be found in literature on berth allocation at container terminals, see e.g. [31, 59], where boats have to be assigned to a certain contiguous length of the quay. Other examples are found in reconfigurable embedded platforms, see e.g. [23, 77], and check-in desks at airports, see [20].

Motivated by the occurrence of adjacent resources in real life problems, we additionally assume that the adjacent resource is not required only by a single job but by groups of jobs. As soon as the processing of a job of such a job group starts the assigned adjacent resource units are occupied, and they are not released before all jobs of that group are completed. One practical application is from the ship building industry and illustrates the adjacency requirements. In this problem the docks form 1-dimensional adjacent resources, and all jobs related to building a single ship form a job group. Clearly, the part of the dock assigned to one ship has to satisfy the adjacency requirement. As soon as the construction of a ship starts, the assigned part of the dock is occupied until the construction is finished and the ship is removed from the dock. Removal or repositioning of a partially assembled ship is in practice too cumbersome and time consuming, and therefore not allowed.

1.7 Outline of the thesis

In the first part of this thesis, Chapters 2 and 3, we study online machine scheduling problems. These chapters are all about proving lower and upper bounds on the competitive ratios. In the second part, Chapters 4 and 5, we focus on the Time-Constrained Project Scheduling Problem with Adjacent Resources. The design of new heuristics are the main results in these chapters. Each chapter is intended to be self contained such that it can be read separately. More specifically, the contents of the chapters is the following.

In Chapter 2 we consider the online-list scheduling of parallel jobs $P|\text{online} - \text{list}, m_j|C_{\max}$ and many of its special cases. This chapter is based on joint work with

Johann Hurink [39, 40, 41]. One of the main results is the 6.6623-competitive algorithm which also applies to the *online orthogonal strip packing problem* (see Section 2.3). Another important result is the tight lower bound of 2 on the competitive ratio of the problem with two machines, $P2|\text{online}-\text{list}, m_j|C_{\max}$ (see Section 2.4).

In Chapter 3 we consider the problem of online-list scheduling of a batching machine, $1|\text{online}-\text{list}, p-\text{batch}, B<n|C_{\max}$. This chapter is based on the joint work with Deshi Ye and Guochuan Zhang [65]. The chapter gives a complete classification of the tractability of this online problem, i.e. with respect to the competitive ratio we derive an optimal algorithm for any capacity B of the batching machine. The novelty of this chapter lies in the lower bound proofs. The design of the optimal online algorithms is based on the well known "doubling" strategy.

In Chapter 4 we deal with the Time-Constrained Project Scheduling Problem. This chapter is based on the joint work with Tom Guldemond, Johann Hurink and Marco Schutten [34]. This chapter proposes a new approach for scheduling with strict deadlines and applies this approach to the Time-Constrained Project Scheduling Problem. To be able to meet these deadlines, it is possible to work in overtime or hire additional capacity in regular time or overtime. For this problem, we develop a two stage heuristic. The key of the approach lies in the first stage in which we construct partial schedules. In these partial schedules, jobs may be scheduled for a shorter duration than required. The second stage uses an ILP formulation of the problem to turn a partial schedule into a feasible schedule, and to perform a neighborhood search. The developed heuristic is quite flexible and, therefore, suitable for practice. We present experimental results on modified RCPSP benchmark instances. The two stage heuristic solves many instances to optimality, and lead only to a small rise in cost if we substantially decrease the deadline.

In Chapter 5 we deal with the same problem as in Chapter 4 with the addition of an adjacent resource. This chapter is based on the joint work with Johann Hurink, Leendert Kok and Marco Schutten [38]. For adjacent resources the resource units are ordered and the units assigned to a job have to be adjacent. On top of that, adjacent resources are not required by single jobs, but by job groups. As soon as a job of such a group starts, the adjacent resource units are occupied, and they are not released before all jobs of that group are completed. The developed decomposition method separates the adjacent resource assignment from the rest of the scheduling problem. Test results demonstrate the applicability of the decomposition method. The presented decomposition method forms a first promising approach for the TCPSP with adjacent resources and may form a good basis to develop more elaborated methods.

Publications underlying this thesis

[34] T.A. Guldemond, J.L. Hurink, J.J. Paulus, and J.M.J. Schutten. Time-constrained project scheduling. *Journal of Scheduling*, 11(2):137-148, 2008.

[38] J.L. Hurink, A.L. Kok, J.J. Paulus, and J.M.J. Schutten. Time-constrained project scheduling with adjacent resources. Working paper 261, Beta Research School for Operations Management and Logistics, Eindhoven, The Netherlands, 2008.

[39] J.L. Hurink and J.J. Paulus. Special cases of online parallel job scheduling. Working paper 235, Beta Research School for Operations Management and Logistics, Eindhoven, The Netherlands, 2007.

[40] J.L. Hurink and J.J. Paulus. Online algorithm for parallel job scheduling and strip packing. *Lecture Notes of Computer Science (WAOA 2007)*, 4927:67-74, 2008.

[41] J.L. Hurink and J.J. Paulus. Online scheduling of parallel jobs on two machines is 2-competitive. *Operations Research Letters*, 36(1):51-56, 2008.

[65] J.J. Paulus, D. Ye, and G. Zhang. Optimal online-list batch scheduling. Working paper 260, Beta Research School for Operations Management and Logistics, Eindhoven, The Netherlands, 2008.

Part I

Online Scheduling

2 Parallel jobs online

Parallel jobs are jobs which require processing on a number of machines simultaneously, this in contrast to traditional machine scheduling where each job requires exactly one machine for processing. Parallel jobs are, for example, encountered in memory allocation and parallel processing in computers [11].

The online parallel job scheduling problem we study in this chapter is formally stated as follows.

Jobs from a sequence $\sigma = (1, 2, ..., n)$ are presented one by one to the decision maker. Each job j is characterized by its processing time p_j and the number of machines m_j, out of the available m machines, which are simultaneously required for its processing. As soon as a job becomes known, it has to be scheduled irrevocably (i.e. its start time has to be set) without knowledge of successive jobs. Preemption is not allowed and the objective is to minimize the makespan.

Using the three field notation for scheduling problems [30], this problem is denoted by $P|\text{online} - \text{list}, m_j|C_{\max}$, see also [42, 70].

There is a great deal of similarity between $P|\text{online} - \text{list}, m_j|C_{\max}$ and the online orthogonal strip packing problem [7]. The orthogonal strip packing problem is a two-dimensional packing problem. Without rotation rectangles have to be packed on a strip with fixed width and unbounded height. The objective is to minimize the height of the strip in which the rectangles are packed. In the online setting rectangles are presented one by one and have to be assigned without knowledge of successive rectangles. To see the similarity, let each machine correspond to one unit of the width of the strip, and time to the height of the strip. The width of a rectangle j corresponds to the machine requirement of job j and its height to the processing time. Minimizing the height of the strip used is equivalent to minimizing the makespan of the machine scheduling problem. However, the difference lies in the choice of machines. In $P|\text{online} - \text{list}, m_j|C_{\max}$ any m_j machines suffice for job j, where in the

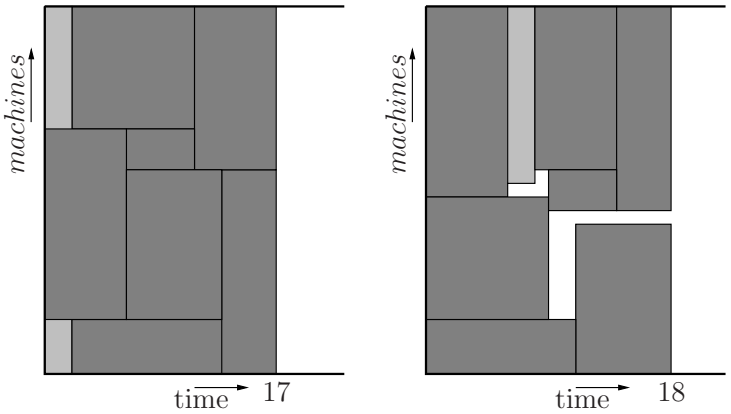
Job j	p_j	m_j
1	2	11
2	6	12
3	4	13
4	8	7
5	5	3
6	11	4
7	7	9
8	6	10

Table 2.1: Set of jobs for counterexample.

strip packing problem rectangle j cannot be split up into several rectangles together having width m_j. Therefore, algorithms for strip packing can be used for parallel job scheduling, but in general not the other way around. The following counterexample is taken from [78]. Consider a set of 8 jobs with characteristics as given in Table 2.1. Suppose there are 23 machines available to process these jobs. In the setting of *parallel jobs* we can complete the processing of the jobs in 17 time units, see Figure 2.1a). In the setting of *strip packing* we need 18 time units, see Figure 2.1b). The job that is split along non-adjacent machines, Job 1, has a lighter shade in these figures. This example shows that, due to the additional adjacency requirement in strip packing, a solution of a parallel job problem cannot always be transformed into a solution of the corresponding strip packing problem.

In Chapter 5 we deal with adjacent resources, the difference between adjacent resources and renewable resources is analogous to the difference between strip packing and parallel jobs.

The current state of the research on problem $P|\text{online}-\text{list}, m_j|C_{\max}$ is summarized in Table 2.2. The first online algorithm for online parallel job scheduling with a constant competitive ratio is presented in [42] and is 12-competitive. In [82], an improvement to a 7-competitive algorithm is given. This *dynamic waiting algorithm* schedules jobs with a small machine requirement greedily and delays the jobs with a large machine requirement. For the strip packing problem in [2] a 6.99-competitive online algorithm is given under the assumption that jobs have a processing time of at most 1. This *shelf algorithm* groups rectangles of similar height together. In this chapter we improve these results by presenting a 6.6623-competitive algorithm which applies to both the parallel job scheduling problem and the orthogonal strip packing problem. The best known lower bound on the competitive ratio for $P|\text{online}-\text{list}, m_j|C_{\max}$ is a lower bound of 2 from the strip packing problem [7], which applies directly to the parallel job scheduling problem with $m \geq 3$. We derive new lower bounds for $Pm|\text{online}-\text{list}, m_j|C_{\max}$ by means of an ILP formulation. This results in a lower bound of 2.43 on the competitive ratio for any online algorithm for $P|\text{online}-\text{list}, m_j|C_{\max}$.



a) as parallel jobs b) as strip packing

Figure 2.1: Schedules for parallel jobs and strip packing.

For the special case of the scheduling problem with two machines we improve, in Section 2.4, the previously best known lower bound of $1 + \sqrt{2/3}$, from [10], to a tight lower bound of 2. As a result we get that the greedy algorithm is optimal for $m = 2$. For the case with three machines, until now, the best known algorithm is the 3-competitive greedy algorithm. In Section 2.5 we improve this by presenting a 2.8-competitive algorithm for the 3 machine case.

Section 2.6 contains new results for the semi-online version of the problem. An overview of the relevant literature on semi-online scheduling of parallel jobs is also given there.

$P\|\text{online} - \text{list}, m_j\|C_{\max}$				
Model	Lower Bound		Upper Bound	
-	2.43,	[Sec. 2.3.2]	6.6623,	[Sec. 2.3.1]
$m = 2$	2,	[Sec. 2.4]	2,	[Greedy]
$m = 3$	2,	[7]	2.8,	[Sec. 2.5]
$3 \leq m \leq 6$	2,	[7]	m,	[Greedy]

Table 2.2: Results for $P|\text{online} - \text{list}, m_j|C_{\max}$

2.1 Bounding the offline solution

To be able to derive an upper bound on the competitive ratio of an online algorithm, we have to compare the online solutions to the optimal offline solutions. However,

in general this is very difficult. Therefore, we do not compare the online solutions to the actual optimal offline solutions but to lower bounds on these values.

Given a sequence of jobs $\sigma = (1, 2, ..., n)$ we denote the optimal offline makespan by $C^*(\sigma)$. For this value we can derive two straightforward lower bounds. On the one hand, the optimal makespan is bounded by the length of the longest job in σ, i.e.

$$C^*(\sigma) \geq \max_{j \in \sigma}\{p_j\} \ . \tag{2.1}$$

We call (2.1) the *length bound*. On the other hand we have the total work load. The work load of a job j is given by $m_j \cdot p_j$. In an optimal offline solution the total work load is at best evenly divided over the m machines. Thus, we get

$$C^*(\sigma) \geq \frac{1}{m} \sum_{j \in \sigma} m_j \cdot p_j \ . \tag{2.2}$$

We call (2.2) the *load bound*.

Let $C_A(\sigma)$ denote the makespan of the online schedule created by online Algorithm A. For a collection of disjoint intervals X from $[0, C_A(\sigma)]$, we denote by $|X|$ the cumulative length of the intervals in X.

The following lemma follows directly from the above presented lower bounds on $C^*(\sigma)$.

Lemma 2.1. *If $[0, C_A(\sigma)]$ can be partitioned in two collections of disjoint intervals X and Y such that $|X| \leq x \cdot \max_{j \in \sigma}\{p_j\}$ and $|Y| \leq y \cdot \frac{1}{m} \sum_{j \in \sigma} m_j \cdot p_j$, then $C_A(\sigma) \leq (x + y) \cdot C^*(\sigma)$.* □

In the design of algorithms for $P|\text{online} - \text{list}, m_j|C_{\max}$ we aim to be able to apply Lemma 2.1 in a clever way in the competitive analysis. To put it differently, it is a key issue to find a good way to combine the length and load bound on $C^*(\sigma)$ to obtain a small value of $x + y$, the resulting competitive ratio.

The methods in Sections 2.2 and 2.3 rely solely on the two lower bounds mentioned above. For the special case with three machines in Section 2.5 and the semi-online case with non-increasing m_j in Section 2.6, the above bounds alone are not enough. To obtain the presented results, problem specific lower bounds are derived. For these problems we sharpen the length and load bound by analyzing the structure of the online and offline schedules.

2.2 Greedy algorithms

The most simple online algorithm one can think of for the considered problem is a greedy algorithm. A greedy algorithm schedules each job j at the earliest time t for which at any time in the interval $[t, t + p_j)$ at least m_j machines are available. Unfortunately, for the online scheduling of parallel jobs a greedy scheduling algorithm

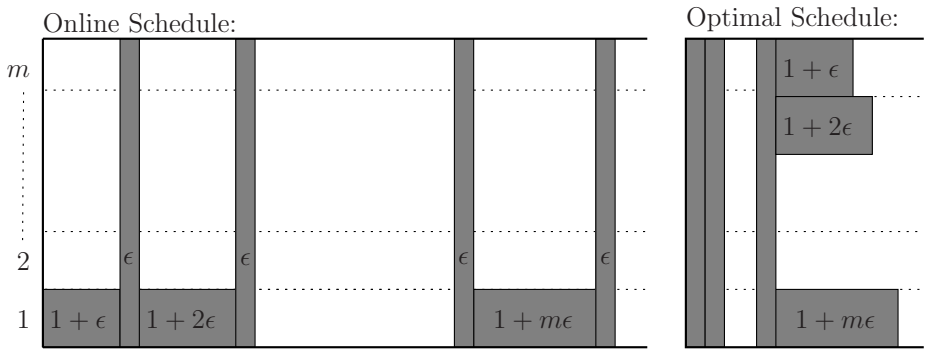


Figure 2.2: A greedy algorithm is no better than m-competitive.

has no constant competitive ratio. This is formalized in the following commonly known theorem.

Theorem 2.2. *The greedy algorithm is m-competitive for $Pm|\text{online} - \text{list}, m_j|C_{\max}$, and this ratio is tight.*

Proof. Consider the following instance with m machines and $2m$ jobs. Let the odd jobs have processing time $p_j = 1 + \frac{1}{2}\epsilon(j+1)$ and machine requirement $m_j = 1$ and let the even jobs have processing time $p_j = \epsilon$ and machine requirement $m_j = m$. The optimal offline schedule has makespan $1 + 2\epsilon m$ and the 'greedy schedule' has makespan $\epsilon m + \sum_{i=1}^{m}(1 + \epsilon i)$, see Figure 2.2. For ϵ going to 0, this results in a competitive ratio of m. On the other hand, as in the online schedule there is at any point in time at least one machine processing a job, the load bound (2.2) implies that the competitive ratio of the greedy algorithm is also at most m. □

Given Theorem 2.2, a greedy strategy does not seem to be a good one. Nevertheless, all algorithms presented in this chapter have a greedy component.

2.3 Arbitrary number of machines

This section discusses the online parallel job scheduling problem with an arbitrary number of machines. We present a 6.6623-competitive algorithm and discuss constructions leading to new lower bounds on the competitive ratio. This section concludes by showing that the presented algorithm is also applicable to the online orthogonal strip packing problem.

2.3.1 A 6.6623-competitive algorithm

We design an online algorithm for $P|\text{online} - \text{list}, m_j|C_{\max}$ such that the constructed schedules can be partitioned in an X and Y part, as in Lemma 2.1, resulting in a small $x + y$ value. To do this, we distinguish between two types of jobs; jobs with a large machine requirement and jobs that require only a few machines for processing. A job j is called *big* if it requires at least half of the machines, i.e. it has machine requirement $m_j \geq \lceil \frac{m}{2} \rceil$, and is called *small* otherwise. Furthermore, the small jobs are classified according to their length. A small job j belongs to job class J_k if $\beta^k \leq p_j < \beta^{k+1}$, where $\beta = 1 + \frac{\sqrt{10}}{5}$ (≈ 1.6325). Note that k may be negative. Similar classifications can be found in *Shelf Algorithms* for Strip Packing [2], which are applied to group rectangles of similar height.

In the schedules created by the online algorithm, big jobs are never scheduled in parallel to other jobs, and (where possible) small jobs are put in parallel to other small jobs of the same job class. The intuition behind the online algorithm is the following. Scheduling big jobs results in a relatively high average load in the corresponding intervals, and small jobs are either grouped together to a high average load or there is a small job with a relative long processing time. In the proof of 6.6623-competitiveness, the intervals with many small jobs, together with the intervals with big jobs will be compared to the load bound for $C^*(\sigma)$ (the Y part in Lemma 2.1), and the intervals with only a few small jobs are compared to the length bound for $C^*(\sigma)$ (the X part in Lemma 2.1).

In the following, a precise description of Algorithm PJ (*Parallel Jobs*) is given. The Algorithm PJ creates schedules where the small jobs of class J_k are either in a sparse interval S_k or in dense intervals D_k^i. With n_k we count the number of dense intervals created for job class J_k. All small jobs scheduled in an interval $[a, b)$ start at a. As a consequence, job j *fits* in interval $[a, b)$ if the machine requirement of the jobs already in $[a, b)$ plus m_j is at most m.

Algorithm PJ

Schedule job j as follows:
if job j is small, i.e. $m_j < \lceil \frac{m}{2} \rceil$, and belongs to job class J_k **then**
Try in the given order:

- Schedule job j in the first D_k^i interval where it fits.
- Schedule job j in S_k.
- Set $n_k := n_k + 1$ and let S_k become $D_k^{n_k}$. Create a new interval S_k at the end of the current schedule with length β^{k+1}. Schedule job j in S_k.

if job j is big, i.e. $m_j \geq \lceil \frac{m}{2} \rceil$ **then**
Schedule job j at the end of the current schedule.

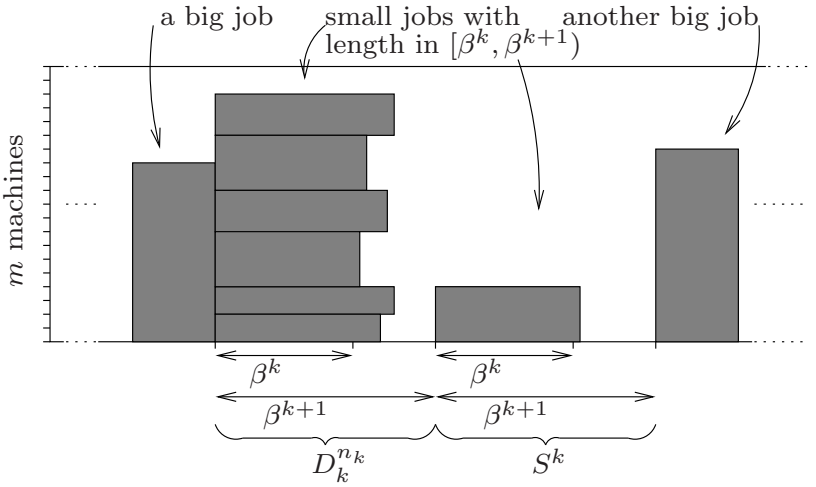


Figure 2.3: Part of a schedule created by Algorithm PJ.

The structure of a schedule created by Algorithm PJ is illustrated by Figure 2.3. It is important to note that at any time for each job class J_k there is at most one sparse interval S_k.

The way Algorithm PJ schedules jobs of a specific job class in the dense and sparse intervals strongly resembles bin packing [26]. We can look at each of these intervals as being bins in which we pack the jobs (items). Since all jobs are scheduled to start at the beginning of the interval only the machine requirement matters. In this way, the machine requirement of the job corresponds to the size of the item to be packed. The Algorithm PJ *packs* with a First-Fit strategy, i.e. a small job (an item) is scheduled (packed) in the first interval (bin) it fits.

To bound the competitive ratio of Algorithm PJ, we use the fact that the dense intervals D_k^i contain quite some load, since for each dense interval there is a small job that did not fit in this dense interval and had to be scheduled in a newly created sparse interval. In terms of bin packing we have the following lemma.

Lemma 2.3. *If items with size less than $\frac{1}{2}$ are packed First-Fit and this packing uses b bins, the total size of the items packed is at least $\frac{2(b-1)}{3}$.*

Proof. Consider an arbitrary sequence of items with size less than $\frac{1}{2}$ which result in the use of b bins by packing it with First-Fit. Let $\tilde{b}$ be the first bin which is filled less than $\frac{2}{3}$. By definition of First-Fit all items in successive bins have size at least $\frac{1}{3}$. This implies that all successive bins, except possibly the last, are filled for at least $\frac{2}{3}$. More precisely, they contain precisely two items with size larger than $\frac{1}{3}$. Thus, there are at most two bins with item size less than $\frac{2}{3}$, which are bin $\tilde{b}$ and

the last bin. However, the existence of $\tilde{b}$ implies that the total item size in the last bin and bin $\tilde{b}$ together is at least 1. So, the total size of the items packed is at least $\frac{2}{3}(b-2)+1 \geq \frac{2(b-1)}{3}$. If no $\tilde{b}$ exists or if $\tilde{b}$ is the last bin, the lemma trivially holds. □

Taking this bin packing view on the dense and sparse intervals we get the following.

Lemma 2.4. *The total work load in the dense and sparse intervals of the schedule created by Algorithm PJ, is at least $\frac{2m}{3\beta}$ times the length of all dense intervals.*

Proof. Consider all dense and sparse intervals in the schedule created by Algorithm PJ corresponding to one job class J_k. There are in total n_k dense intervals and 1 sparse interval, each with length β^{k+1}. By Lemma 2.3 and the definition of job classes, we know that the total work load of the jobs in job class J_k is at least $\frac{2}{3}n_k m\beta^k$, which equals $\frac{2m}{3\beta}$ times the length of all dense intervals of job class J_k. □

Using Lemma 2.4 we can connect the length of the online schedule with the load bound on the optimal offline schedule. This gives the necessary tools to prove the upper bound on the performance guarantee of online Algorithm PJ.

Theorem 2.5. *The competitive ratio of the Algorithm PJ is at most $\frac{7}{2}+\sqrt{10}$ ($\approx$ 6.6623).*

Proof. Let σ be an arbitrary sequence of jobs. We partition the schedule $[0, C_{PJ}(\sigma)]$ created by the online Algorithm PJ into three parts: The first part B consists of the intervals in which big jobs are scheduled, the second part D consists of the dense intervals, and finally the third part S contains the sparse intervals.

Since part B contains only jobs with machine requirement $m_j \geq \lceil \frac{m}{2} \rceil$, the total work load in B is at least $\frac{m}{2} \cdot |B|$. According to Lemma 2.4, the total work load in D and S is at least $\frac{2m}{3\beta} \cdot |D|$. Since this work load also has to be scheduled in the optimal offline solution, we get $\min\{\frac{m}{2}, \frac{2m}{3\beta}\} \cdot (|B|+|D|) \leq m \cdot C^*(\sigma)$. For $\beta \geq \frac{4}{3}$, this results in

$$|B| + |D| \leq \frac{3\beta}{2} \cdot C^*(\sigma) \ . \tag{2.3}$$

To simplify the arguments for bounding $|S|$, we normalize the processing times of the jobs such that J_0 is the smallest job class, i.e. the smallest processing time of the small jobs is between 1 and β. Then, $|S_k| = \beta^{k+1}$. Let $\bar{k}$ be the largest k for which there is a sparse interval in the online schedule. Since there is at most one sparse interval for each job class J_k, the length of S is bounded by

$$|S| \quad \leq \quad \sum_{k=0}^{\bar{k}} |S_k| = \sum_{k=0}^{\bar{k}} \beta^{k+1} = \frac{\beta^{\bar{k}+2} - \beta}{\beta - 1} \ .$$

On the other hand, since interval $S_{\bar{k}}$ is non empty, we know that there is a job in the sequence σ with processing time at least $\frac{|S_{\bar{k}}|}{\beta} = \beta^{\bar{k}}$. Thus, using the length bound we get

$$|S| \quad \leq \quad \frac{\beta^2}{\beta - 1} \cdot C^*(\sigma) \ . \tag{2.4}$$

Lemma 2.1, (2.3) and (2.4) lead to the following bound on the makespan of the schedule created by online algorithm PJ:

$$C_{PJ}(\sigma) \quad = \quad |B| + |D| + |S| \leq \left(\frac{3\beta}{2} + \frac{\beta^2}{\beta - 1} \right) \cdot C^*(\sigma) \ .$$

Choosing $\beta = 1 + \frac{\sqrt{10}}{5}$ (which is larger than $\frac{4}{3}$), Algorithm PJ has a competitive ratio of at most $\frac{7}{2} + \sqrt{10}$ (≈ 6.6623). □

It is interesting to note that for other values of β the analysis will only result in bounds worse than 6.6623. However, defining big jobs as jobs with machine requirement of at least $\lceil \alpha m \rceil$, results for all $\alpha \in [\frac{10}{3(5+\sqrt{10})}, \frac{1}{2}]$ ($\approx [.4084, 0.5]$) in 6.6623-competitiveness of PJ.

Note. *In the independent work of Ye et al. [80] the* 6.6623*-competitive algorithm is obtained in the setting of online orthogonal strip packing. They also show that the analysis is tight, i.e. there exists an instance for which Algorithm PJ is no better than* 6.6623*-competitive.*

2.3.2 Lower bounds by linear programming

The best known lower bound on the competitive ratio of any online algorithm for the considered problem is 2 [7]. The lower bound construction is given in the setting of online strip packing but can be applied to online parallel job scheduling with at least 3 machines as shown in the next theorem.

Theorem 2.6. *For $Pm|\text{online} - \text{list}, m_j|C_{\max}$, with $m \geq 3$ and $\delta > 0$, no $(2-\delta)$-competitive online algorithm for exists.*

Proof. We omit the details of the proof, and only give the instance construction used. Let s_i denote the start time of job i in the online schedule, and let ϵ be a small positive value. Job 1 has $m_1 = 1$ and processing time $p_1 = 1$. The second job has $m_2 = m$ and $p_2 = s_1 + \epsilon$. There is no other choice but to schedule the second job after the first. Now, job 3 has $m_3 = 1$ and $p_3 = s_2 + \epsilon$. Again, this job can only be scheduled at the end of the online schedule. Job 4 has $m_4 = m$ and $p_4 = \max\{s_1, s_2 - s_1 - 1, s_3 - s_2 - p_2\} + \epsilon$. This job has to be scheduled after job 3. Finally, job 5 has $m_5 = 1$ and $p_5 = s_4 - s_2 - p_1 + \epsilon$ and has to be scheduled at the end of the online schedule. The resulting online and offline schedules are illustrated in Figure 2.4. With ϵ small enough one can show that no online algorithm is $(2-\delta)$-competitive on this instance. □

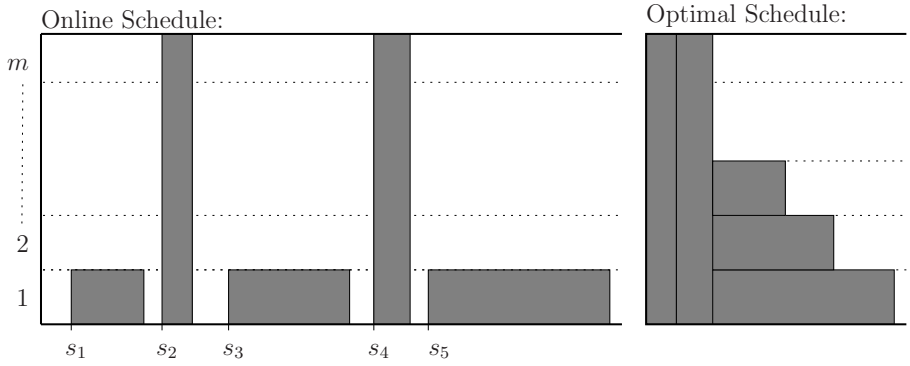


Figure 2.4: Lower bound constructing of Theorem 2.6.

In the following we extend the instance construction of Theorem 2.6 to obtain a lower bound of 2.43. Besides some concrete lower bounds, we also show that by constructing job sequences similar to the ones used in [42] and in the this section, no lower bound greater than 2.5 can be obtained. Thus, to get better lower bounds one has to consider completely different job sequences.

We define σ_{m-1} as the sequence of jobs $(p_0, q_1, p_1, q_2, p_2, \ldots, q_{m-1}, p_{m-1})$, where p_i (q_i) denotes a job with processing time p_i (q_i) and a machine requirement of 1 (m). The job lengths are defined as:

$$\begin{aligned} p_0 &= 1 \\ p_i &= x_{i-1} + p_{i-1} + y_i + \epsilon && \forall i \geq 1 \\ q_1 &= x_0 + \epsilon \\ q_i &= \max\{y_{i-1}, q_{i-1}, x_{i-1}\} + \epsilon && \forall i \geq 2 \ , \end{aligned}$$

where x_i and y_i are values given by delays the online algorithm has used for placing jobs p_i and q_i, respectively, and where ϵ is a small positive value. By definition of the job lengths, the jobs can only be scheduled in the order of the sequence σ_{m-1}. As a consequence, Figure 2.5 illustrates the structure of any online schedule for sequence σ_{m-1}. Due to this structure, any online algorithm for sequence σ_{m-1} can be described by the delays it incurs. An optimal schedule for σ_{m-1} is obtained by scheduling the jobs $p_0, ..., p_{m-1}$ parallel to each other on the m different machines, after a block containing the jobs $q_1, ..., q_{m-1}$. To simplify notation, for the remaining we let ϵ go to zero and omit it from the rest of the analysis.

This construction is somehow similar to the construction in [42]. The main difference is that in [42] only integer delays, processing times and starting times are considered, leading to a different definition of the processing times p_i and q_i, i.e. the additive term $+\epsilon$ is replaced by $+1$. As a consequence, the lower bound derived in

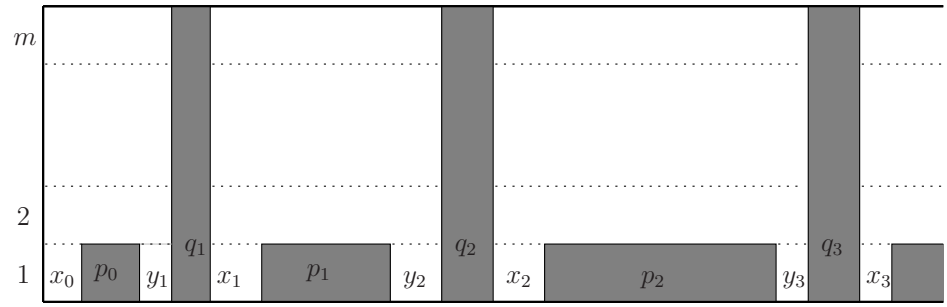


Figure 2.5: Structure of the online schedule for sequence σ_{m-1}.

[42] (a bound of 2.25) is not a valid lower bound for the general case with arbitrary processing times.

If an online algorithm is ρ-competitive for σ_{m-1}, the following linear inequalities have to be fulfilled.

$$x_0 + p_0 \leq \rho \cdot p_0 \tag{2.5}$$

$$x_0 + p_0 + \sum_{j=1}^{i}(y_j + q_j + x_j + p_j) \leq \rho \cdot (\sum_{j=1}^{i} q_j + p_i) \qquad \forall 1 \leq i \leq m-1 \tag{2.6}$$

$$\sum_{j=1}^{i}(y_j + q_j + x_{j-1} + p_{j-1}) \leq \rho \cdot (\sum_{j=1}^{i} q_j + p_{i-1}) \qquad \forall 1 \leq i \leq m-1 \tag{2.7}$$

Inequalities (2.5) and (2.6) state that the online solution is within a factor of ρ of the optimal, after scheduling job p_i. Inequality (2.7) states the same after scheduling job q_i.

Based on (2.5)-(2.7) it is possible to derive an ILP formulation in order to check whether a given value of ρ is a lower bound on the competitive ratio. We have to add to (2.5)-(2.7) the constraints that guarantee that the processing time p_i and q_i are according to sequence σ_{m-1}. Constraints (2.8)-(2.10) model the job lengths of the p-jobs and q_1. To model the lengths of the other q-jobs we employ a set of binary variables λ_i^y, λ_i^q and λ_i^x, where $\lambda_i^y = 0$ implies that $q_i = y_{i-1}$, $\lambda_i^q = 0$ that $q_i = q_{i-1}$ and $\lambda_i^x = 0$ that $q_i = x_{i-1}$. Constraints (2.11)-(2.13) guarantee that $q_i \geq \max\{y_{i-1}, q_{i-1}, x_{i-1}\}$ holds. Constraint (2.14) states that exactly one of λ_i^y, λ_i^q and λ_i^x equals 0 for all i. Together with constraints (2.15)-(2.17), where M is a sufficiently large constant, the equality $q_i = \max\{y_{i-1}, q_{i-1}, x_{i-1}\}$ is guaranteed.

m	2	3	4	5	10	20	30
LB	1.707	1.999	2.119	2.201	2.354	2.413	2.430

Table 2.3: Lower Bounds on the Competitive Ratio

$$p_0 = 1 \tag{2.8}$$
$$p_i = x_{i-1} + p_{i-1} + y_i \qquad \forall 1 \le i \le m-1 \tag{2.9}$$
$$q_1 = x_0 \tag{2.10}$$
$$y_{i-1} \le q_i \qquad \forall 2 \le i \le m-1 \tag{2.11}$$
$$q_{i-1} \le q_i \qquad \forall 2 \le i \le m-1 \tag{2.12}$$
$$x_{i-1} \le q_i \qquad \forall 2 \le i \le m-1 \tag{2.13}$$
$$\lambda_i^y + \lambda_i^q + \lambda_i^x = 2 \qquad \forall 2 \le i \le m-1 \tag{2.14}$$
$$q_i \le y_{i-1} + M \cdot \lambda_i^y \qquad \forall 2 \le i \le m-1 \tag{2.15}$$
$$q_i \le q_{i-1} + M \cdot \lambda_i^q \qquad \forall 2 \le i \le m-1 \tag{2.16}$$
$$q_i \le x_{i-1} + M \cdot \lambda_i^x \qquad \forall 2 \le i \le m-1 \tag{2.17}$$

In the above system of equations, the variables y_i, q_i, x_i, p_i are nonnegative and λ_i^y, λ_i^q and λ_i^x are binary variables.

Lemma 2.7. *If for a given m there exists no solution satisfying constraints (2.5)-(2.17), ρ is a lower bound on the competitive ratio of any online algorithm for $Pm|online-list, m_j|C_{max}$.*

Proof. Suppose there exists an ρ-competitive online algorithm. This algorithm will yield for the job sequence σ_{m-1} values of x_i and y_i such that constraints (2.5)-(2.17) are satisfied. □

Based on Lemma 2.7, we can obtain new lower bounds on the competitive ratio by checking infeasibility of the constraint set (2.5)-(2.17). Since the system (2.5)-(2.17) is linear, for given values of ρ and m, we may use an ILP solver (e.g. CPLEX) to check whether ρ is a lower bound by trying to find a feasible setting of the x_i and y_i variables with respect to (2.5)-(2.17). By employing binary search on ρ, we get the new lower bounds displayed in Table 2.3. Note that a lower bound obtained for the m machine case is also a lower bound for the $m+1$ machine case. As a result, the following theorem holds.

Theorem 2.8. *No online algorithm for $P|online-list, m_j|C_{max}$ can have a competitive ratio less than* 2.43. □

Since σ_{m-1} contains exactly m jobs with a machine requirement of 1, these jobs can be scheduled parallel to each other on the m different machines in the offline

solution. Let σ_n be a job sequence defined in the same manner as σ_{m-1}, but now with $n \geq m$. For such a sequence, one might expect a more efficient packing of the p-jobs in the optimal offline solution. Adapting the ILP formulation for such a longer sequences leads to a much more involved formulation, while the lower bound increases only slightly. The following theorem explains why there is only such a slight increase.

Theorem 2.9. *With job sequence σ_n, no lower bound on the competitive ratio larger than* 2.5 *can be obtained for $Pm|online-list, m_j|C_{max}$.*

Proof. Consider an online algorithm which chooses $x_i = p_i$ and $y_i = 0$ for all i. As a consequence, $p_i = 2 \cdot p_{i-1}$ and $q_i = x_{i-1} = p_{i-1}$. This results in an online schedule with makespan

$$2 \cdot \sum_{j=0}^{i} p_j + \sum_{j=1}^{i} q_j = 3 \cdot \sum_{j=0}^{i-1} p_j + 2 \cdot p_i = 5 \cdot \sum_{j=0}^{i-1} p_j + 2$$

after scheduling job p_i, and a makespan of

$$2 \cdot \sum_{j=0}^{i-1} p_j + \sum_{j=1}^{i} q_j = 3 \cdot \sum_{j=0}^{i-1} p_j = 6 \cdot \sum_{j=0}^{i-2} p_j + 3$$

after scheduling job q_i.

Since the p-jobs grow with a factor of 2, the makespan of the optimal offline schedule equals $\sum_{j=1}^{i} q_j + p_i = 2 \cdot \sum_{j=0}^{i-1} p_j + 1$ after job p_i and $\sum_{j=1}^{i} q_j + p_{i-1} = \sum_{j=0}^{i-1} p_j + p_{i-1} = 3 \cdot \sum_{j=0}^{i-2} p_j + 2$ after job q_i.

Both after scheduling p_i and q_i the competitive ratio of the proposed online algorithm is less or equal to 2.5. So, with this type of job sequence no lower bound on the competitive ratio larger than 2.5 can be proven for $Pm|online-list, m_j|C_{max}$. □

Note that even when the length of the p-jobs is defined such that $p_i \geq x_{i-1} + p_{i-1} + y_i$, Theorem 2.9 holds. Given all the above, we believe that as m grows to infinity the lower bound resulting from sequences σ_{m-1} tends to 2.5. Therefore, we pose the following conjecture.

Conjecture 2.10. *No online algorithm for $P|\text{online} - \text{list}, m_j|C_{\max}$ can have a competitive ratio less than* 2.5 *for job sequence σ_{m-1}, with $m \to \infty$.*

2.3.3 Application to strip packing

The presented online Algorithm PJ also applies to scheduling problems where the machines are topologically ordered on a line and only adjacent machines can be assigned to a specific job. To let Algorithm PJ apply to this case, we simply specify

that whenever a job j is assigned to some interval, it is scheduled not only at the beginning of the interval, but also assigned to the *first* m_j machines available (first with respect to the line ordering of the machines). This way we guarantee that each job j gets assigned to m_j adjacent machines and the algorithm still gives the same schedule as before. To the best of our knowledge the presented online algorithm is the first with constant competitive ratio for this problem. For previous developed online algorithms for $P|\text{online} - \text{list}, m_j|C_{\max}$ no such adaptation to this special case is possible. Since the presented online algorithm applies to this special case, it also applies to the online orthogonal strip packing problem.

Although most of the research on online orthogonal strip packing focuses on asymptotic performance ratios, Baker and Schwarz [2] developed a *Shelf Algorithm* that has competitive ratio 6.99 under the assumption that the height of a rectangle is at most 1. The performance guarantee of 6.6623 is also valid when Algorithm PJ is applied to online orthogonal strip packing, because the load and length bound from Section 2.1 also apply to the optimal offline strip packing solution. Thus, the presented algorithm not only improve the best known competitive ratio for online orthogonal strip packing, it also does not require the assumption on the bounded height.

2.4 The 2 machine case

From the previous section we see that there is a huge gap between the lower and upper bound on the competitive ratio. To obtain more insight in the problem, it is useful to study special cases with a small number of machines. They are easier to study and the ideas behind the algorithms for these special cases might form a basis for improvements on the general case.

For the case with two machines, Chan et al. [10] prove a lower bound of $1+\sqrt{2/3}$ on the competitive ratio by considering the same sequence of jobs as in Theorem 2.6. The drawback of their approach lies in determining the makespan of the optimal offline schedule. Since there are 2 machine and 3 jobs with machine requirement of 1, a case distinction has to be made on the optimal solution. Trying to improve the lower bound by using job sequence σ_n as defined in Section 2.3.2 will only require more case distinctions. To avoid this, we construct a slightly difference job sequence where the optimal schedule is uniquely determined. This sequence leads to the lower bound of 2.

2.4.1 Lower bound of 2

To prove a lower bound of 2 on the competitive ratio of any online algorithm for $P2|online - list, m_j|C_{max}$, we are going to construct a series of job sequences and argue that no online algorithm can have a makespan strictly less than twice the makespan of the optimal offline solution. In the following we assume A to be an

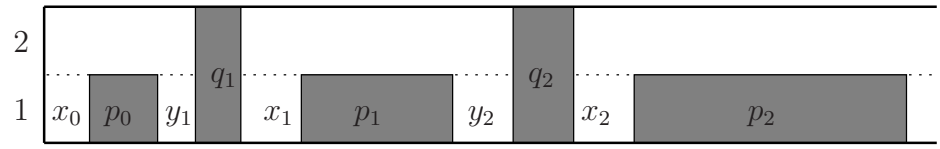


Figure 2.6: Structure of the online schedule for σ_2.

online algorithm with competitive ratio $2 - \delta$, with δ a small positive value, and we show that such an algorithm cannot exist.

We define a job sequence similar to the job sequence in Section 2.3.2. We define σ_n as the sequence of jobs $(p_0, q_1, p_1, q_2, p_2, \ldots, q_n, p_n)$, where p_i (q_i) denotes a job with processing time p_i (q_i) and a machine requirement of 1 (2). The job lengths are defined as:

$$\begin{aligned} p_0 &= 1 \\ p_1 &= x_0 + p_0 + y_1 + \epsilon \\ p_i &= 2 \cdot p_{i-1} && \forall i \geq 2 \\ q_1 &= x_0 + \epsilon \\ q_i &= \max\{y_{i-1}, q_{i-1}, x_{i-1}\} + \epsilon && \forall i \geq 2 \ , \end{aligned}$$

where x_i and y_i are values given by delays the online algorithm has used for placing earlier jobs, and ϵ is a small positive value. Note that this job sequence only differs from the one in Section 2.3.2 in the definition of p_i jobs. Again, the job lengths depend on the online Algorithm A.

From the above definition of the processing times, it is not directly clear that the jobs can only be scheduled in the order they appear. Therefore, one part of the proof consists of showing that any online algorithm with competitive ratio strictly less than 2 has to schedule the jobs in the same order as they appear in the sequence σ_n. As a consequence, Figure 2.6 illustrates the structure of the online schedule. Again, the only remaining decision for the online Algorithm A is to decide how long it delays the start of a job, i.e. how much time is left between the start of the current job and the completion of the previous job. As before, we denote by x_i (y_i) the delay incurred by Algorithm A on job p_i (q_i), completing thereby also the definition of the processing times.

To simplify notation for the remaining, we let $Q_n = \sum_{i=1}^{n} q_i$ denote the sum of processing times of the q-jobs and let $D_n = x_0 + \sum_{i=1}^{n}(y_i + x_i)$ denote the total delay on the jobs. Assuming that jobs are scheduled in the same order as they appear in σ_n, the makespan of the online schedule for σ_n is given by:

$$C_A(\sigma_n) = x_0 + p_0 + \sum_{i=1}^{n}(y_i + q_i + x_i + p_i) = \sum_{i=0}^{n} p_i + Q_n + D_n \ .$$

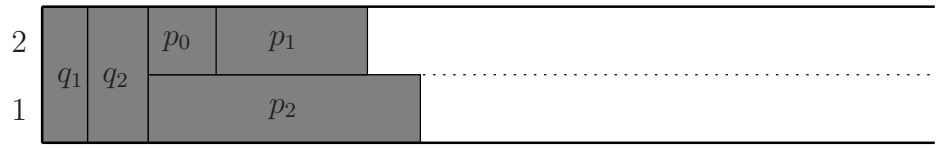


Figure 2.7: Structure of the optimal offline schedule for σ_2.

An optimal offline schedule for σ_n is obtained by scheduling the jobs $p_0, ..., p_{n-1}$ parallel to job p_n after a block containing the jobs $q_1, ..., q_n$ (see Figure 2.7). Note that due to the definition of the p-jobs we have $p_n > \sum_{i=0}^{n-1} p_i$. Therefore, the makespan of the optimal schedule is given by:

$$C^*(\sigma_n) = \sum_{i=1}^{n} q_i + p_n = Q_n + p_n \ .$$

Using $p_n = 2^{n-1} \cdot p_1$ and $\sum_{i=1}^{n} p_i = (2^n - 1) \cdot p_1$, the competitive ratio of the online Algorithm A on instance σ_n is given by

$$\begin{aligned} \frac{C_A(\sigma_n)}{C^*(\sigma_n)} &= \frac{\sum_{i=0}^{n} p_i + Q_n + D_n}{Q_n + p_n} \\ &= \frac{p_0 + (2^n - 1) \cdot p_1 + Q_n + D_n}{Q_n + 2^{n-1} \cdot p_1} \\ &= 2 - \frac{Q_n - D_n - p_0 + p_1}{Q_n + 2^{n-1} \cdot p_1} \ . \end{aligned}$$

In Lemma 2.12 we prove that for an online Algorithm A with competitive ratio $2 - \delta$ we have $Q_i + q_{i+1} < p_i$ and $x_i + y_{i+1} < p_i$. This last inequality implies that online Algorithm A schedules the job p_i after job q_i ($i \geq 2$), since $p_i = 2 \cdot p_{i-1} > x_i + p_{i-1} + y_{i+1}$ implies that before q_i no gap on any of the machines is large enough to accommodate p_i. Furthermore, by definition of the length of job q_i there is no gap in the schedule before p_{i-1} in which job q_i can be scheduled. The same holds for p_1. Summarizing, an online algorithm with competitive ratio $2 - \delta$ has to schedule the jobs in the order they appear in σ_n.

In Lemma 2.13 we prove that for an online Algorithm A with competitive ratio $2 - \delta$

$$\frac{Q_n - D_n - p_0 + p_1}{Q_n + 2^{n-1} \cdot p_1} \to 0 \ ,$$

as n goes to infinity. However, this is a contradiction with the competitive ratio being strictly less than 2. As a result, we have proven the main theorem of this section.

Theorem 2.11. *No online algorithm for $P2|online - list, m_j|C_{max}$ is $(2 - \delta)$-competitive, with $\delta > 0$.* □

To complete the proof, in the following the proof of the two lemmata are given.

Proof of the Lemmata

Lemma 2.12. *If an online algorithm A has a competitive ratio of $2-\delta$, then*

$$Q_i + q_{i+1} < p_i \tag{2.18}$$

and

$$x_i + y_{i+1} < p_i \ . \tag{2.19}$$

Proof. We prove (2.18) and (2.19) simultaneously by induction on i. If ϵ is chosen sufficiently small than the following inequalities follow from the $(2-\delta)$-competitiveness of algorithm A.

- $x_0 < p_0$: After scheduling job p_0 we have $x_0 + p_0 \leq (2-\delta) \cdot p_0$.
- $y_1 < p_0$: After scheduling job q_1 we have $x_0 + p_0 + y_1 + q_1 \leq (2-\delta) \cdot (q_1 + p_0)$, or equivalently $x_0 + y_1 \leq (1-\delta) \cdot (q_1 + p_0)$. Using $q_1 = x_0 + \epsilon$ and ϵ small enough, the inequality follows.
- $x_1 < x_0$: After scheduling job p_1 we have $x_0 + p_0 + y_1 + q_1 + x_1 + p_1 \leq (2-\delta) \cdot (q_1 + p_1)$, or equivalently $x_0 + p_0 + y_1 + x_1 \leq (1-\delta) \cdot (q_1 + p_1)$. Using $p_1 = x_0 + p_0 + y_1$, $q_1 = x_0 + \epsilon$ and ϵ small enough, the inequality follows.

By definition $q_2 = \max\{x_0, y_1, x_1\}$. Combining this with the above, we get $q_2 < p_0$. Thus, $q_1 + q_2 < x_0 + p_0 \leq p_1$ and (2.18) holds for $i = 1$.

To prove that (2.18) holds for $i \geq 2$ we assume that both (2.18) and (2.19) hold up to $i-1$. Since (2.19) holds up to $i-1$, the jobs up to job q_{i+1} are scheduled in the order as they are in σ_n. Since Algorithm A is $2-\delta$ competitive, after scheduling job p_i we have

$$\frac{p_0 + (2^i - 1) \cdot p_1 + Q_i + D_i}{Q_i + 2^{i-1} \cdot p_1} \leq 2 - \delta \ ,$$

which implies that

$$D_i - x_0 - y_1 < Q_i \ . \tag{2.20}$$

(This inequality is also used in the proof of Lemma 2.13.) By definition of the length of q_{i+1} we have either $q_{i+1} = q_i \leq Q_i$ or $q_{i+1} = \max\{x_i, y_i\} \leq D_i - x_0 - y_1 < Q_i$ since $i \geq 2$. Combining this with the induction hypothesis we have

$$Q_i + q_{i+1} < 2 \cdot Q_i < 2 \cdot p_{i-1} = p_i \ ,$$

and (2.18) also holds for i.

To prove that (2.19) holds for i we assume that (2.18) holds up to i. Since A is $2-\delta$ competitive after scheduling job q_{i+1} we have

$$\frac{p_0 + (2^i - 1) \cdot p_1 + Q_i + q_{i+1} + D_i + y_{i+1}}{Q_i + q_{i+1} + 2^{i-1} \cdot p_1} \leq 2 - \delta \ ,$$

which implies that

$$D_i + y_{i+1} - x_0 - y_1 < Q_i + q_{i+1} \ . \tag{2.21}$$

(This inequality is also used in the proof of Lemma 2.13.) Combining this with (2.18), we have

$$x_i + y_{i+1} \leq D_i + y_{i+1} - x_0 - y_1 < Q_i + q_{i+1} < p_i \ ,$$

and (2.19) also holds for i. □

Lemma 2.13. *If an online algorithm A has a competitive ratio of $2-\delta$, then*

$$\frac{Q_n - D_n - p_0 + p_1}{Q_n + 2^{n-1} \cdot p_1} \to 0 \tag{2.22}$$

if $n \to \infty$.

Proof. We prove that (2.22) holds by bounding the asymptotic growth of Q_n, i.e. by showing that $Q_n \in O(1.8^n)$. Since the denominator of (2.22) is in $\Omega(2^n)$, this proves the lemma.

We claim that either $Q_{i+1} \leq 1.8 \cdot Q_i$ or $Q_{i+2} \leq 3.2 \cdot Q_i$. Combining this with the fact that Q_n is monotone and $1.8^2 > 3.2$, we have that $Q_n \in O(1.8^n)$.

To prove the claim, assume that $Q_{i+1} > 1.8 \cdot Q_i$. This implies that $q_{i+1} > 0.8 \cdot Q_i$ by definition of Q_{i+1}, and that $D_i - x_0 - y_1 > 0.8 \cdot Q_i$ since the value of q_{i+1} is attained by one of the delays.

Now consider q_{i+2}. If $q_{i+2} > q_{i+1}$, then by using (2.20)

$$q_{i+2} \leq y_{i+1} + x_{i+1} = D_{i+1} - D_i \leq Q_{i+1} - 0.8 \cdot Q_i \ ,$$

and by using (2.21)

$$Q_{i+2} = Q_{i+1} + q_{i+2} \leq 2 \cdot Q_{i+1} - 0.8 \cdot Q_i \leq (4 - 0.8) \cdot Q_i \ .$$

On the other hand, if $q_{i+2} = q_{i+1}$, we get $Q_{i+2} = Q_i + 2 \cdot q_{i+1} \leq 3 \cdot Q_i$, since $q_{i+1} \leq Q_i$.

So in both cases the claim is true, and we have proven the lemma. □

For problem $P2|\text{online} - \text{list}, m_i|C_{\max}$, we have by Theorem 2.2 that the greedy algorithm is 2-competitive. Together with the lower bound of 2 from Theorem 2.11 we know that the greedy algorithm is best possible.

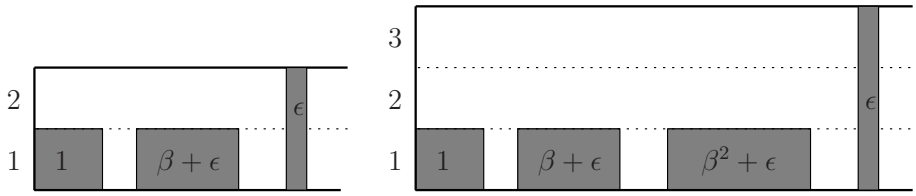


Figure 2.8: Bad instances for Algorithm PJ with small number of machines.

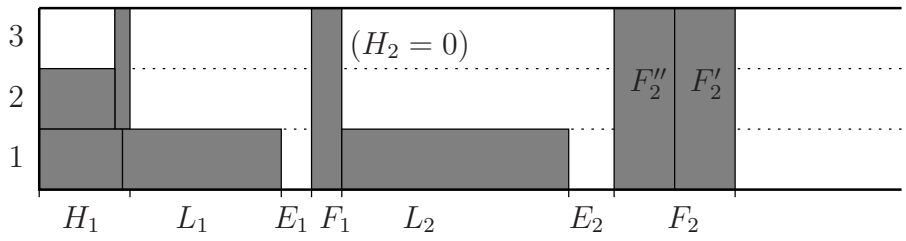


Figure 2.9: Structure of an online schedule created by Algorithm 3M.

2.5 The 3 machine case: a 2.8-competitive algorithm

The Algorithm PJ presented in Section 2.3.1 is of no use when dealing with a small number of machines. For example, for the problem with 2 machines and a sequence of 3 jobs with processing times $1, \beta+\epsilon, \epsilon$ and machine requirements $1, 1, 2$, respectively, the resulting competitive ratio of Algorithm PJ is $\frac{\beta+\beta^2+\epsilon}{\beta+2\epsilon}$. When ϵ goes to 0, this ratio goes to $\beta+1$ (≈ 2.63), see Figure 2.8. And for the problem with 3 machines and a sequence of 4 jobs with processing times $1, \beta+\epsilon, \beta^2+\epsilon, \epsilon$ and machine requirements $1, 1, 1, 3$, respectively, the resulting competitive ratio of Algorithm PJ is $\frac{\beta+\beta^2+\beta^3+\epsilon}{\beta^2+2\epsilon}$. When ϵ goes to 0, this ratio goes to $\beta+1+\frac{1}{\beta}$ (≈ 3.24), see Figure 2.8. Thus, for small number of machines the m-competitive greedy algorithm outperforms Algorithm PJ.

For the online parallel job scheduling problem with only two machines it is not possible to improve upon the 2-competitive greedy algorithm, as shown in Section 2.4. In the following, we show that for the case with three machines it is possible to design an online algorithm which is better than the 3-competitive greedy algorithm. However, its competitive ratio is only slightly better, i.e. it is 2.8-competitive, and thereby still far above the best known lower bound of 2 given in [7].

Consider the following algorithm for $P3|\text{online} - \text{list}, m_j|C_{\max}$.

Algorithm 3M

Schedule job j by the following rules:
if $m_j = 1$ or 2 **then** schedule job j in a greedy fashion.
if $m_j = 3$ **then** consider:

- **if** there is an empty interval within the current online schedule large enough to accommodate job j, **then** schedule job j in the first of these intervals and as late as possible within that interval. (This will be immediately before another job with machine requirement 3.)
- **else**, **if** the last job in the schedule has machine requirement of 3, **then** concatenate job j to this job at the back.
- **else**, delay job j for a period d (which we define later) after the last scheduled job.

This algorithm differs from the greedy algorithm only by the way it schedules jobs which need all three machines for processing. Each schedule created by Algorithm 3M consists of intervals of four different types: *full* intervals F, *high* loaded intervals H, *low* loaded intervals L, and *empty* intervals E. The F-intervals contain the jobs with $m_j = 3$, the H-intervals are the intervals containing jobs with $m_j = 2$ or 1 and in which at least 2 machines are busy, the L-intervals are the intervals which have exactly 1 machine busy (and, thus, contain only jobs with $m_j = 1$), and the E-intervals are the intervals with no machine busy.

Using this classification, each online schedule created by Algorithm 3M can be partitioned into consecutive blocks where the i^{th} block consists of four consecutive intervals H_i, L_i, E_i, F_i, where some of the intervals H_i, L_i or E_i may have zero length. Since we schedule jobs with $m_j = 1$ and $m_j = 2$ in a greedy fashion, the interval H_i, L_i and E_i always occur in this order between two consecutive non-empty F-intervals F_{i-1} and F_i. We use the terms H_i, L_i, E_i, F_i to indicate both the interval and to indicate the length of the respective interval. In Figure 2.9 an example of the structure of an online schedule is given.

Each interval F_i contains one job that was delayed by Algorithm 3M. This job is the job of F_i which was scheduled first by the online algorithm. Let this job together with all jobs concatenated after it form the interval F_i', and let the jobs that are concatenated before this job form the interval F_i''. Thus, $F_i = F_i'' + F_i'$ (see Figure 2.9).

Now consider the situation that a job with $m_j = 3$ is delayed by Algorithm 3M. At that moment the online schedule ends with an H_i or L_i interval. We define the delay d for this job as $\left(\frac{1}{2}L_i - \frac{1}{4}H_i\right)^+$ $\left(= \max\{0, \frac{1}{2}L_i - \frac{1}{4}H_i\}\right)$. By scheduling this job, we create the interval E_i of length d, and F_i consists only of the last job scheduled. During the course of the algorithm, E_i may decrease in length and F_i may increase in length (but not vice versa). With $\tilde{H}_i$, $\tilde{L}_i$, and $\tilde{E}_i$ we refer to the intervals/ values of H_i, L_i, and E_i at the moment that interval F_i is created.

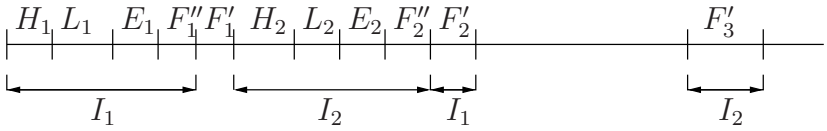


Figure 2.10: Definition of intervals I_i

In the following, we evaluate an online schedule created by Algorithm 3M for a sequence of jobs σ. Let k be the number of F_i intervals in the online schedule. The makespan of the online schedule $C_{3\mathrm{M}}(\sigma)$ is given by

$$C_{3\mathrm{M}}(\sigma) = \sum_{i=1}^{k} (H_i + L_i + E_i + F_i) + H_{k+1} + L_{k+1} \ ,$$

where H_{k+1} and L_{k+1} may have length 0. To get a more helpful description of the makespan, we introduce intervals I_i by

$$I_i := H_i \cup L_i \cup E_i \cup F_i'' \cup F_{i+1}' \ ,$$

see Figure 2.10. The idea behind defining I_i in this way, is that the average load in I_i will be sufficiently high. The lack of load in L_i and E_i is compensated by the load in F_{i+1}', since interval F_{i+1}' contains a job with processing time larger than E_i and machine requirement of 3.

Using the above definition, the makespan $C_{3\mathrm{M}}(\sigma)$ can be expressed by

$$C_{3\mathrm{M}}(\sigma) = \sum_{i=1}^{k-1} I_i + F_1' + H_k + L_k + E_k + F_k'' + H_{k+1} + L_{k+1} \ . \tag{2.23}$$

Now, let $l(t)$ be the load (the number of machines in use) at time t in the schedule. The total load of the schedule in I_i can be bounded from below as in the following lemma.

Lemma 2.14. *For $i \leq k-1$ we have:*

$$\int_{I_i} l(t)dt > \frac{5}{3} I_i - \frac{5}{3} F_{i+1}' \ .$$

Proof. The definition of interval I_i implies:

$$\begin{aligned} \int_{I_i} l(t)dt &= \int_{H_i} l(t)dt + \int_{L_i} l(t)dt + \int_{F_i''} l(t)dt + \int_{F_{i+1}'} l(t)dt \\ &\geq 2H_i + L_i + 3F_i'' + 3F_{i+1}' \\ &= \frac{5}{3} I_i + \frac{1}{3} H_i - \frac{2}{3} L_i - \frac{5}{3} E_i + \frac{4}{3} \left(F_i'' + F_{i+1}' \right) \ . \end{aligned} \tag{2.24}$$

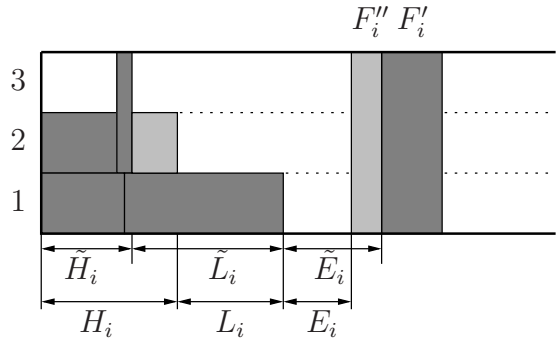


Figure 2.11: The case $H_i + L_i = \tilde{H}_i + \tilde{L}_i$

At the time the first job in F'_{i+1} was placed, it had to be delayed since it did not fit in the empty intervals before F_i. Together with the fact that E_i is non-increasing this yields

$$F'_{i+1} > E_i \ . \tag{2.25}$$

There are two cases to distinguish.

Case 1: $H_i + L_i = \tilde{H}_i + \tilde{L}_i$

This implies that $\tilde{E}_i$ has not decreased due to the insertion of jobs with $m_j = 1$ or 2, i.e. $\tilde{E}_i = E_i + F_i'' <_{(2.25)} F'_{i+1} + F_i''$. Since furthermore $H_i \geq \tilde{H}_i$ and $L_i \leq \tilde{L}_i$ (see Figure 2.11), we get

$$\begin{aligned}
\int_{I_i} l(t)dt \quad &\geq \quad \frac{5}{3}I_i + \frac{1}{3}\tilde{H}_i - \frac{2}{3}\tilde{L}_i - \frac{5}{3}E_i + \frac{4}{3}\left(F_i'' + F'_{i+1}\right) \\
&> \quad \frac{5}{3}I_i + \frac{1}{3}\tilde{H}_i - \frac{2}{3}\tilde{L}_i - \frac{5}{3}E_i + 3\tilde{E}_i - \frac{5}{3}\left(F_i'' + F'_{i+1}\right) \\
&=_{(-E_i - F_i'' = -\tilde{E}_i)} \quad \frac{5}{3}I_i + \frac{1}{3}\tilde{H}_i - \frac{2}{3}\tilde{L}_i + \frac{4}{3}\tilde{E}_i - \frac{5}{3}F'_{i+1} \ .
\end{aligned}$$

Since $\tilde{E}_i$ is equal to the delay of the first scheduled job of F_i, we have $\tilde{E}_i = (\frac{1}{2}\tilde{L}_i - \frac{1}{4}\tilde{H}_i)^+$, and thus,

$$\int_{I_i} l(t)dt \quad > \quad \frac{5}{3}I_i - \frac{5}{3}F'_{i+1} \ .$$

Case 2: $H_i + L_i > \tilde{H}_i + \tilde{L}_i$

In this case $\tilde{E}_i > 0$ and $\tilde{E}_i$ has been decreased (partially) due to the insertion of $m_j = 1$ or $m_j = 2$ jobs. Due to the greedy nature of Algorithm 3M this can only happen if the whole interval $\tilde{L}_i$ becomes part of H_i (see Figure 2.12).

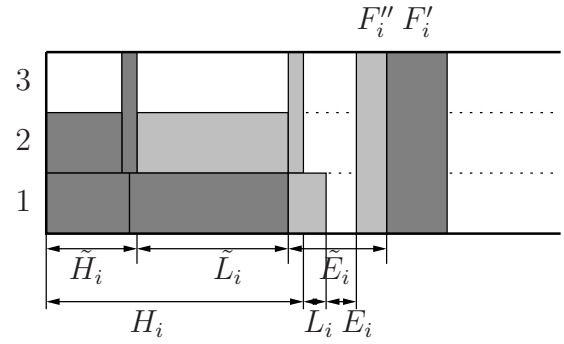


Figure 2.12: The case $H_i + L_i > \tilde{H}_i + \tilde{L}_i$

Defining e via $H_i = \tilde{H}_i + \tilde{L}_i + e$, we get $L_i = \tilde{E}_i - e - E_i - F_i''$. Starting from (2.24), we have

$$\begin{aligned}
\int_{I_i} l(t)dt \quad &\geq \quad \frac{5}{3}I_i + \frac{1}{3}H_i - \frac{2}{3}L_i - \frac{5}{3}E_i + \frac{4}{3}\left(F_i'' + F_{i+1}'\right) \\
&= \quad \frac{5}{3}I_i + \frac{1}{3}(\tilde{H}_i + \tilde{L}_i) + e - E_i - \frac{2}{3}\tilde{E}_i + 2F_i'' + \frac{4}{3}F_{i+1}' \\
&>_{(2.25)} \quad \frac{5}{3}I_i + \frac{1}{3}(\tilde{H}_i + \tilde{L}_i) + e - \frac{2}{3}\tilde{E}_i + 2F_i'' + \frac{1}{3}F_{i+1}' \ .
\end{aligned}$$

Since $\tilde{E}_i > 0$, the delay is given by $\tilde{E}_i = \frac{1}{2}\tilde{L}_i - \frac{1}{4}\tilde{H}_i$. This yields

$$\begin{aligned}
\int_{I_i} l(t)dt \quad &> \quad \frac{5}{3}I_i + \frac{1}{2}\tilde{H}_i + e + 2F_i'' + \frac{1}{3}F_{i+1}' \\
&\geq \quad \frac{5}{3}I_i + \frac{1}{2}\tilde{H}_i + \frac{1}{3}F_{i+1}' \quad \geq \quad \frac{5}{3}I_i - \frac{5}{3}F_{i+1}' \ .
\end{aligned}$$

Thus, in both cases the lemma holds. □

Lemma 2.14 is a useful tool to connect the online makespan $C_{3\mathrm{M}}(\sigma)$ with the load bound on $C^*(\sigma)$. Using this connection, the competitive ratio of Algorithm 3M can be bounded to 2.8.

Theorem 2.15. *For* $P3|\mathrm{online-list}, m_j|C_{\max}$ *Algorithm* 3M *is* 2.8*-competitive.*

Proof. Combining the load bound with Lemma 2.14 we get

$$\begin{aligned} C^*(\sigma) \quad &\geq \quad \frac{1}{3}\sum m_j p_j \quad = \frac{1}{3}\int_0^{C_{3\mathrm{M}}(\sigma)} l(t)dt \\ &\geq_{(2.23)} \quad \frac{1}{3}\sum_{i=1}^{k-1}\int_{I_i} l(t)dt + \frac{2}{3}\left(H_k + H_{k+1}\right) + \frac{1}{3}\left(L_k + L_{k+1}\right) + F_1' + F_k'' \\ &\geq_{(\text{Lem. } 2.14)} \quad \sum_{i=1}^{k-1}\left(\frac{5}{9}I_i - \frac{5}{9}F_{i+1}'\right) + \frac{2}{3}\left(H_k + H_{k+1}\right) + \\ &\qquad \frac{1}{3}\left(L_k + L_{k+1}\right) + F_1' + F_k'' \ . \end{aligned} \tag{2.26}$$

Besides the load bound on the optimal makespan $C^*(\sigma)$, another lower bound resulting from an improved length argument can be used. Let $p_{\max}$ denote the longest processing time of the jobs with $m_j < 3$, i.e. $p_{\max} = \max_{(j\in\sigma | m_j<3)} p_j$. In the optimal schedule all jobs with $m_j = 3$ and this long job have to be processed one after the other. This gives

$$C^*(\sigma) \geq \sum_{j=1}^{k} F_j + p_{\max} \ . \tag{2.27}$$

The lower bounds (2.26) and (2.27) on $C^*(\sigma)$ enable us to show that $C_{3\mathrm{M}}(\sigma) \leq 2.8 \cdot C^*(\sigma)$ holds. With the load bound (2.26) we can 'get rid' of the I_i intervals upto $i = k-1$ in the expression (2.23) for $C_{3\mathrm{M}}(\sigma)$ by loosing only a factor $\frac{9}{5}$ compared to $C^*(\sigma)$, i.e. rewriting (2.26) gives

$$\begin{aligned} \sum_{i=1}^{k-1} I_i \quad &\leq \quad \frac{9}{5}C^*(\sigma) + \sum_{i=1}^{k-1} F_{i+1}' - \frac{6}{5}\left(H_k + H_{k+1}\right) - \\ &\qquad \frac{3}{5}\left(L_k + L_{k+1}\right) - \frac{9}{5}(F_1' + F_k'') \ . \end{aligned} \tag{2.28}$$

Thus,

$$\begin{aligned} C_{3\mathrm{M}}(\sigma) \quad &=_{(2.23)} \quad \sum_{i=1}^{k-1} I_i + F_1' + H_k + L_k + E_k + F_k'' + H_{k+1} + L_{k+1} \\ &\leq_{(2.28)} \quad \frac{9}{5}C^*(\sigma) + \sum_{i=2}^{k} F_i' - \frac{1}{5}\left(H_k + H_{k+1}\right) + \frac{2}{5}\left(L_k + L_{k+1}\right) \\ &\qquad + E_k - \frac{4}{5}\left(F_1' + F_k''\right) \\ &\leq \quad \frac{9}{5}C^*(\sigma) + \sum_{i=1}^{k} F_i - \frac{1}{5}\left(H_k + H_{k+1}\right) + \frac{2}{5}\left(L_k + L_{k+1}\right) \\ &\qquad + E_k - \frac{9}{5}F_k'' \ . \end{aligned}$$

Let

$$\Delta = -\frac{1}{5}\left(H_k + H_{k+1}\right) + \frac{2}{5}\left(L_k + L_{k+1}\right) + E_k - \frac{9}{5}F_k'' \ .$$

By a number of case distinctions we show that $\Delta \leq p_{\max}$, which implies

$$C_{3\mathrm{M}}(\sigma) \quad \leq \quad \frac{9}{5}C^*(\sigma) + \sum_{i=1}^{k} F_i + p_{\max} \quad \leq_{(2.27)} \quad \frac{14}{5}C^*(\sigma) \ .$$

Due to the greedy nature of Algorithm 3M we know that no job starts in the interior of an interval L_i. Therefore, $p_{\max}$ is larger than L_{k+1} and $\tilde{L_k}$. Furthermore,

$$E_k \leq \tilde{E_k} = \left(\frac{1}{2}\tilde{L_k} - \frac{1}{4}\tilde{H_k}\right)^+ \leq \frac{1}{2}\tilde{L_k} \leq \frac{1}{2}p_{\max} \ . \tag{2.29}$$

To show that $\Delta \leq p_{\max}$, we consider 4 cases.

Case 1: $\tilde{E_k} = 0$

Since $\tilde{E_i} = (\frac{1}{2}\tilde{L_i} - \frac{1}{4}\tilde{H_i})^+ = 0$, we know that $\frac{1}{4}\tilde{H_k} \geq \frac{1}{2}\tilde{L_k}$. Therefore, $\frac{1}{4}H_k \geq \frac{1}{2}L_k$. Since, furthermore $E_k \leq \tilde{E_k} = 0$ and $F_k'' \leq \tilde{E_k} = 0$, we get

$$\begin{aligned} \Delta \quad &\leq \quad -\frac{1}{5}\left(H_k + H_{k+1}\right) + \frac{2}{5}\left(\frac{1}{2}H_k + L_{k+1}\right) \\ &\leq \quad -\frac{1}{5}H_{k+1} + \frac{2}{5}L_{k+1} \quad \leq \quad \frac{2}{5}p_{\max} \ . \end{aligned}$$

Case 2: $H_{k+1} > 0$ (and $\tilde{E_k} > 0$)

Due to the greedy nature of Algorithm 3M we have $H_{k+1} > L_k + E_k$. Thus,

$$\begin{aligned} \Delta \quad &\leq \quad -\frac{1}{5}H_k - \frac{1}{5}(L_k + E_k) + \frac{2}{5}(L_k + L_{k+1}) + E_k \\ &\leq \quad \frac{1}{5}L_k + \frac{2}{5}L_{k+1} + \frac{4}{5}E_k \\ &\leq \quad \left(\frac{1}{5} + \frac{2}{5} + \frac{4}{5}\frac{1}{2}\right)p_{\max} = p_{\max} \ . \end{aligned}$$

Case 3: $H_k + L_k > \tilde{H_k} + \tilde{L_k}$, (and $H_{k+1} = 0$, $\tilde{E_k} > 0$)

This case is depicted in Figure 2.12 on page 39. We have $L_k + E_k < \tilde{E_k} \leq \frac{1}{2}\tilde{L_k}$ and $H_k > \tilde{L_k}$. Thus,

$$\begin{aligned} \Delta \quad &\leq \quad -\frac{1}{5}H_k + \frac{2}{5}(L_k + L_{k+1}) + E_k \quad \leq \quad -\frac{1}{5}\tilde{L_k} - \frac{3}{5}L_k + \frac{2}{5}L_{k+1} + \tilde{E_k} \\ &\leq \quad -\frac{1}{5}\tilde{L_k} + \frac{2}{5}L_{k+1} + \tilde{E_k} \quad \leq \quad -\frac{1}{5}\tilde{L_k} + \frac{2}{5}L_{k+1} + \frac{1}{2}\tilde{L_k} \\ &\leq \quad \frac{3}{10}\tilde{L_k} + \frac{2}{5}L_{k+1} \quad \leq \quad \frac{7}{10}p_{\max} \ . \end{aligned}$$

Case 4: $H_k + L_k = \tilde{H}_k + \tilde{L_k}$, (and $H_{k+1} = 0$, $\tilde{E}_k > 0$)
This case is depicted in Figure 2.11 on page 38. Let $\gamma \geq 0$ be such that $L_k = \tilde{L_k} - \gamma\tilde{E}_k$. Then $H_k = \tilde{H}_k + \gamma\tilde{E}_k$. Due to the greedy nature of Algorithm 3M we know that L_{k+1} consists only of one job and, thus, is larger than $L_k + E_k$. This gives,

$$\begin{aligned} L_{k+1} &> L_k + E_k \\ &= \tilde{L_k} - \gamma\tilde{E}_k + \tilde{E}_k - F_k'' \\ &\geq_{(2.29)} (3-\gamma)\,\tilde{E}_k - F_k'' \ . \end{aligned}$$

As long as $\gamma \leq 3$ we have:

$$\tilde{E}_k \leq \frac{L_{k+1} + F_k''}{3-\gamma} \ . \tag{2.30}$$

Thus,

$$\begin{aligned} \Delta &\leq -\frac{1}{5}H_k + \frac{2}{5}(L_k + L_{k+1}) + E_k - \frac{9}{5}F_k'' \\ &\leq -\frac{1}{5}\left(\tilde{H}_k + \gamma\tilde{E}_k\right) + \frac{2}{5}\left(\tilde{L_k} - \gamma\tilde{E}_k + L_{k+1}\right) + (\tilde{E}_k - F_k'') - \frac{9}{5}F_k'' \\ &\leq -\frac{1}{5}\tilde{H}_k + \frac{2}{5}\left(\tilde{L_k} + L_{k+1}\right) + \left(1 - \frac{3\gamma}{5}\right)\tilde{E}_k - \frac{14}{9}F_k'' \ . \end{aligned} \tag{2.31}$$

Since $\tilde{E}_k > 0$, we have by definition $\tilde{E}_k = \frac{1}{2}\tilde{L_k} - \frac{1}{4}\tilde{H}_k$. This implies

$$\frac{2}{5}\tilde{L_k} = \frac{2}{5}\left(2\tilde{E}_k + \frac{\tilde{H}_k}{2}\right) = \frac{4}{5}\tilde{E}_k + \frac{1}{5}\tilde{H}_k \ .$$

Combining this with (2.31) gives

$$\Delta \leq \frac{2}{5}L_{k+1} + \left(\frac{9}{5} - \frac{3\gamma}{5}\right)\tilde{E}_k - \frac{14}{5}F_k'' \tag{2.32}$$

For $\gamma \geq 3$, inequality (2.32) results in

$$\Delta \leq \frac{2}{5}L_{k+1} \leq \frac{2}{5}p_{\max} \ .$$

For $\gamma \in [0, 3]$ we can use (2.30) in (2.32), leading to

$$\begin{aligned} \Delta &\leq \frac{2}{5}L_{k+1} + \frac{\frac{9}{5} - \frac{3\gamma}{5}}{3-\gamma}(L_{k+1} + F_k'') - \frac{14}{5}F_k'' \\ &= \frac{2}{5}L_{k+1} + \frac{3}{5}(L_{k+1} + F_k'') - \frac{14}{5}F_k'' \\ &\leq \frac{2}{5}L_{k+1} + \frac{3}{5}L_{k+1} \leq p_{\max} \ . \end{aligned}$$

Summarizing, for each case we have $\Delta \leq p_{\max}$, proving that, Algorithm 3M is 2.8-competitive. □

For the analysis in Lemma 2.14 and Theorem 2.15 the definition of the delay d is crucial. Defining the delay as $(xL_i - yH_i)^+$ and optimizing on the values of x and y, gives that the delay $d := (\frac{1}{2}L_i - \frac{1}{4}H_i)^+$ is the best possible. It is however unknown if the analysis is tight; no instance is known for which Algorithm 3M attains the ratio 2.8. So, to improve the upon the 2.8-competitive Algorithm 3M one either needs to find new arguments in bounding the optimal solution or a new design for the online algorithm.

2.6 Semi-online parallel jobs

In the previous sections we dealt with *pure* online-list scheduling of parallel jobs on different numbers of machines. With the exception of the two machine case, the upper bounds on the competitive ratios are far above the lower bounds. If we assume that there is some limited knowledge of the jobs to come, we can obtain smaller competitive ratios. An online problem with some a priori knowledge of the job sequence is called a semi-online problem.

In this section we consider the semi-online case of $P|\text{online} - \text{list}, m_j|C_{\max}$ where the jobs arrive in non-increasing order of machine requirement m_j. The concrete processing times and machine requirements are still unknown until the jobs are revealed to the scheduler. However, the scheduler knows that the machine requirement of job σ_{j+1} will be no larger than that of job σ_j. Before presenting an online algorithm which improves the best known algorithm for this semi-online case, we review known results for several semi-online versions of $P|\text{online} - \text{list}, m_j|C_{\max}$.

2.6.1 Known results on semi-online parallel job scheduling

A number of semi-online versions of parallel job scheduling are studied in the literature. They differ from each other in the a priori knowledge about the job sequence. The results presented in this section are summarized in Table 2.4.

Non-increasing m_j. In case the jobs appear in non-increasing order of machine requirement the best known lower bound is 1.88 from classical parallel machine scheduling, where all jobs have $m_j = 1$ [72]. Scheduling the jobs greedily is 2.75-competitive and no better than 2.5 [81]. In Section 2.6.2 we improve this result by showing a 2.4815-competitive algorithm. For $m = 2$ and $m = 3$ a greedy algorithm is $(2 - \frac{1}{m})$-competitive, and from classical parallel machine scheduling we know this is the best possible [21]. Surprisingly, for $m = 4$ and $m = 5$ the competitive ratio of greedy is the same, both being 2-competitive. We show in Section 2.6.3 a 1.7808-competitive algorithm for $m = 4$.

Non-increasing p_j. In case the jobs appear in non-increasing order of processing time a greedy algorithm is 2-competitive [81]. The best known lower bound is $\frac{5}{3}$ from

the strip packing problem [7]. For $m = 2$, a lower bound of $\frac{9}{7}$ and a $\frac{4}{3}$-competitive online algorithm is known [10].

Non-decreasing p_j. For the case where jobs appear in non-decreasing order of processing times a $\frac{3}{2}$-competitive algorithm is given in [10]. This algorithm is optimal for the problem with two machines. Optimality follows from the lower bound from classical parallel machine scheduling [21].

Agreeable. Jobs in a sequence are called agreeable if whenever $p_j > p_i$ also $m_j \geq m_i$ holds. If jobs are agreeable and are non-increasing in processing time (thus also non-increasing in machine requirement) a generalization of the the First-Fit-Decreasing heuristic for 1-dimensional bin-packing, first developed in [43], gives a 1.875-competitive algorithm. If $m_j \leq \frac{m}{2}$ for all j then this algorithm is 1.75-competitive [57]. Later in [83], the ratio for arbitrary machine requirements is improved to 1.75. Only a lower bound implied by the results for the classical parallel machine scheduling problem is known, that is $\frac{4}{3}$.

If the jobs are agreeable and are non-decreasing in processing time (and machine requirement), a lower bound of $\frac{1+\sqrt{7}}{2} > 1.8229$ is given in [7]. No special upper bounds are known.

Longest processing time known. For the case where the longest processing time is known, a 4-competitive algorithm is given in [82]. This known length can be used in the competitive analysis even if the job attaining this length has not appeared yet, since in this situation we know that the sequence is not finished.

Semi-online $P\|online-list, m_j\|C_{max}$				
Model	Lower Bound		Upper Bound	
-non-increasing m_j	1.88 ($m_j = 1$),	[72]	2.4815,	[Sec. 2.6.2]
$m = 2$ or 3	$2 - \frac{1}{m}$,	[21]	$2 - \frac{1}{m}$	[Greedy]
$m = 4$	-		1.7808,	[Sec. 2.6.3]
-non-increasing p_j	$\frac{5}{3}$,	[7]	2,	[81]
$m = 2$	$\frac{9}{7}$,	[10]	$\frac{4}{3}$,	[10]
-non-decreasing p_j	-		-	
$m = 2$	$\frac{3}{2}$,	[21]	$\frac{3}{2}$,	[10]
-non-increasing p_j and m_j	$\frac{4}{3}$ ($m_j = 1$)		$\frac{7}{4}$,	[83]
-non-decreasing p_j and m_j	1.8229,	[7]	-	
-largest p_j known	-		4,	[82]

Table 2.4: Results for semi-online scheduling of $P|\text{online} - \text{list}, m_j|C_{\max}$

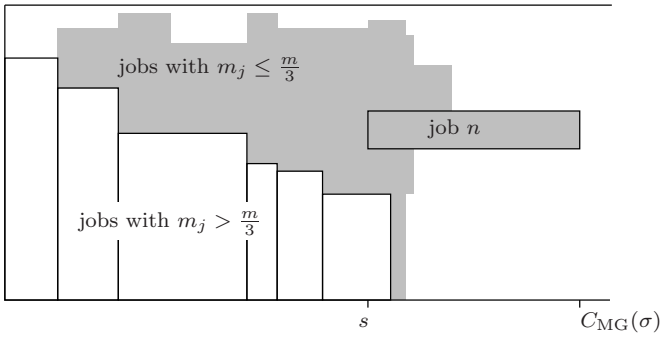


Figure 2.13: Structure of the online schedule created by MG.

2.6.2 Non-increasing m_j: a 2.4815-competitive algorithm

For the semi-online parallel job scheduling problem with jobs appearing with non-increasing machine requirement we present an algorithm that is 2.4815-competitive. This algorithm classifies the jobs into two categories depending on their machine requirement. In contrast to Section 2.3.1, we now call a job j *big* if $m_j > \frac{m}{3}$ and *small* if $m_j \leq \frac{m}{3}$. Let B be the set of big jobs and S be the set of small jobs. The presented algorithm is a modified version of the greedy algorithm in the following way. It schedules first all the big jobs consecutively and then schedules the small jobs in a greedy manner.

Algorithm Modified Greedy (MG)
- Schedule the big jobs one after the other.
- Schedule the small jobs in a greedy fashion.

Since the jobs appear in non-increasing order of m_j, the two steps of the algorithm are always executed in the given order. Figure 2.13 illustrates the structure of online schedules created by the Algorithm MG. The shaded area indicates where small jobs are scheduled in a greedy fashion.

Theorem 2.16. *Algorithm MG is* $\left(\frac{5}{2} - \frac{3}{2m}\right)$*-competitive for* $m \geq 3$ *and* 2*-competitive for* $m = 2$.

Proof. Let $\sigma = (1, \ldots, n)$ be an arbitrary sequence of jobs with non-increasing m_j. First we consider a few easy cases. If $m = 2$ all jobs are big jobs scheduled one after the other, resulting in a ratio of 2. If $m \geq 3$ and as long as only big jobs arrive, the algorithm is 2-competitive, since in the optimal offline schedule no more than two jobs with $m_j > \frac{m}{3}$ can be scheduled in parallel. Therefore, we only have to analyze the algorithm if at least one small job has been scheduled. Furthermore, we only have to consider a situation where the last scheduled job determines the makespan of

the schedule created by MG, since the arrival of successive jobs that do not increase the online makespan might increase the optimal offline makespan and can therefore only lead to a better competitive ratio. Let the last job be job n, and let s denote the starting time of this job n (see Figure 2.13).

The makespan of the online schedule is given by $C_{\mathrm{MG}}(\sigma) = s + p_n$. Since no jobs with $m_j > \frac{m}{3}$ are scheduled in parallel and the jobs with $m_j \leq \frac{m}{3}$ are scheduled in a greedy fashion, there are at least $\frac{2m}{3}$ machines busy at each time in $[0, s)$. The load in $[0, s)$ is, therefore, at least $\frac{2m}{3}s$. Since job n starts at s, this implies $s \leq \frac{3}{2m}\sum_{i=1}^{n-1} m_i p_i$. Using the load and length bound to lower bound the optimal offline solution (see Section 2.1) we get

$$\begin{aligned} C_{\mathrm{MG}}(\sigma) &= s + p_n \quad \leq \frac{3}{2m}\sum_{i=1}^{n-1} m_i p_i + p_n \\ &= \frac{3}{2m}\sum_{i=1}^{n} m_i p_i + \left(1 - \frac{3m_n}{2m}\right) p_n \\ &\leq \frac{3}{2}C^*(\sigma) + \left(1 - \frac{3m_n}{2m}\right) C^*(\sigma) \quad \leq \left(\frac{5}{2} - \frac{3}{2m}\right) C^*(\sigma) \ , \end{aligned}$$

proving the theorem. □

For a small number of machines, Theorem 2.16 gives competitive ratios considerably smaller than 2.5. However, for a large number of machines the competitive ratio given in Theorem 2.16 still tends to 2.5, which is the best known lower bound on the competitive ratio of the greedy algorithm. In the following we prove that Algorithm MG has for every number of machines a competitive ratio of at most 2.4815. For this, let r be the start time of the job that is the first job Algorithm MG has scheduled parallel to another job (first in execution of Algorithm MG not necessary first on the time axis). Furthermore, let t be the start time of the job that is the first job starting after r that is scheduled parallel to two other jobs. Let h be this jobs starting at t. A sketch of a possible online schedule constructed upto h is given in Figure 2.14. In the schedule of Figure 2.14 no small job starts before r, but this possibility is not excluded.

For instances that have a resulting online schedule created by Algorithm MG containing a job h as defined above, and thus time t is defined, we have the following lemma.

Lemma 2.17. *A lower bound on the optimal makespan $C^*(\sigma)$ is given by $\frac{3}{4}t$, i.e. $C^*(\sigma) \geq \frac{3}{4}t$.*

Proof. Let $\sigma = (1, \ldots, n)$ be an arbitrary sequence of jobs with non-increasing m_j. We can assume that h is the last job in the sequence of jobs, because the addition of more jobs will only increase the value of $C^*(\sigma)$ and not of t. Since h is scheduled in parallel to two other jobs and has the smallest machine requirement of these jobs,

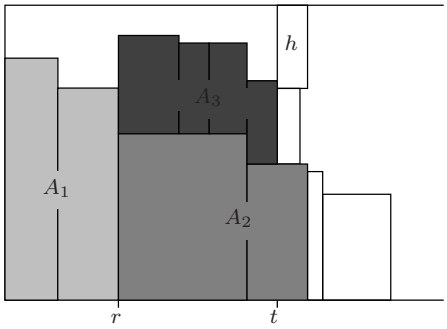


Figure 2.14: Sketch of an online schedule upto job h.

we have $m_h \leq \frac{m}{3}$. Furthermore, since h is the first such job, at each point in time before t at least one job from B is scheduled, i.e. $t \leq \sum_{j \in B} p_j$.

Let A_1 be the subset of B which contains all jobs with $m_j > \frac{2m}{3}$ which are scheduled to start in $[0, r)$. Due to the definition of r one job $j \in A_1$ has to complete at r (or $A_1 = \emptyset$ and $r = 0$). Let A_2 (A_3) be the subset of B (S) which contain the jobs scheduled to start in $[r, t)$ (see Figure 2.14).

To bound the makespan of the optimal schedule observe the following. The jobs in A_2 cannot be scheduled in parallel to jobs in A_1 but possibly can be scheduled in parallel with other jobs in A_2 (see Figure 2.15). Let $|A_i|$ denote the total processing time of the jobs in A_i. Since the total processing time of the jobs in A_2 is at least $t - r$ we have

$$\begin{aligned} C^*(\sigma) &\geq |A_1| + \frac{1}{2}|A_2| \\ &\geq r + \frac{1}{2}(t - r) = \frac{1}{2}(r + t) \ . \end{aligned} \tag{2.33}$$

On the other hand, due to the definition of A_i, jobs from the sets A_1, A_2 and A_3 which can be scheduled in parallel with two other jobs from these sets, are the jobs belonging to A_3. No two jobs from A_2 and one from A_3 can go in parallel, since this would contradict the definition of job h. Thus, the most compact way to schedule the jobs from A_1, A_2 and A_3 are as indicated in Figure 2.15. Formally, this yields

$$\begin{aligned} C^*(\sigma) &\geq |A_1| + \frac{1}{2}|A_2| + \frac{1}{3}(|A_3| - |A_1|) \\ &\geq r + \frac{1}{2}(t - r) + \frac{1}{3}(t - 2r) = \frac{5}{6}t - \frac{1}{6}r \ . \end{aligned} \tag{2.34}$$

If $r \geq \frac{1}{2}t$ we have by (2.33) that

$$C^*(\sigma) \geq \frac{1}{2}\left(\frac{1}{2}t + t\right) = \frac{3}{4}t \ ,$$

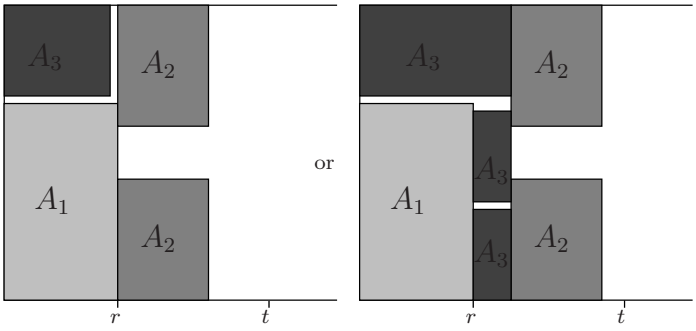


Figure 2.15: Sketch of the optimal solution.

and if $r \leq \frac{1}{2}t$ we have by (2.34) that

$$C^*(\sigma) \geq \frac{5}{6}t - \frac{1}{12}t = \frac{3}{4}t \ .$$

This proves the lemma. □

Lemma 2.17 gives a new lower bound on the optimal offline makespan $C^*(\sigma)$, enabling us to strengthen the upper bound on the competitive ratio of Algorithm MG.

Theorem 2.18. *Algorithm MG is* $\frac{67}{27}$ (≈ 2.4815)*-competitive.*

Proof. Let σ be an arbitrary sequence of jobs with non-increasing m_j. Without loss of generality, we can assume that the last completing job, called job n, is the last job in the sequence of jobs and it has start time s.

Case 1: There are no three jobs scheduled in parallel to each other after r, i.e. job h does not exist and point t is not defined.

When job n is a big job, the online makespan is within a factor of two of the optimal makespan. When job n is small, we get by the same reasoning as in the proof of Lemma 2.17 (substitute t by s) that $C^*(\sigma) \geq \frac{3}{4}s$, implying

$$C_{\mathrm{MG}}(\sigma) = s + p_n \leq \frac{4}{3}C^*(\sigma) + C^*(\sigma) = \frac{7}{3}C^*(\sigma) \ .$$

Case 2: There are three jobs scheduled in parallel to each other after r.

Again, if job n is a big job the schedule is 2-competitive. In the following job n is a small job. If $s \leq t$, we get

$$C_{\mathrm{MG}}(\sigma) = s + p_n \leq t + p_n \leq \frac{4}{3}C^*(\sigma) + C^*(\sigma) = \frac{7}{3}C^*(\sigma) \ .$$

Thus, we only have to consider the case $s > t$. Let αt be the length of the interval $[t, s)$, i.e. $\alpha t = s - t$ and

$$C_{\mathrm{MG}}(\sigma) \quad = \quad (1+\alpha)t + p_n \ . \tag{2.35}$$

Case 2.1: The online schedule contains a point after t where at least 4 jobs are scheduled in parallel.

The job k, which is the first job scheduled in parallel with three other jobs, has $m_k \leq \frac{m}{4}$. Since jobs appear with non-increasing m_j, the number of machines in use between t and the start of job k is non-increasing at the moment job k is scheduled. The load in $[t, s)$ is therefore at least $\frac{3m}{4}\alpha t$. Together with the fact that the load in $[0, t)$ is at least $\frac{2m}{3}t$, we get $C^*(\sigma) \geq (\frac{2}{3} + \frac{3}{4}\alpha)t$. Incorporating this bound and the length bound in (2.35) yields

$$C_{\mathrm{MG}}(\sigma) \quad \leq \quad \left(\frac{1+\alpha}{\frac{2}{3}+\frac{3}{4}\alpha} + 1\right) C^*(\sigma) \ . \tag{2.36}$$

On the other hand, incorporating Lemma 2.17 and the length bound in (2.35) yields

$$C_{\mathrm{MG}}(\sigma) \quad \leq \quad \left((1+\alpha)\frac{4}{3} + 1\right) C^*(\sigma) \ . \tag{2.37}$$

If $\alpha \geq \frac{1}{9}$ then by (2.36) we have

$$C_{\mathrm{MG}}(\sigma) \leq \left(\frac{1+\frac{1}{9}}{\frac{2}{3}+\frac{3}{4}\frac{1}{9}} + 1\right) C^*(\sigma) = \frac{67}{27} C^*(\sigma) \ ,$$

and if $\alpha \leq \frac{1}{9}$ then by (2.37) we have

$$C_{\mathrm{MG}}(\sigma) \leq \left((1+\frac{1}{9})\frac{4}{3} + 1\right) C^*(\sigma) = \frac{67}{27} C^*(\sigma) \ .$$

Case 2.2: In the online schedule there are no 4 jobs scheduled in parallel after t.

In this case we use a load argument where we take the load of p_n into account as well. The load in $[0, t)$ is at least $\frac{2m}{3}t$, and in $[t, s)$ the load is at least $(m - m_n)(s - t)$, since the machine usage after t is non-increasing. In $[s, C_{\mathrm{MG}})$ the load is at least $m_n p_n$. Thus,

$$C^*(\sigma) \quad \geq \quad \frac{2}{3}t + \frac{m - m_n}{m}(s - t) + \frac{m_n}{m} p_n \ . \tag{2.38}$$

If $s - t \leq p_n$ the bound (2.38) becomes $C^*(\sigma) \geq s - \frac{1}{3}t$. Thus,

$$\begin{aligned} C_{\mathrm{MG}}(\sigma) \quad &= \quad s + p_n \quad = \quad s - \frac{1}{3}t + \frac{1}{3}t + p_n \\ &\leq \quad 2C^* + \frac{1}{3}t \quad \leq_{(Lem.\ 2.17)} \quad \frac{22}{9} C^*(\sigma) \ . \end{aligned}$$

Now consider the case that $s-t \geq p_n$. Since job n is small, the load in $[0, s-p_n)$ is at least $\frac{2m}{3}(s-p_n)$, and the load in $[s-p_n)$ plus the load of job n is at least mp_n. Therefore, we get

$$C^*(\sigma) \quad \geq \quad \frac{2}{3}(s-p_n)+p_n \quad \geq \quad \frac{2}{3}s+\frac{1}{3}p_n \ . \tag{2.39}$$

Thus,

$$C_{\mathrm{MG}}(\sigma) \quad = \quad s+p_n \quad \leq_{(2.39)} \quad \frac{3}{2}C^*(\sigma)+\frac{1}{2}p_n \quad \leq 2C^*(\sigma) \ .$$

Summarizing, in all the cases the Algorithm MG is at most $\frac{67}{27}$-competitive. □

2.6.3 Non-increasing m_j: small number of machines

In the following we consider the semi-online case of $Pm|\mathrm{online}-\mathrm{list}, m_j|C_{\max}$ where jobs appear in non-increasing order of machine requirement and the number of available machines is either 2, 3, 4 or 5.

Theorem 2.19. *For the* 2 *and the* 3*-machine problem the greedy algorithm is* $(2-\frac{1}{m})$*-competitive if jobs appear in non-increasing order of machine requirement, and this is the best possible.*

Proof. For the 2-machine problem, no worst case example contains a job with $m_j = 2$, since removing this job decreases the online and optimal makespan by the same amount. If no jobs with $m_j = 2$ appear, the greedy algorithm is equal to the list scheduling algorithm for the classical parallel machine scheduling problem, which is $(2-\frac{1}{m})$-competitive.

For the 3-machine problem, no worst case example contains jobs with $m_j = 3$. All jobs with $m_j = 2$ are scheduled one after the other on two machines. Afterwards, jobs with $m_j = 1$ are first scheduled parallel to the jobs with $m_j = 2$. Again, the performance of the greedy algorithm is equal to the list scheduling algorithm for the classical parallel machine scheduling problem, which is $(2-\frac{1}{m})$-competitive.

The greedy algorithm is also best possible, since list scheduling is $(2-\frac{1}{m})$-competitive and best possible for the classical parallel machine schedule with $m \leq 3$. □

Theorem 2.19 shows that if $m \leq 3$ the parallelism of jobs is of no importance when jobs appear in non-increasing order of m_j. For more than 3 machines the machine requirement does play a role, i.e. the greedy algorithm has competitive ratio larger than $2-\frac{1}{m}$.

Theorem 2.20. *If* $m \geq 4$ *and jobs appear in non-increasing order of machine requirement, then the greedy algorithm has competitive ratio at least* ≥ 2*.*

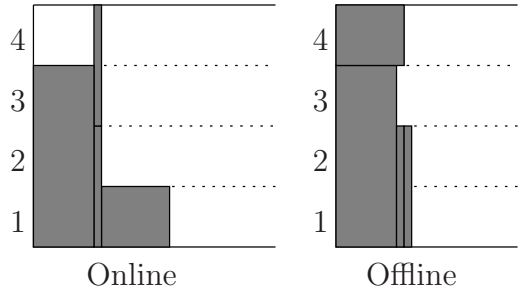


Figure 2.16: Counter example for $(2-\epsilon)$-competitiveness.

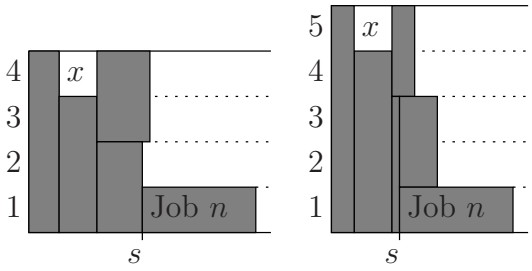


Figure 2.17: Greedy for $m = 4$ and $m = 5$.

Proof. Suppose the greedy algorithm has ratio $2-\epsilon$. Consider the following sequence σ of 4 jobs. Job 1 has $p_1 = 1$ and $m_1 = m - 1$, job 2 has $p_2 = \frac{\epsilon}{2}$ and $m_2 = \lceil \frac{m}{2} \rceil$, job 3 has $p_3 = \frac{\epsilon}{2}$ and $m_3 = \lfloor \frac{m}{2} \rfloor$, and job 4 has $p_4 = 1 + \frac{\epsilon}{2}$ and $m_4 = 1$. The greedy algorithm schedules job 2 and 3 in parallel, where in the optimum jobs 1 and 4 are in parallel (see Figure 2.16 for the case $m = 4$).

Therefore, $C_{\text{Greedy}}(\sigma) = 2 + \epsilon$ and $C^*(\sigma) = 1 + \epsilon$, and the resulting competitive ratio is:

$$\frac{C_{\text{Greedy}}(\sigma)}{C^*(\sigma)} = \frac{2+\epsilon}{1+\epsilon} = 2 - \frac{\epsilon}{1+\epsilon} > 2 - \epsilon \ ,$$

yielding the required contradiction. □

Theorem 2.21. *For the* 4 *and* 5*-machine problem, the greedy algorithm is* 2*-competitive if jobs appear in non-increasing order of machine requirement.*

Proof. For $m = 4$, the schedule created by Greedy contains first the jobs with $m_j = 4$, then jobs with $m_j = 3$ followed by jobs with $m_j = 2$ in parallel. Denote the length of this interval, where jobs with $m_j = 3$ are scheduled, without any job with $m_j = 1$

in parallel, by x (see Figure 2.17). Let job n be the job determining the makespan of the online schedule, and s its start time. Again assume this is the last job in the sequence. We only have to consider the case with $m_n = 1$, because if $m_n = 2$ the resulting competitive ratio is $\frac{3}{2}$. By definition, x is smaller than all jobs with $m_j = 1$ scheduled after x, e.g. $x \leq p_n$. Therefore, the total load of all jobs is at least $4s$, implying that $s \leq C^*(\sigma)$. This leads to

$$C_{\text{Greedy}}(\sigma) = s + p_n \leq 2C^*(\sigma) \ ,$$

proving the theorem for $m = 4$.

Similar for $m = 5$, there is one interval with machine usage 4 before the start of job n, and this interval is shorter than p_n (see Figure 2.17). By the same arguments as for the case with $m = 4$ we obtain that the greedy algorithm is 2-competitive if $m = 5$. □

Theorem 2.22. *Algorithm MG is 2-competitive if $m \leq 5$ and jobs appear in non-increasing order of machine requirement.*

Proof. Since the only jobs with $m_j \leq \frac{m}{3}$ are the jobs with $m_j = 1$, only these are scheduled in a greedy fashion. All other jobs are scheduled in series. Before any $m_j = 1$ job appears, MG is 2-competitive since no more than 2 jobs can be scheduled in parallel in the optimum. As soon as jobs with $m_j = 1$ appear the machine usage is monotone decreasing, thus there is no *gap* in the schedule. So, the online schedule has a load of at least $m \cdot s$ in $[0, s]$, where s is the start time of the last job. This implies $(2 - \frac{1}{m})$-competitiveness. □

The most intuitive choice of algorithm for $P4|\text{online} - \text{list}, m_j|C_{\max}$ is the greedy algorithm. However, the greedy algorithm and also the Algorithm MG are exactly 2-competitive. It seems that if we want to improve upon Greedy and MG, we need to focus on the way jobs with $m_j = 2$ are scheduled. Informally, the jobs with other machine requirements can be scheduled greedily without hurting the performance too much, and two jobs of $m_j = 2$ can create a gap and thereby disturbing the monotone decreasing machine usage of the online schedule.

Suppose we schedule jobs with $m_j > 2$ in a greedy fashion and we are now considering the first job with $m_j = 2$. For this job we have no choice but to schedule it greedily. For the successive $m_j = 2$ jobs we do have a choice, either schedule it greedy or sequential. As soon as we schedule one of them greedily, it makes sense that we schedule the remainder also greedily. These considerations result in the following algorithm.

Algorithm $4M(\lambda)$**:**

- Schedule the jobs with $m_j = 4$, 3 and 1 in a greedy fashion.
- Schedule the jobs with $m_j = 2$ one after the other, unless the cumulative length of the jobs with $m_j = 3$ is less then λ times the cumulative length of the jobs with $m_j = 2$ (including the current job), then schedule the job in a greedy fashion.

This algorithm cannot have a competitive ratio less than $\frac{7}{4}$, since if only jobs with $m_j = 1$ appear the algorithm is equal to list scheduling. To minimize the competitive ratio, we choose $\lambda = 3 + \sqrt{17} \approx 7.123$.

Theorem 2.23. *Algorithm* $4M(\lambda)$ *is* 1.7808*-competitive with* $\lambda = 3 + \sqrt{17}$*, and this ratio is tight.*

Proof. For the Algorithm $4M(\lambda)$ no worst case instance contains a job with $m_j = 4$, since removing such a job decreases the online and optimal offline makespan by exactly the same amount. If no job with machine requirement of 3 arrives or the second job has a machine requirement of 1, then all jobs are scheduled greedily and there is no gap in the schedule, i.e. the algorithm is $\frac{7}{4}$-competitive. So, we can assume that there are jobs with machine requirement of 2 in the job sequence. Additionally, we can assume w.l.o.g. that the cumulative processing time of the jobs having machine requirement 3 is exactly 1.

Let σ be such an instance, and let job n be the last completing job in the online schedule with start time s. Again we assume that job n is the last job in the sequence σ. In the following, let y be the cumulative length of the jobs with $m_j = 2$.

Case 1: No two jobs with $m_j = 2$ are scheduled in parallel, i.e. $y \leq \lambda$.

If the machine requirement of the last job is 1, $m_n = 1$, the Algorithm $4M(\lambda)$ is $\frac{7}{4}$-competitive, since the online schedule contains no idle machines before s. If $m_n = 2$, we know that $y \leq \lambda$ and the optimal offline makespan is no less than $C^*(\sigma) \geq 1 + \frac{y}{2}$. This implies,

$$\begin{aligned} C_{4M(\lambda)}(\sigma) &= s + p_n = 1 + y \\ &\leq \frac{1+y}{1+\frac{y}{2}} C^*(\sigma) \\ &\leq_{(y \leq \lambda)} \left(1 + \frac{\lambda}{2+\lambda}\right) C^*(\sigma) \ . \end{aligned}$$

Case 2: Two jobs with $m_j = 2$ are in parallel, i.e. $y > \lambda$.

If $m_n = 2$, then $C^*(\sigma) \geq 1 + \frac{y}{2}$ and $C^*(\sigma) \geq p_n$, leading to

$$\begin{aligned} C_{4M(\lambda)}(\sigma) &= s + p_n \leq \left(1 + \frac{y - p_n}{2}\right) + p_n \\ &= 1 + \frac{y}{2} + \frac{p_n}{2} \leq \frac{3}{2} C^*(\sigma) \ . \end{aligned}$$

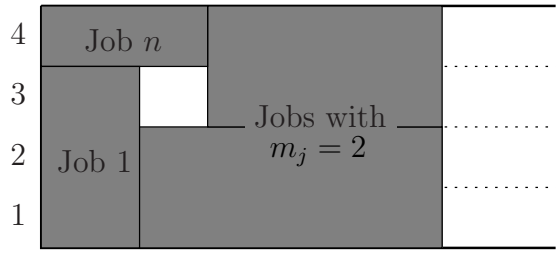


Figure 2.18: Sketch of optimal schedule (Case 2 of Theorem 2.23).

If $m_n = 1$, we use a load argument. Note that there is at most one *gap* in the schedule, and that this gap is parallel to the jobs with machine requirement of 3. Therefore, the machine usage in $[0, 1)$ is at least 3 and in $[1, s)$ exactly 4. This load argument gives $C^*(\sigma) \geq \frac{1}{4}(3 + 4(s-1) + p_n) = s - \frac{1}{4} + \frac{1}{4}p_n$. If $p_n \leq 1$ we use this bound together with $C^*(\sigma) \geq 1 + \frac{\lambda}{2}$ to get

$$\begin{aligned}
C_{4\mathrm{M}(\lambda)}(\sigma) &= s + p_n = \left(s - \frac{1}{4} + \frac{1}{4}p_n\right) + \frac{1}{4} + \frac{3}{4}p_n \\
&\leq C^*(\sigma) + \frac{1}{4} + \frac{3}{4}p_n \leq_{(p_n \leq 1)} C^*(\sigma) + 1 \\
&\leq \left(1 + \frac{1}{1 + \frac{\lambda}{2}}\right) C^*(\sigma) \\
&\leq_{(\lambda = 3 + \sqrt{17})} 1.22 C^*(\sigma) \ .
\end{aligned}$$

If $p_n > 1$, the optimal offline makespan is also bounded by $C^*(\sigma) \geq \frac{1}{2}(\lambda + 1 + p_n)$, as sketched in Figure 2.18. This last bound is equivalent to

$$\frac{1}{4} \leq \frac{2C^*(\sigma) - p_n}{4(\lambda + 1)} \ .$$

Thus,

$$\begin{aligned}
C_{4M(\lambda)}(\sigma) &= s + p_n \leq C^*(\sigma) + \frac{1}{4} + \frac{3}{4}p_n \\
&\leq C^*(\sigma) + \frac{2C^*(\sigma) - p_n}{4(\lambda + 1)} + \frac{3}{4}p_n \\
&\leq_{(C^*(\sigma) \geq p_n)} C^*(\sigma) + \frac{C^*(\sigma)}{2(\lambda + 1)} + \left(\frac{3}{4} - \frac{1}{4(\lambda + 1)}\right) C^*(\sigma) \\
&= \frac{7\lambda + 8}{4\lambda + 4} C^*(\sigma) \ .
\end{aligned}$$

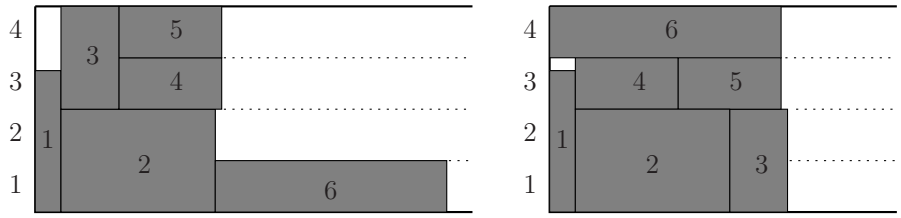


Figure 2.19: Online and offline schedule for tight instance for $4M(\lambda)$.

By choosing $\lambda = 3 + \sqrt{17}$ we have $1 + \frac{\lambda}{2+\lambda} = \frac{7\lambda+8}{4\lambda+4}$ and therefore obtain the best possible performance of $4M(\lambda)$. With this choice for λ, the competitive ratio is $\frac{C_{4M(\lambda)}(\sigma)}{C^*(\sigma)} \leq \frac{2(4+\sqrt{17})}{5+\sqrt{17}} \approx 1.7808$.

The following instance shows that the analysis is tight. Consider the sequence of 6 jobs with characteristics given in Table 2.5 and the resulting online and offline schedules displayed in Figure 2.19, where ϵ is an arbitrary small positive amount. The makespan of the online schedule is $2 + \frac{7}{4}\lambda$ while the makespan of the optimal offline schedule is only $\lambda + 1 + \epsilon$. Thus, the competitive ratio of $4M(\lambda)$ is at least $\frac{2+\frac{7}{4}\lambda}{\lambda+1} \approx 1.7808$. □

job j	1	2	3	4	5	6
p_j	1	$\frac{3}{4}\lambda$	$\frac{1}{4}\lambda + \epsilon$	$\frac{1}{2}\lambda$	$\frac{1}{2}\lambda$	$\lambda + 1$
m_j	3	2	2	1	1	1

Table 2.5: Lower bound on Algorithm $4M(\lambda)$.

2.7 Concluding remarks

In this chapter many of the existing bound on the competitive ratio for the various special cases of $P|\text{online} - \text{list}, m_j|C_{\max}$ have been improved. However, in particular for the general problem the gap between the lower bound (2.43) and the upper bound (6.6623) on the competitive ratio is still large. Only for the case with two machines the tractability of the problem is resolved.

For the case with three machines a new algorithm is presented. To beat the 3-competitive greedy algorithm, a delay for some of the jobs which require all machines for processing has been introduced. We believe that for the three machine case neither the lower bound of 2 nor the 2.8-competitive algorithm are best possible.

With the knowledge that jobs appear with non-increasing m_j, better competitive ratios are obtained. Given that jobs are non-increasing in m_j it seems important to

have a machine usage that is decreasing. This prevents the occurrence of gaps in the online schedules.

The presented algorithm and lower bound construction for the general online parallel job scheduling problem also apply to the online orthogonal strip packing. It is an interesting open question whether or not the additional requirement of a line ordering will lead to a different competitive ratio of the problem.

For future research it would be interesting to find a proof for Conjecture 2.10; the lower bound of 2.5 for any online algorithm for the general problem. By Theorem 2.9, we know this is the best result one can hope for with the presented instance construction. So, in order to obtain a higher lower bound, complete new ideas for the adversary instance construction are necessary.

3 Batching machine online

3.1 Introduction

In this chapter we consider online-list scheduling on a batching machine. Using the three field notation for scheduling problems [30], this problem is denoted by $1|\text{online}-\text{list}, p-\text{batch}, B < n|C_{\max}$. A set of jobs has to be scheduled on the batching machine which processes jobs parallel in batches. Each job j is characterized by its processing time p_j. The batching machine has capacity B, which gives the maximum number of jobs that can be scheduled in a single batch. The processing time of a batch must be larger or equal to the maximum processing time of all jobs in the batch. The objective is to minimize the makespan, that is the completion time of the last batch. Note that the order of the batches is of no importance, it does not influence the objective function, only the processing times of the created batches are of interest. The above type of batching is referred to as parallel batching or *p-batch*, contrary to an *s-batch* which processes jobs sequential with a start-up time for each batch [8]. The model of parallel batching finds applications in, for example, scheduling burn-in ovens used for circuit board manufacturing [55].

In the *online-list* version of this problem jobs from a sequence σ are presented one by one to the scheduler. Upon arrival, the processing time of the job becomes known and the job has to be assigned immediately and irrevocably to a batch. The job is either included in a non-full existing batch (i.e. a batch with less than B jobs assigned to it) or a new batch is created for this job. The processing time of each batch has to be fixed upon its creation, and a job j can only be assigned to a batch with processing time at least p_j.

In the corresponding offline problem, the scheduler has all jobs available at $t = 0$, and an optimal offline schedule can be found by applying the algorithm known as FBLPT (Full Batch Longest Processing Time) [54]. This algorithm schedules the

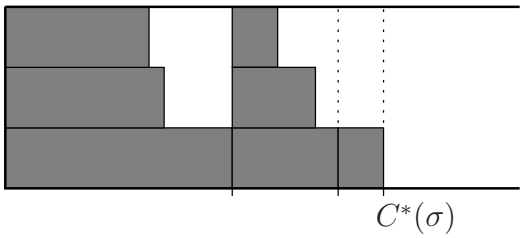


Figure 3.1: Applying FBLPT.

B jobs with largest processing time in the first batch, the next B jobs with largest processing time in a second batch, etc. The structure of a resulting scheduling is sketched in Figure 3.1.

For a sequence of jobs σ, we denote the makespan of the optimal offline schedule by $C^*(\sigma)$ and the makespan of the online schedule created by an online Algorithm A by $C^A(\sigma)$. The performance of an online Algorithm A is given by its competitive ratio defined as $\sup_\sigma\{C^A(\sigma)/C^*(\sigma)\}$. An online algorithm is called optimal if it has the smallest possible competitive ratio among all online algorithms.

In the literature only related problems have been studied. In [4] the *online-list* batching problem with the objective to minimize the average flow time is studied and an optimal 4-competitive algorithm is given. The considered model allows only to schedule the next job in the last created batch or to create a new batch, and the capacity of the batching machine is unlimited. Much more work has been done on the *online-time* version of the batching problem to minimize the makespan, where jobs arrive according to their release date. For the unlimited capacity case, optimal $\left(\sqrt{5}+1\right)/2$-competitive algorithms were given in [19] and [84] and generalized in [68]. The tractability of the online-time problem has not yet been resolved for bounded capacity. The best known online algorithm is a 2-competitive algorithm for any capacity B [69]. Only for the case $B = 2$ a better algorithm is presented in [69], which is 7/4-competitive. An algorithm proposed in [84] for the bounded capacity online-time problem is conjectured to be optimally $(\sqrt{5}+1)/2$-competitive. Although this conjecture is not valid it is still being supported in the literature, e.g. [71]. We refer to [63] for the more general problem with job families and a more extensive overview of the results on the online-time model.

The algorithms designed in this chapter use what is called the "doubling" strategy. The idea behind this strategy is to use geometrically increasing batch processing times to approximate the optimal offline solution. However, as mentioned in [12], the increase is not always done by a factor of 2. Online algorithms designed with this principle can be found, for example, in the literature on the problem of *searching a line in the plane* [1]. A short overview of other online problems solved with the "doubling" strategy can be found in [12].

The outline of this chapter is as follows. Section 3.2 presents an optimal 4-competitive algorithm in case of unlimited capacity. Optimality follows from the known bounds for the *online bidding problem*, which is basically the same problem. The online bidding problem is a folklore online problem, for example described in [13]. Section 3.3 deals with the bounded capacity case and contains the main contribution of the chapter: For any given capacity B we derive an optimal online algorithm. With these results, the tractability of online-list batch scheduling is settled.

3.2 Unlimited capacity and online bidding

In this section we consider the case of unlimited batch capacity and show an optimal 4-competitive algorithm. In case of unlimited batch capacity the optimal offline schedule has all jobs in one and the same batch of length equal to the largest processing time. So, the optimal offline makespan is given by $C^*(\sigma) = \max_{j\in\sigma}\{p_j\}$.

With unlimited batch capacity, the online-list batch scheduling problem is the same as the folklore *online bidding problem*, see e.g. [13]. The online bidding problem is stated as follows: An online player submits *bids* b_i until it submits a bid larger than or equal to a threshold $T \geq 1$. The online player pays the sum of all submitted bids. It is not difficult to see that these problems are equivalent. The *online scheduler* determines a sequence of batch lengths $b_1 < b_2 < \cdots < b_{k-1} < b_k$ such that $b_{k-1} < p_{\max} \leq b_k$, and has makespan $\sum_{i=1}^{k} b_i$. Since the batch capacity is unlimited, no reasonable algorithm creates a batch for an arriving job that can be included in an existing batch. The *online bidder* determines a sequence of bids $b_1 < b_2 < \cdots < b_{k-1} < b_k$ such that $b_{k-1} < T \leq b_k$, and pays $\sum_{i=1}^{k} b_i$. Again, no reasonable bidder submits a bid smaller than the previous bid. The offline costs are $p_{\max}$ and T for the scheduling and bidding problem, respectively.

To deal with this online problem we propose the following "doubling" algorithm.

Algorithm A^∞:

Schedule a job j with processing time $p_j \in \left(2^{i-1}, 2^i\right]$ in a batch of length 2^i. If no such batch exists, create it at the end of the current schedule.

The idea behind this algorithm is the same as behind all "doubling" type online algorithms. Even if the algorithm is forced to construct many different batches, we know that there exists a *relatively* long job compared to the total batch length. Informally, there is a growth rate of 2 in the batch lengths. This leads to the following performance guarantee of A^∞.

Theorem 3.1. *For online-list batch scheduling with unlimited capacity, Algorithm A^∞ is 4-competitive.*

Proof. For any sequence of jobs σ, Algorithm A^∞ creates for jobs with lengths in $\left(2^{i-1}, 2^i\right]$ at most one batch of length 2^i. By normalizing, we let the smallest batch

have length 2. Now, if the largest batch created for sequence σ has length 2^n, then

$$C^*(\sigma) = \max_{j \in \sigma}\{p_j\} > 2^{n-1}$$

and

$$C^{A^\infty}(\sigma) \leq \sum_{i=1}^{n} 2^i = 2^{n+1} - 2 \ .$$

The competitive ratio of A^∞ is bounded by

$$\frac{C^{A^\infty}(\sigma)}{C^*(\sigma)} < \frac{2^{n+1}-2}{2^{n-1}} = 4 - \frac{1}{2^{n-2}} < 4 \ .$$

Thus, Algorithm A^∞ is 4-competitive. □

To show that Algorithm A^∞ has the best possible competitive ratio, we consider one special adversary job sequence:

Definition. *The infinite job sequence σ^{adv} has $p_1 = 1$ and each following job has length equal to the last created batch by the online algorithm plus a small amount $\epsilon > 0$. The subsequence σ_k^{adv} is given by the first k jobs of sequence σ^{adv}.*

Job sequence σ^{adv} depends on the online algorithm used, but any online algorithm must create a new batch for each new arriving job. Intuitively, this adversary is the strongest possible for both the bounded capacity and unlimited capacity case, meaning that it creates for any online algorithm the worst case instance. Therefore we use this sequence of jobs throughout the chapter to prove lower bounds on the competitive ratio and performance of online algorithms. With this adversary we have $p_{\max} = b_{k-1} + \epsilon$ if the sequence stops after job k. (In online bidding this means $T = b_{k-1} + \epsilon$.)

For online bidding, it is known that no bidding strategy is better than 4-competitive [13]. This implies that no algorithm for online-list batch scheduling with unlimited capacity can have a competitive ratio less than 4. However, for completeness of this chapter, we include the proof of optimality of Algorithm A^∞ and adopt it from [13].

Theorem 3.2. *For online-list batch scheduling with unlimited capacity, no online algorithm is $(4-\delta)$-competitive, for any $\delta > 0$.*

Proof. Suppose there exists a $(4-\delta)$-competitive Algorithm A, and this algorithm is presented with job sequence σ^{adv}. For simplicity we denote the optimal offline makespan by C_k^* and the online makespan of Algorithm A by C_k^A for the subsequence σ_k^{adv}. All what follows is subject to the small value ϵ in the construction of σ^{adv}, but by choosing ϵ appropriately small it does not affect the outcome. So, we choose ϵ small enough and leave it from the remainder of the analysis.

Define $\gamma_k = \frac{C_{k+1}^A}{C_k^A}$. The assumption on the competitiveness of Algorithm A gives $C_{k+1}^A \leq (4-\delta) \cdot C_{k+1}^*$, which, by definition of σ^{adv} (see Figure 3.2), can be rewritten

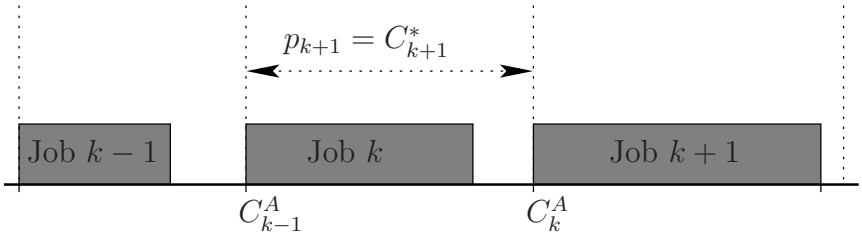


Figure 3.2: Part of an online schedule for σ^{adv}.

as

$$C^A_{k+1} \leq (4-\delta) \cdot \left(C^A_k - C^A_{k-1}\right) \quad .$$

Dividing this inequality by C^A_k gives a recursion on γ_k:

$$\gamma_{k+1} \leq (4-\delta) \cdot \left(1 - \frac{1}{\gamma_k}\right) \quad .$$

Since $1 - \frac{1}{x} \leq \frac{x}{4}$, this implies $\gamma_{k+1} \leq (4-\delta) \cdot \frac{\gamma_k}{4}$. Thus $\gamma_k \leq \left(\frac{4-\delta}{4}\right)^k \gamma_0$, and so eventually $C^A_{k+1} < C^A_k$, which is a contradiction. □

As a consequence of Theorem 3.2 we get that Algorithm A^∞ is an optimal online algorithm.

3.3 Bounded capacity

In the previous section we have seen how the doubling strategy leads to an optimal algorithm if the batch capacity is unlimited. In this section we consider online-list batch scheduling with a fixed bounded capacity B for each batch. To obtain an optimal algorithm for the bounded case, we have to use a different growth rate in batch lengths (different for each capacity B). However, the basic structure of the scheduling algorithm is the same as in Algorithm A^∞.

Concrete, we propose the following online algorithm for the online-list batch scheduling problem with capacity B. If $B \leq 3$ we schedule the jobs greedily. If $B \geq 4$, we use a growth rate of z_B in batch lengths instead of the rate 2 for the unlimited capacity case.

B	2	3	4	5	6	7	8	∞
z_B	1	1	1.5214	1.7614	1.8768	1.9349	1.9651	2
ρ_B	2	3	3.6107	3.8344	3.9254	3.9651	3.9833	4

Table 3.1: Values of z_B and ρ_B

Algorithm A^B:

If $B \leq 3$, then schedule a job j with processing time p_j in a non-full batch of length at least p_j. If no such batch exists, create a batch with length p_j at the end of the current schedule.
If $B \geq 4$, then schedule a job j with processing time $p_j \in \left(z_B^{i-1}, z_B^i\right]$ in a non-full batch of length z_B^i. If no such batch exists, create it at the end of the current schedule.

We choose z_B such that

$$z_B = \text{argmin}_{x \geq 1}\{x + 1 + \frac{1}{x} + \frac{1}{x^2} + \cdots + \frac{1}{x^{B-2}}\} \ , \tag{3.1}$$

and show that the competitive ratio of Algorithm A^B is

$$\rho_B = \min_{x \geq 1}\{x + 1 + \frac{1}{x} + \frac{1}{x^2} + \cdots + \frac{1}{x^{B-2}}\} \ .$$

Before we determine the competitive ratio of A^B, we point out that z_B and ρ_B are unique. There is only one minimum in (3.1), since the derivative $1 - \frac{1}{x^2} - \frac{2}{x^3} - \cdots - \frac{B-2}{x^{B-3}}$ is increasing in x for $x \geq 1$. To indicate what kind of growth rates and competitive ratios we are dealing with, we display in Table 3.1 the values of z_B and ρ_B for some specific values of B.

Theorem 3.3. *For online-list batch scheduling with capacity B, Algorithm A^B is ρ_B-competitive.*

Proof. For $B \leq 3$, we know that each batch in the online schedule contains at least one job with processing time equal to the length of the batch. So, by a load argument the offline makespan cannot be less than $\frac{1}{B}$ times the online makespan. Thus, Algorithm A^B is B-competitive.

Consider $B \geq 4$. Let σ be a worst-case instance for Algorithm A^B. By normalizing the job lengths let z_B^1 be the smallest online batch and n such that z_B^n is the largest online batch. Thus the online schedule consists of batches with lengths in $\{z_B^1, z_B^2, \ldots, z_B^n\}$. Note that for each i there is at most one non-full batch of length z_B^i. In the following we derive three properties which we may assume for worst-case instance σ:

1. Each job j scheduled in a batch of length z_B^i has length $p_j = z_B^{i-1} + \epsilon$, with $\epsilon > 0$ arbitrary small.
   Decreasing the job lengths in a batch of length z_B^i to $z_B^{i-1} + \epsilon$ does not affect the online makespan and may decreases the offline makespan. As in the proof of Theorem 3.2, we ignoring ϵ from now on.

2. For each batch length z_B^i, there is at most one batch.
   If the worst case instance has more than B jobs in batches of length z_B^i, then B of these jobs are together in a batch in both the online and offline schedule. Due to property 1, removing these B jobs causes a decrease of z_B^i in the online makespan and a decrease of z_B^{i-1} in the optimal offline makespan. Let $\tilde{\sigma}$ be the instance resulting by removal of these B jobs from σ. Since σ is a worst-case instance we have

$$\frac{C^A(\sigma)}{C^*(\sigma)} \geq \frac{C^A(\tilde{\sigma})}{C^*(\tilde{\sigma})} = \frac{C^A(\sigma) - z_B^i}{C^*(\sigma) - z_B^{i-1}} \quad .$$

   This implies that $z_B \cdot C^*(\sigma) \geq C^A(\sigma)$, and that the algorithm has competitive ratio of at most $z_B < \rho_B$. So, we only have to consider instances which result in an online schedule with for each batch length z_B^i at most one batch.

3. Each batch consists of only one job.
   If the only batch of length z_B^i contains k jobs with $2 \leq k \leq B$, then we can remove $k-1$ of these jobs without decreasing the online makespan and possibly decrease the offline makespan.

By the above properties of σ, we get that the cumulative length of the B largest batches in the online schedule is at most $z_B^n + z_B^{n-1} + \cdots + z_B^{n-B+1}$. By (3.1) this is equal to $\rho_B \cdot z_B^{n-1}$, that is ρ_B times the largest offline batch. This argument can be repeated for the next B largest batches in the online schedule. They are at most ρ_B times the second largest offline batch, etc. Thus, Algorithm A^B is ρ_B-competitive. □

It remains to show the optimality of Algorithm A^B. For such a proof, the known results on online bidding are of no use since they need the unlimited capacity of the batches. More precisely, the change in the offline cost structure makes the comparison with online bidding impossible and complicates the analysis. Where the offline makespan is just $p_{\max}$ for the unlimited capacity case, we now have to consider the FBLPT solution. The next theorem, which we consider the main contribution of this chapter, uses the structure of the FBLPT solution to prove that Algorithm A^B is optimal. In fact, the next theorem also implies Theorem 3.2 by letting B go to infinity.

Theorem 3.4. *For online-list batch scheduling with capacity B, no online algorithm is $(\rho_B - \delta)$-competitive, for any $\delta > 0$ and B.*

Proof. Suppose there exists a $(\rho_B - \delta)$-competitive Algorithm A, and this algorithm is presented with job sequence σ^{adv}. Recall that due to the construction of σ^{adv} each job has its own batch in the online schedule regardless of the online algorithm used. As in Theorem 3.2, we choose the ϵ in the instance construction small enough to ignore it. Again, for simplicity we denote the optimal offline makespan by C_k^* and the online makespan of Algorithm A by C_k^A for the subsequence σ_k^{adv}.

The optimal offline and online makespan can be expressed by

$$\begin{aligned} C_k^* &= p_k + p_{k-B} + p_{k-2B} + \dots \\ C_k^A &= p_{k+1} + p_k + \dots + p_2 \\ &= C_{k+1}^* + C_k^* + \dots + C_{k-B+2}^* - C_1^* \ . \end{aligned}$$

Let $\gamma_k = \frac{C_{k+1}^*}{C_k^*}$, be the ratio between the value of two subsequent optimal offline makespans. Note that this is a different ratio from the one used in Theorem 3.2. Obviously the optimal offline makespan increases in k, thus $\gamma_k \geq 1$. By Algorithm A being $(\rho_B - \delta)$-competitive, we have

$$\begin{aligned} \frac{C_k^A}{C_k^*} &= \frac{C_{k+1}^* + C_k^* + \dots + C_{k-B+2}^* - C_1^*}{C_k^*} \\ &= \gamma_k + 1 + \frac{1}{\gamma_{k-1}} + \frac{1}{\gamma_{k-1}\gamma_{k-2}} + \dots + \frac{1}{\gamma_{k-1}\gamma_{k-2}\cdots\gamma_{k-B+2}} - \frac{C_1^A}{C_k^*} \\ &\leq \rho_B - \delta \ . \end{aligned}$$

We assume k to be large enough such that $\frac{C_1^*}{C_k^*} \leq \frac{\delta}{2}$, thus

$$\gamma_k + 1 + \frac{1}{\gamma_{k-1}} + \frac{1}{\gamma_{k-1}\gamma_{k-2}} + \dots + \frac{1}{\gamma_{k-1}\gamma_{k-2}\cdots\gamma_{k-B+2}} \leq \rho_B - \frac{\delta}{2} \ . \tag{3.2}$$

In the remainder of this proof we show that (3.2) and $\gamma_k \geq 1$ are contradicting. To obtain this contradiction, we introduce $\tilde{\gamma}_k := \max\{\gamma_{k-1}, \dots, \gamma_{k-B+2}\}$ and show that $\gamma_k < \tilde{\gamma_k}$ and $\tilde{\gamma}_k$ decreases below 1.

By (3.2) and the definition of $\tilde{\gamma}_k$ we have

$$\begin{aligned} \rho_B - \frac{\delta}{2} &\geq \gamma_k + 1 + \frac{1}{\gamma_{k-1}} + \frac{1}{\gamma_{k-1}\gamma_{k-2}} + \dots + \frac{1}{\gamma_{k-1}\gamma_{k-2}\cdots\gamma_{k-B+2}} \\ &\geq \gamma_k + 1 + \frac{1}{\tilde{\gamma}_k} + \frac{1}{\tilde{\gamma}_k^2} + \dots + \frac{1}{\tilde{\gamma}_k^{B-2}} \ . \end{aligned} \tag{3.3}$$

Since z_B minimizes $x+1+\frac{1}{x}+\frac{1}{x^2}+\dots+\frac{1}{x^{B-2}}$ and the minimum is ρ_B, we have $\gamma_k < \tilde{\gamma}_k$. This can be seen by assuming $\gamma_k \geq \tilde{\gamma}_k$. The value of γ_k can be decreased to $\tilde{\gamma}_k$ without violating (3.3). So, this would yield a better minimum for $x+1+\frac{1}{x}+\frac{1}{x^2}+\dots+\frac{1}{x^{B-2}}$ than z_B does.

As a direct consequence of $\gamma_k < \max\{\gamma_{k-1}, \ldots, \gamma_{k-B+2}\}$, we get by induction $\tilde{\gamma}_k < \tilde{\gamma}_{k-B+2}$. Now assume that $\tilde{\gamma}_k$ converges to some y. Then equation (3.2) holds when all γ_i's are substituted by y, implying that y gives a better minimum in (3.1) than z_B does. Thus, $\tilde{\gamma}_k$ cannot converge.

By the above we have that the value $\tilde{\gamma}_k$ is an upper bound on γ_k and decreases below any fixed value. Thus, eventually $\gamma_k < \tilde{\gamma}_k \leq 1$, contradicting $C^*_{k+1} \geq C^*_k$. □

By Theorems 3.3 and 3.4, we obtain the optimality of online Algorithm A^B. From the proof of Theorem 3.4 we see that any optimal online algorithm presented with σ^{adv} must behave like Algorithm A^B as k grows large. No matter which optimal algorithm is used, the upper bound $\tilde{\gamma}_k$ must converge to z_B. In order to let $\tilde{\gamma}_k$ converge to z_B the value γ_k must converge to z_B. Therefore, as k grows large the batch size has growth rate z_B.

3.4 Concluding remarks

This chapter presents an optimal online algorithm for online-list batch scheduling with any batch capacity B. Thus, the tractability of $1|\text{online} - \text{list}, p - \text{batch}, B < n|C_{\max}$ is settled. For batch capacity $B \leq 3$ this algorithm is a greedy type algorithm, i.e. each batch has the same length as the first job scheduled in it. As B goes to infinity the growth rate z_B in the proposed online algorithm goes to 2 and its competitive ratio ρ_B to 4. Therefore, the known results for the unlimited capacity case are implied by the new results for the bounded capacity case.

For future research it is interesting to work on the *online-time* model with bounded batch capacity. Even for the specific problem with $B = 2$ there is an interesting gap between the lower bound of $(\sqrt{5}+1)/2$ [19, 84] and the upper bound of $7/4$ [69].

For the online bidding problem there exists an optimal e-competitive randomized online algorithm [13]. This algorithm starts by drawing a random variable ξ from $(0, 1)$ and then submits bids $b_i = 2^{i+\xi}$. This gives immediately the optimal scheduling strategy for the unlimited capacity case. It is an interesting question whether such a randomization of the algorithm presented here for the bounded capacity case, results in an optimal randomized algorithm.

Part II

Project Scheduling

4 Time-constrained project scheduling

4.1 Introduction

In this chapter we present a new scheduling methodology for scheduling problems with strict deadlines. The new approach is applied to a project scheduling problem with strict deadlines on the jobs, which we call the Time-Constrained Project Scheduling Problem (TCPSP). In many project scheduling problems from practice, jobs are subject to strict deadlines. In order to meet these deadlines, different ways to speed up the project are given, e.g. by working in overtime or hiring additional resource capacity. These options are costly but often not avoidable. The question arises how much, when, and what kind of extra capacity should be used to meet the deadlines against minimum cost.

The TCPSP is a variant on the well studied RCPSP (Resource-Constrained Project Scheduling Problem). However, there are fundamental differences between the time-constrained and the resource-constrained variant. In the TCPSP, the deadlines are strict and resource capacity profiles can be changed, whereas in the RCPSP, the given resource availability cannot be exceeded and the objective is to minimize the makespan. Moreover, in the TCPSP a nonregular objective function is considered. Therefore, the existing solution techniques of the RCPSP are not suitable for the TCPSP. For an overview of the literature on the RCPSP see, e.g. [36, 49, 50].

Although in practice deadlines often occur in projects, Time-Constrained Project Scheduling obtained far less attention in the literature than the Resource-Constrained Project Scheduling has. In [62] the TCPSP with regular and non-regular objective functions is discussed and it lists both priority-rule methods and branch-and-bound

algorithms. However, the concept of overtime is not considered. In [15] an ILP-formulation for the TCPSP is given and the concept of project crashing is discussed. In project crashing, the processing time of a job can be reduced to meet the projects deadline at the cost of an increased resource usage, see also [44, 58]. In [45] a heuristic procedure for the TCPSP with limited hiring is discussed. Another related problem is the resource investment problem (RIP), see [16, 60]. For the RIP, it is also the goal to invest in resources as little as possible in order to meet a given project deadline. However, in the RIP only the investment in (new) resources is considered. If for a resource an investment in several units of this resource is done, these units are available for the complete planning horizon. The TCPSP is a scheduling problem where the resource availability in a time unit is fixed (e.g. amount of employees in a time unit during a working day) and it may be decided to extend in some of the time units these resource capacities by hiring extra resource units. Similarly, the time driven rough-cut capacity planning problem, see [25], is about matching demand for resources and availability of resources. For each time bucket, it can be decided how many resources are assigned to a job.

For the TCPSP, as presented here, the basic capacities of the resources are fixed. The resource investment decision took place in an earlier stage. What remains is the short term scheduling of the jobs so that costs of working in overtime and hiring are minimized. The TCPSP in this form is motivated by a cooperation with a Dutch company developing commercial planning software. It encountered this problem with several of its clients and feels a need to extend its planning tool by a component which allows to tackle TCPSP like problems. To the best of our knowledge, in this chapter the concept of overtime and multiple forms of irregular capacity are simultaneously included in the modeling for the first time.

The outline of this chapter is as follows. In Section 4.2, we introduce the presented solution approach that starts by planning jobs only fractionally. Section 4.3 gives a detailed problem description and presents an ILP formulation to model the TCPSP. To solve the TCPSP, we develop a two stage heuristic in Section 4.4. The first stage of the heuristic constructs a partial schedule. The key of the approach lies in this first stage where jobs may be scheduled for a shorter duration than required. The second stage turns the partial schedule into a feasible schedule. Section 4.5 concerns the computational results. The test instances that we use are RCPSP benchmark instances from the PSPlib, see [52], that are modified to TCPSP instances. In Section 4.6, we indicate how to extend the solution approach, e.g. for the multi-mode case. Section 4.7 gives some concluding remarks.

4.2 Scheduling with strict deadlines

For a good solution of the TCPSP, it is preferable to use regular working hours and avoid work in overtime and hiring, since these add to the costs. A typical greedy like planning heuristic would start by scheduling jobs using only regular time and

available resources as long as possible, and thereby avoiding costs. Only if a job would miss its deadline by using only regular working hours, working in overtime and hiring capacity comes into the picture. In our experience, using such a greedy strategy results in bad solutions. The reason for this is that by avoiding costs in the beginning, bottlenecks get shifted toward the end of the horizon and result in a pile up of cost toward the deadlines. The best solution might be to hire a small amount of additional capacity at the beginning to avoid costly situations when fitting in jobs that get scheduled close to their deadline.

Therefore, we propose a new scheduling methodology. Instead of scheduling jobs for their complete duration, we start by scheduling the jobs only for a fraction of their duration, and then we gradually increase the fraction for which the jobs are scheduled. This way, all jobs are partially admitted in the schedule before we address the bottlenecks. During this procedure it is guaranteed that we can turn the partial schedule into a feasible schedule. The goal is to create a schedule in which the usage of irregular capacity is low, i.e. we try to prevent a pile up of costs toward the deadlines.

In the remainder of this chapter we apply the idea of fractional planning and gradually increasing the fraction for which jobs are planned to the TCPSP. The computational test show that this idea works very well for the TCPSP. We believe that this new concept can be successfully applied to other scheduling problems with strict deadlines as well.

4.3 Problem description and ILP formulation

In this section, we describe the Time-Constrained Project Scheduling Problem and formulate it as an integer linear program (ILP). Since distinguishing between regular time and overtime is crucial for the TCPSP, we first discuss (in Section 4.3.1) how time is divided in different units, what the cost structure of working in these different time units is, and what non-preemptiveness of jobs means. After that, Sections 4.3.2 and 4.3.3 give a formal problem formulation of the TCPSP and a corresponding ILP model. Due to the complexity of the problem, we cannot expect to solve the TCPSP via this ILP-model within reasonable time. However, we present the ILP, since the heuristic presented in Section 4.4 makes use of the ILP formulation to construct feasible solutions and to perform a neighborhood search.

4.3.1 Modeling regular time and overtime

In project scheduling, a time horizon $[0, T)$ is divided into T time units, $t = 0, ..., T-1$, where time unit t represents the time interval $[t, t+1)$. The granularity of this discretization depends on the granularity of the input data. In the following, we use T to refer to both the set of time units and the horizon $[0, T)$. However, for the Time-Constrained Project Scheduling Problem, this is not enough. The time

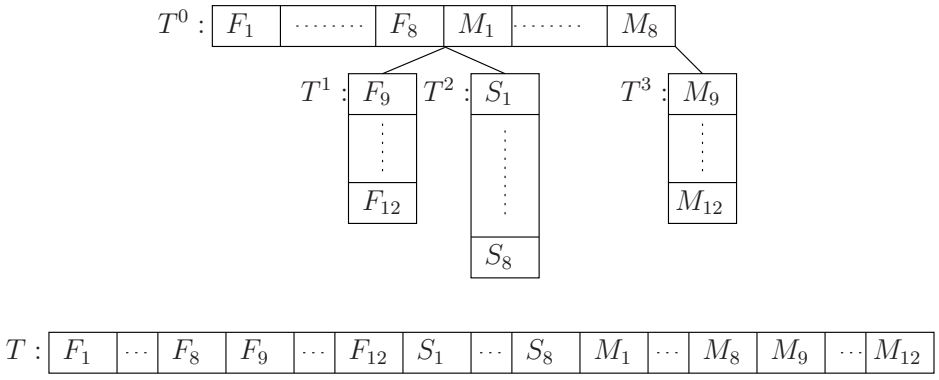


Figure 4.1: Example: Chains of time units.

units that represent the overtime have different properties than the time units that represent the regular time of a work day, e.g. different cost for hiring.

Therefore, we take a different view on the time units. We define one chain of regular time units $T_0 = \{t^0_1, ..., t^0_{N_0}\}$, and L chains of time units that are available for working in overtime $T_l = \{t^l_1, ..., t^l_{N_l}\}$, $l = 1, ..., L$. For each chain of overtime time units T^l, there is an index $\tau^l \in \{1, ..., N_0\}$ that indicates the last regular time unit ($t^0_{\tau^l} \in T_0$) before the start of the chain T^l. Furthermore, we assume that the chains do not overlap in time and that the overtime chain T^{l+1} is later in time than the overtime chain T^l. If the first time unit is not a regular time unit but an overtime time unit, we introduce an artificial regular time unit to start with. The corresponding set of time units T is the union of all chains, i.e. $T = \bigcup_{l=0}^{L} T^l$. As a consequence, each time unit $t \in T$ belongs to one unique chain T^l, $l \in \{0, ..., L\}$. Due to the above mentioned constraints, the set of time units itself is also a chain, so we can compare each pair of time units, no matter whether these time units are in regular time or in overtime. Note that it is possible that certain regular time units are followed by more than one chain of overtime time units.

Consider the following example. For processing, we have 8 regular hours available on Friday and 8 on Monday, and 4 overtime hours on Friday evening, 8 on Saturday, and again 4 on Monday evening. This means that $T^0 = \{F_1, ..., F_8, M_1, ..., M_8\}$, $T^1 = \{F_9, ..., F_{12}\}$, $T^2 = \{S_1, ..., S_8\}$ and $T^3 = \{M_9, ..., M_{12}\}$. Furthermore, $t^0_{\tau^1} = t^0_{\tau^2} = F_8$ and $t^0_{\tau^3} = M_8$. Figure 4.1 illustrates this example.

As in most project scheduling problems, preemption is not allowed during regular time. However, for the problem at hand, it is allowed to work (without gaps) in some of the time units in overtime, then stop and continue the work in the next regular time unit or in a subsequent chain of overtime, if no regular time is in between these two chains of overtime. This modified non-preemption requirement can be formally stated as follows. If a job is processed in two time units of a chain T^l, it is also

processed in all time units in between these two time units, and if a job is processed in two different overtime chains T^k and T^l, $1 \leq k < l \leq L$, then this job has to be processed in all regular time units $\{t^0_{\tau^k+1}, t^0_{\tau^k+2}, ..., t^0_{\tau^l}\} \subset T^0$.

4.3.2 TCPSP with working in overtime, and hiring in regular time and in overtime

For the TCPSP, a set of n jobs, $\{J_1, ..., J_n\}$, each job J_j with a release date r_j and a deadline d_j, has to be scheduled without preemption (according to the modified non-preemption requirement of Section 4.3.1) for p_j time units on a time horizon $[0, T)$. This time horizon is divided into one chain of regular time units and multiple chains of overtime time units (as in Section 4.3.1). The release date r_j gives the first time unit in which job J_j is allowed to be processed and its processing has to be finished before d_j, i.e. $d_j - 1$ is the last time unit where it is allowed to work on job J_j. For processing the jobs a set of K resources, $\{R_1, ..., R_K\}$, is available, where resource R_k has a capacity Q_{kt} in regular time unit t. To hire one extra unit of resource R_k in time unit $t \in T^0$, an amount c^H_{kt} has to be paid. In an overtime time unit $t \in T \setminus T^0$ the use of one unit of the available resource R_k costs c^O_{kt} and hiring one extra unit costs c^{OH}_{kt}. It is assumed that the amount of regular resource available in overtime T^l, is equal to the regular resource capacity in the last regular time unit, i.e. $Q_{k,t^0_{\tau^l}}$. There is no limitation to the amount of resource hired. The processing of the jobs is restricted by precedence relations, which are given by sets of jobs P_j , denoting all direct predecessors of job J_j , that have to complete before J_j starts. Each job J_j has a specified processing time p_j and during the processing of job J_j it requires q_{jk} units of resource R_k.

Motivated by practice we incorporate one additional requirement on working in overtime. If personnel work during one time unit in overtime, they have to be present from the beginning of that overtime chain until that time unit. Although, they might not have to work immediately. This requirement reflects the fact that normally personnel start working in overtime immediately after the regular time, and then work for a continuous period in that one overtime chain. Thus, the amount of available resource used in an overtime chain is non-increasing.

4.3.3 ILP-formulation of the TCPSP

As a generalization of the classical TCPSP, the considered problem is also NP-hard (see [62]). Nevertheless, we present an ILP-formulation since it plays an important role in the heuristic procedure presented in the next section. To model the TCPSP as an ILP, we employ one type of binary decision variables x_{jt} that are equal to 1 if job J_j is being processed in time unit t. To formulate the problem as an ILP, we use four types of variables that can be deduced from the variables x_{jt}. We use binary variables s_{jt} that are equal to 1 if time unit t is the first time unit where job J_j is being processed. The nonnegative variables H_{kt} represent the amount of capacity

hired of resource R_k in time unit $t \in T^0$, and nonnegative variables O_{kt} and HO_{kt} represent the amount of capacity made available through working in overtime and hiring in overtime for resource R_k in time unit $t \in T \setminus T^0$, respectively.

Using these variables, the TCPSP can be modeled by the following ILP:

$$\text{minimize:} \quad \sum_{k=1}^{K} \Big[\sum_{t \in T^0} c_{kt}^{H} H_{kt} + \sum_{t \in T \setminus T^0} (c_{kt}^{O} O_{kt} + c_{kt}^{HO} HO_{kt}) \Big] \tag{4.1}$$

subject to:

$$\sum_{t=r_j}^{d_j-1} x_{jt} = p_j \qquad \forall j \tag{4.2}$$

$$\sum_{j=1}^{n} q_{jk} x_{jt} \leq Q_{kt} + H_{kt} \qquad \forall k, t \in T^0 \tag{4.3}$$

$$\sum_{j=1}^{n} q_{jk} x_{jt} \leq O_{kt} + HO_{kt} \qquad \forall k, t \in T \setminus T^0 \tag{4.4}$$

$$O_{k,t_1^l} \leq Q_{k,t_{\tau^l}^0} \qquad \forall l \geq 1, k \tag{4.5}$$

$$O_{k,t_h^l} \leq O_{k,t_{h-1}^l} \qquad \forall l \geq 1, 1 < h \leq N_l, k \tag{4.6}$$

$$\sum_{t=r_j}^{d_j-p_j} s_{jt} = 1 \qquad \forall j \tag{4.7}$$

$$x_{j,t_1^0} = s_{j,t_1^0} \qquad \forall j \tag{4.8}$$

$$x_{j,t_h^0} \leq x_{j,t_{h-1}^0} + s_{j,t_h^0} + \sum_{t \in \cup_{\{l|\tau^l = h-1\}} T^l} s_{j,t} \qquad \forall j, h \geq 1 \tag{4.9}$$

$$x_{j,t_1^l} \leq x_{j,t_{\tau^l}^0} + s_{j,t_1^l} + \sum_{t|t<t_1^l, t \in \cup_{\{k|\tau^k = \tau^l\}} T^k} s_{j,t} \qquad \forall l \geq 1, j \tag{4.10}$$

$$x_{j,t_h^l} \leq x_{j,t_{h-1}^l} + s_{j,t_h^l} \qquad \forall h > 1, l \geq 1, j \tag{4.11}$$

$$p_i\big(1 - s_{j,t}\big) \geq \sum_{\bar{t} \geq t} x_{i\bar{t}} \qquad \forall t \in T, J_j, J_i \in P_j \tag{4.12}$$

$$s_{jt} = 0 \qquad \forall t \notin \{r_j, ..., d_j - p_j\} \tag{4.13}$$

$$x_{jt} = 0 \qquad \forall t \notin \{r_j, ..., d_j - 1\} \tag{4.14}$$

$$x_{jt}, s_{jt} \in \{0, 1\} \qquad \forall j, t \in T \tag{4.15}$$

$$H_{kt} \geq 0 \qquad \forall k, t \in T^0 \tag{4.16}$$

$$O_{kt}, HO_{kt} \geq 0 \qquad \forall k, t \in T \setminus T^0 \ . \tag{4.17}$$

The objective function (4.1) minimizes the total costs. Constraint (4.2) ensures that each job is processed for the required duration between the release date and

the deadline. Constraint (4.3) ensures that the amount of required resource does not exceed the amount of available resource in regular time units, using regular capacity and hiring. Constraint (4.4) ensures that the resource usage in an overtime time unit cannot exceed the amount that is available through working in overtime and hiring in overtime. Constraints (4.5) and (4.6) force the amount of work in overtime to be non-increasing in each chain of overtime, and not to exceed the capacity of the regular resource in the last regular time unit. Constraint (4.7) ensures that each job starts exactly once. The modified non-preemption requirement is specified in Constraints (4.8) to (4.11). In Constraint (4.8), we guarantee that job J_j is processed in the first time unit of T^0 if it also starts in that time unit. For the other regular time units, Constraint (4.9) ensures that job J_j can only be processed if it is processed in the previous regular time unit, or starts in this regular time unit, or starts in an overtime time unit directly succeeding the previous regular time unit. For processing in overtime time units, Constraint (4.10) states that we are allowed to work on a job J_j in the first time unit of an overtime chain if we work on it in the last regular time unit, or we start the job at this time unit, or the job starts in an overtime time unit which succeeds the last regular time unit, but preceding this overtime time unit. Constraint (4.11) states that it is only allowed to work on a job J_j in a time unit that is not the start of an overtime chain, if that is done also in the previous overtime time unit or it is starts in this overtime time unit. The precedence relations are managed in Constraint (4.12). If job J_j starts in time unit t, then the left hand side of the constraint becomes zero, implying that in none of the time units after t there can be worked on job J_i, i.e. job J_i is finished before t. On the other hand, if job J_j does not start in time unit t, Constraint (4.12) gives no restriction. Constraints (4.13) and (4.14) put all non-relevant s_{jt} and x_{jt} variables to zero. Finally, Constraints (4.15) to (4.17) define the domain of the variables.

4.4 Solution approach

In the previous section, we presented an ILP-formulation of the TCPSP. However, we cannot expect to solve large instances with the ILP-formulation. In this section, we present a heuristic approach based on the concept of planning fractionally.

4.4.1 Two stage heuristic

Outline

After initialization, in which feasibility of the instance is checked, the first stage constructs a number of different partial schedules by randomized sampling. Partial schedules are constructed by scheduling one job at a time for only a fraction of its duration. The job to be scheduled next is selected with a probability derived from its deadline and the state of the schedule constructed so far. Once all jobs are partially admitted in the schedule, the fraction for which the jobs are scheduled is gradually

increased. In this first stage, no use of overtime is made and, therefore, the resulting schedule in general does not contain the jobs for their complete duration. However, it is guaranteed that in the second stage the partial schedules can be made feasible by allowing the use of overtime. On a small subset of good feasible schedules, a neighborhood search will be performed.

Initialization

In the initialization stage, we calculate modified release dates and modified deadlines, which are sharp bounds on the start and completion times of the jobs. From these modified release dates and modified deadlines, it is possible to determine in advance whether there exists a feasible schedule.

The modified release date $\tilde{r}_j$ of a job J_j is the first time unit, such that if the job starts in this time unit, all its predecessors can be scheduled if there is abundant resource capacity, both in regular time and overtime. The modified deadline $\tilde{d}_j$ of a job J_j is the last time unit, such that if the job finishes the unit before, all its successors can still be scheduled if there is abundant resource capacity, both in regular time and overtime. A feasible schedule exists if and only if for each job J_j the interval $[\tilde{r}_j, \tilde{d}_j]$ is large enough to process the job.

The modified release dates $\tilde{r}_j$ can be calculated by a forward recursion through the precedence network:

$$\tilde{r}_j := \max\{r_j, \max_{i \in P_j}\{\tilde{r}_i + p_i\}\}, \quad \forall j \ . \tag{4.18}$$

With a backward recursion, we calculate the modified deadlines $\tilde{d}_j$:

$$\tilde{d}_j := \min\{d_j, \min_{\{i|j \in P_i\}}\{\tilde{d}_i - p_i\}\}, \quad \forall j \ . \tag{4.19}$$

All time windows, the time between the modified release date and modified deadline, should be large enough to accommodate the job:

$$\tilde{d}_j - \tilde{r}_j \geq p_j, \quad \forall j \ . \tag{4.20}$$

Note that the modified release dates and modified deadlines are calculated with respect to the time horizon T.

Stage 1

In the first stage, we generate a schedule containing all jobs in which jobs get intentionally scheduled for a shorter duration than necessary, using only regular time units. For each job J_j, we are going to determine a start time $S_j \geq \tilde{r}_j$, a completion time $C_j < \tilde{d}_j$ such that in $[S_j, C_j]$ there is enough time available to process the job (with respect to T). However, at this first stage, we assign only the regular time in $[S_j, C_j]$ to job J_j. To start, we aim to create a partial schedule such that each job is

scheduled for at least a fraction $a_0 \in (0, 1]$. However, there is no guarantee that each job reaches this fraction, but we do ensure that the assigned starting and completion times allow for a feasible solution when using overtime and hiring extra capacity are allowed. Next, we describe this selecting and scheduling in detail.

We generate the partial schedules with a randomized sampling procedure and then try to improve them by increasing the fraction for which jobs are scheduled. In a serial manner, we select jobs and include them in the schedule. Let D_{jobs} denote the decision set from which we select the job to be scheduled next. The set D_{jobs} contains all jobs for which all predecessors are already in the current schedule. Initially,

$$D_{jobs} := \{J_j | P_j = \emptyset\} \ .$$

In each iteration, we select a job from the decision set D_{jobs}. For each job $J_j \in D_{jobs}$, we determine a priority ν_j. This priority depends on how early the job can start. Let e_j denote the earliest start time of job J_j in the following sense: e_j is the earliest time unit in T, greater than or equal to the modified release date $\tilde{r}_j$ and greater than the completion times of job J_j's predecessors, such that for each of the $\lceil a_0 \cdot p_j \rceil$ regular time unit following e_j still enough resource capacity is available to schedule job J_j. This, because at this stage we only try to schedule job J_j for a fraction a_0 in only the regular time T^0.

For each job J_j, we define a slack, sl_j, with respect to T^0 (see Figure 4.2):

$$sl_j := \tilde{d}_j - \lceil a_0 \cdot p_j \rceil - e_j, \quad \forall J_j \in D_{jobs} \ .$$

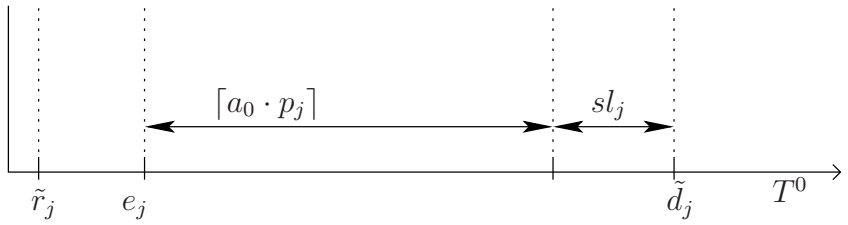


Figure 4.2: Derivation of the slack of job J_j

It is possible that the slack of a job becomes negative, implying that we cannot schedule the job for a fraction a_0 without hiring or working in overtime. As a consequence, we either have to schedule this job for a smaller fraction at this stage or hire a small amount of resources in regular time to enlarge the fraction for which this job is scheduled in regular time. We get back to this problem later.

If a job has a small slack value, there is not much room to maneuver this job and therefore we prefer to schedule this job next. We give such a job a high priority.

More precisely, the priority value of job J_j becomes:

$$\nu_j := \max_{J_i \in D_{jobs}} \{sl_i\} - sl_j, \quad \forall J_j \in D_{jobs} \ .$$

Note that $\nu_j \geq 0$. To get strictly positive selection probabilities for each job in D_{jobs}, we add 1 to the priority and normalize these priority values. The resulting selection probability η_j of job J_j is:

$$\eta_j := \frac{(\nu_j + 1)^\alpha}{\sum_{J_i \in D_{jobs}} (\nu_i + 1)^\alpha}, \quad \forall J_j \in D_{jobs} \ ,$$

where the value of α lies in $[0, \infty]$, and indicates the importance of the priority value when selecting a job. If α equals 0, jobs are selected uniformly at random from D_{jobs}. If α tends to infinity, the job with the highest priority gets selected deterministically.

Now that we have selected a job, we determine its start and completion time, $S_j \in T$ and $C_j \in T$ respectively. Since the job J_j gets scheduled in the regular time units in $[S_j, C_j]$, for S_j and C_j the following must hold to ensure that we get a large enough interval for job J_j and leave enough room for the remaining jobs to be scheduled:

$$S_j \geq \tilde{r}_j \tag{4.21}$$
$$C_j < \tilde{d}_j \tag{4.22}$$
$$S_j > C_i, \quad \forall J_i \in P_j \tag{4.23}$$
$$C_j - S_j + 1 \geq p_j, \quad \text{with respect to } T \ . \tag{4.24}$$

There can be many pairs (S_j, C_j) satisfying these four constraints. Therefore, we select a pair by applying the following criteria, in the presented order:

1. Select a pair that hires as less as possible.
2. Select a pair that minimizes $\max\{0, \lceil a_0 \cdot p_j \rceil - (C_j - S_j)\}$, with respect to T^0.
3. Select a pair that minimizes C_j.
4. Select a pair that maximizes S_j.

After the first three criteria there may be still several pairs left, whereas after criterion 4 the values for S_j and C_j are uniquely determined. Due to Constraints (4.21) to (4.24), it might be necessary to hire resources in regular time, but via criterion 1 we try to avoid this. The second criterion states that we select from the remaining start and completion times, those that schedule the job with minimal shortage to the fraction a_0. This way we try to schedule each job close to the desired fraction a_0. The third criterion gives, from the remaining start and completion times, those that minimize the completion time, not to hinder the jobs that still have to be scheduled.

Finally, we choose from the remaining pairs of start and completion time, the pair that maximizes S_j.

If all jobs are partially scheduled, it might be possible to extend the processing time of jobs within regular time, such that they are scheduled for a larger fraction. To prevent the shifting of problematic situations (as described in Section 4.2), we would like to spread this increase evenly over all jobs. Therefore, we use a procedure that repeatedly tries to extend the jobs to a higher fraction, going through the schedule alternating from back to the front and from the front to the back. Let $(a_0, a_1, \ldots, a_k)$ denote a non-decreasing sequence of fractions that we apply, where a_0 equals the fraction used in the randomized sampling.

One extension step consists of a backward and a forward extension. In the i^{th} backward extension, we go through the current schedule from the back to the front and search for each job J_j a new pair (S_j, C_j) satisfying (4.21) to (4.24), by the following four criteria in the presented order:

1. Select a pair that requires no more hiring than before.
2. Select a pair that minimizes $\max\{0, \lceil a_i \cdot p_j \rceil - (C_j - S_j)\}$, with respect to T^0.
3. Select a pair that maximizes S_j.
4. Select a pair that minimizes C_j.

The i^{th} backward extension is followed by the i^{th} forward extension, which is a mirrored version of the i^{th} backward extension.

Stage 2

The result of Stage 1 is a schedule containing all jobs, scheduled in regular time units and not necessarily for the required length. In Stage 2, we use working in overtime and hiring in regular time and overtime to get a feasible solution of the TCPSP. The main idea to get a feasible solution is the following. Iteratively, for each job J_j that is not scheduled for its required duration, we solve an ILP. This ILP is given by a restricted version of the ILP in Section 4.3.3, where all jobs except job J_j get 'frozen' as they are in the current schedule and only the timing of job J_j is left to the ILP solver. More precisely, we deduce a release date and deadline for job J_j, imposed by the current schedule and the original release date and deadline. Furthermore, we deduce the regular capacity that is still available for scheduling job J_j. For the regular time units, this is the original capacity minus the capacity used by the other 'frozen' jobs in the current schedule. For the overtime time units, this is the work in overtime that is imposed by other jobs, but not used as a consequence of Constraint (4.6). The ILP solver returns the timing of the job J_j and the corresponding use of irregular capacity. This, together with the 'frozen' jobs, gives a schedule that is the same as before except for the scheduling of job J_j. Job J_j is now scheduled for its required duration. Note that the requirements (4.21) to (4.24) on the start

and completion times S_j and C_j of job J_j, ensure that there is always a feasible scheduling of job J_j. At the end of this iterative process, we have a feasible solution for the TCPSP.

The order in which the jobs are extended to their required duration, can be chosen through numerous criteria. Possible orderings are: Smallest scheduled fraction ($\bar{a}_j$) first, largest unscheduled processing time ($(1-\bar{a}_j)p_j$) first, and an ordering deduced from the precedence network.
Now that we have a feasible schedule for the TCPSP, we apply a neighborhood search to improve upon this schedule. We use a neighborhood search based on a method proposed in [64]. This method selects a number of jobs, 'freezes' the remainder of the schedule, and calculates for the resulting ILP an optimal schedule. It is similar to the first part of Stage 2, but now the timing of a small number of jobs is left to the ILP-solver. One iteration of the neighborhood search consists of the following steps:

1. Select a subset of the jobs, $J^{\text{Neighbor}} \subset \{J_1, ..., J_n\}$.
2. 'Freeze' all jobs $J_j \notin J^{\text{Neighbor}}$.
3. Determine release dates, deadlines, available capacities for the jobs in J^{Neighbor}.
4. Solve the resulting ILP.

There are numerous ways to select a subset J^{Neighbor} of jobs. For example, Palpant et al. propose 1) to select a job together with all its predecessors, or 2) select a job and all jobs scheduled parallel with, and contiguous to it. One can think of many more selection criteria, but the main idea is not to select jobs arbitrarily, but to select jobs that occur close to each other in the schedule. Otherwise, there is not much to improve, i.e. the neighborhood is too small.

4.5 Computational results

This section describes the setup of the computational tests and the results for testing the solution approach presented in this chapter. Since this is the first attempt to tackle the TCPSP with working in overtime, and hiring in regular time and overtime, the presented heuristic cannot be compared with any existing method. However, there is much known for the RCPSP. Therefore, we take benchmark instances of the RCPSP and transform them into instances of the TCPSP. This is done in such a way that we can draw conclusions with respect to the quality of the TCPSP solutions. This section starts with describing the transformation of the instances and the parameter setting in the heuristic, before presenting the computational results.

4.5.1 Construction of TCPSP instances

For the RCPSP, a set of benchmark instances called PSPlib generated by [52] and [53] can be found on the web ([51]). These instances have been employed in many studies,

for example in [17, 48, 53]. These RCPSP instances form the base of the TCPSP instances. To transform the instances from the PSPlib into TCPSP instances, three additional aspects have to be introduced: Overtime and hiring possibilities with their associated costs, and deadlines.

The TCPSP distinguishes between regular and overtime time units, where the RCPSP has only one type of time units. We let each time unit from the RCPSP correspond to a regular time unit in the TCPSP. In addition, we introduce overtime in a weekly pattern. Day 1 of the week, the Sunday, contains only 8 overtime time units. Day 2 to 6, the weekdays, start with 8 regular time units, followed by 4 overtime time units. Day 7, the Saturday, contains again only 8 overtime time units. This weekly pattern is repeated until the largest deadline of the jobs is reached. All other aspects, like job durations, precedence relations, resource availability, and resource requirements remain unchanged.

To get insight in the quality of the solution generated by the two stage heuristic, we compare the TCPSP solution with the RCPSP solution. A comparison can be made if we set all costs equal to 1 and let the deadline of all jobs be the (best known upper bound on the) minimum makespan of the RCPSP instance. This means that the number of regular time units before the deadline in the TCPSP, is exactly the (best known upper bound on the) minimum makespan of the RCPSP instance. The quality of the schedule is given by its costs. If a schedule has zero costs, the two stage heuristic gives an optimal (best known) schedule for the RCPSP instance. If not, the costs of a schedule gives an indication on how far we are from the best known schedule, i.e. it gives the amount of irregular capacity used to reach the best known makespan.

Setting all costs equal to 1 implies that working in overtime is equally costly as hiring in regular time or overtime. This does not fit the real world situation, but it allows us to measure the total amount of irregular capacity needed. Note that due to Constraint (4.6) and all costs equal to 1, hiring in overtime is at least as good as working in overtime. Therefore, the possibility of working in overtime could be removed from the model. However, we choose not to do this, since it would reduce the computational time and thereby give a false indication on the computational time of real life instances.

Besides choosing the deadline equal to the (best known upper bound the) minimum makespan, which we denote by C_{max}, it can be chosen as a fraction of $C_{\max}$. By letting the deadline be equal to $\lceil b \cdot C_{\max} \rceil$, where $b \in (0, 1]$, the problem becomes tighter and more irregular capacity will be needed. The resulting objective value will indicate the costs to complete the project earlier. Since the problem becomes tighter with $b < 1$, we get a better insight in the influence of the different parameter settings of the two stage heuristic.

# iterations	2	2	4	4	4	4	5	5	5	5	10	10	10	30
α	100	50	20	10	9	8	7	6	5	4	3	2	1	0

Table 4.1: Different values of α on the 100 schedules generated.

4.5.2 Parameter setting

In each stage of the heuristic, there are a number of parameters that need to be set. In the first stage, these are the fraction a_0, the α value for the randomized sampling, and the extend sequence in the improvement. In the second stage, there is the order in which the jobs are extended to their required duration, and the choice of the neighborhood for the improvement. This subsection concerns the setting of these parameters.

Since there are far too many different parameter settings to test all possible combinations, we determine the parameter setting one parameter after the other and evaluate the achieved result after the feasibility step (and not after the neighborhood search). For each instance, we generate a number of partial schedules (by random sampling), extend them, and turn them into a feasible schedule. Out of these schedules, we select the one with minimum costs to be the solution.

For testing the parameter settings, we use a selection of 10 instances with 30 jobs and 10 instances with 120 jobs from the PSPlib. Initially, we use a deadline of 90% of the best known upper bound on the makespan, i.e. $b = 0.9$. We are then quite sure that the objective value will not equal zero, allowing us a better measurement of the effect of the parameters. For these initial tests, we use *largest unscheduled processing time first* as priority rule in the feasibility step.

To determine good values for α, we fix all other parameters. Testing different values for α, it turns out that if a small number of schedules for each instance are generated, large values for α outperform small values. However, as the number of schedules generated per instance increases, the random search ($\alpha = 0$) outperforms the higher values for α. Therefore, we conclude that it is best to start with a high value for α and decrease it as more schedules for the same instance are being generated. From these tests, we conclude that taking 100 randomized samples gives enough diversity. Table 4.1 presents the decreasing values of α we use for the 100 random samples. It states that the first two random samples are taken with an α value of 100, the next two with value of 50, and so on. These values are used in the remainder of the computational experiments.

For the randomized sampling (RS) to generate partial schedules, we need an initial fraction a_0. If we choose a_0 too close to 1, we do not benefit from the idea that scheduling only a fraction prevents the shifting of problems toward the deadlines. If we choose a_0 too small, we observed that the jobs are pulled to the front of the scheduling period, causing problematic situations in the beginning of the horizon. Tables 4.2 and 4.3 report on tests using different values of a_0. Within the tests,

RS a_0	Extension seq. $a_1, a_2, \ldots$	Average percentage planned after RS	Average percentage planned after extend	Average cost after feasibility
0.5	$0.6, 0.7, 0.8, 0.9, 1$	78.0	88.9	97.4
0.6	$0.7, 0.8, 0.9, 1$	80.2	89.1	90.5
0.7	$0.8, 0.9, 1$	83.7	90.3	91.1
0.8	$0.9, 1$	87.3	91.0	76.2
0.9	1	88.0	90.7	87.0
1	–	88.0	88.0	154.4

Table 4.2: Different values of a_0 on instances with 30 jobs, and $b = 0.9$.

RS a_0	Extension seq. $a_1, a_2, \ldots$	Average percentage planned after RS	Average percentage planned after extend	Average cost after feasibility
0.5	$0.6, 0.7, 0.8, 0.9, 1$	82.9	92.6	467.3
0.6	$0.7, 0.8, 0.9, 1$	84.7	92.7	439.4
0.7	$0.8, 0.9, 1$	86.9	93.7	417.2
0.8	$0.9, 1$	89.4	93.8	357.9
0.9	1	92.8	92.8	445.7
1	–	90.1	90.1	580.4

Table 4.3: Different values of a_0 on instances with 120 jobs, and $b = 0.9$.

an extension with step size 0.1 is chosen. The tables give the results after the randomized sampling, after the extension, and after the feasibility procedure. Before the feasibility procedure, the costs of a schedule have no meaning and we therefore display the average percentage planned. If we only consider the average fraction planned after the randomized samples, $a_0 = 0.9$ is best. However, with the extension and feasibility, $a_0 = 0.8$ is best. Note that the average cost for instances with 120 jobs is about 4 times as large as the average cost for instances with 30 jobs. This is due to the fact that the instances with 120 jobs approximately have a 4 times higher total resource requirement (*work content*), while the deadlines are about the same.

As already can be seen from Tables 4.2 and 4.3, the extension step is important. In a next series of tests, we have applied four sequences: No extension, directly from $a_0 = 0.8$ to $a_1 = 1$, using sequence $(0.9, 1)$, and using sequence $(0.8, 0.9, 1)$. Comparing the resulting schedules when using no extension at all with an extension sequence equal to $[0.9, 1]$, we observe a major improvement, see Table 4.4. With extension, the result is a factor 1.5 better. The result of using different extend sequences is less diverse. We choose to use the extension sequence $[0.9, 1]$.

For Stage 2, we have to decide in which order we address the jobs to schedule them for the required duration. We compare: *Largest unscheduled processing time first*, *smallest scheduled fraction first*, *smallest start time first*, and *largest start time first*. It turns out that between these different orderings there is no significant difference in the costs after the feasibility step. However, *largest unscheduled processing time*

Extension seq. $a_1, a_2, \ldots$	Average percentage planned after RS	Average percentage planned after extend	Average cost after feasibility
$0.8, 0.9, 1$	89.4	93.3	386.9
$0.9, 1$	89.4	93.8	357.9
1	89.4	93.6	370.3
–	89.4	89.4	539.8

Table 4.4: Different extend sequences, 120 jobs, $b = 0.9$ and $a_0 = 0.8$.

first and *smallest scheduled fraction first*, require far less computational time, and the second performs slightly better. Therefore, we choose to use *smallest scheduled fraction first*.

The neighborhood search is the most time consuming part of the heuristic, since it has to solve many ILP's. Therefore, we select out of the 100 constructed schedules of stage 1 the one with lowest costs, and do only the neighborhood search on that schedule. Moreover, for the neighborhood search, it is important to keep the running time low, but still search a large part of the neighborhood. In each step, a number of jobs are removed from the schedule, the other jobs are fixed before we reinsert the removed jobs optimally into the schedule. To do this, we choose a point in time and remove each job that is contiguous to it. Due to the concept of overtime we define a job contiguous to time t if its start time is at most the first regular time unit after t and its completion time is at least the last regular time unit before t. By choosing a single point in time, we keep the number of removed jobs small, and all these jobs are close together. The set of considered points in time is chosen as the set of completion times of the jobs in the schedule. We process these points in an increasing order. Always when an improvement occurs, we replace the current schedule by the new schedule. We do not recalculate the set of time points to be considered. One option is to go through the schedule only once (a single pass); another option is to go through the schedule several times (a multi pass). If we do a multi pass, we use in each pass the new completion times, and stop if the last pass does not improve the schedule. From our tests, we have seen that it can take a long time to determine the optimal placements of the removed jobs, i.e. solve the ILP. We can restrict the computational time spent on one reinsertion by using time limits. If we reach such a time limit, we use the best found reinsertion of the removed jobs (this can be the placement we had before, so we are guaranteed to have a solution that is at least as good as before). Table 4.5 displays the quality against time trade-off. ($CT(s)$ stands for computational time in seconds.) Multi pass gives solutions of higher quality at a price of larger computational times; single pass with time limit (we used a 10 second time limit) gives a solution very fast, but one with higher costs. From the second and third column in Table 4.5, we see that there are a few instances with very large computational time. We have observed that there are only a few instances in which the time limit of 10 seconds is reached. Therefore, it pays to limit the computational

Strategy	Average CT (s)	Max CT (s)	Objective
Single pass/ No time limit	3,700.8	32,919.4	291.0
Single pass/ 10s time limit	84.7	192.6	289.1
Multi pass/ No time limit	3,900.9	33,385.3	264.1
Multi pass/ 10s time limit	315.8	750.7	265.6

Table 4.5: Quality time trade-off, 120 jobs, $b = 0.9$.

time spent on one reinsertion. Comparing the values in Tables 4.4 and 4.5, we see that the neighborhood search reduces the objective value by 20% to 25%.

From this subsection, we may conclude that each step in the presented method has its contribution to find a good schedule, and that the presented method is very flexible. Depending on the purpose of use, not only the parameters can be chosen appropriately, but there is also a choice in the quality against time trade-off.

4.5.3 Computational results

In the previous subsection, we have used only a small number of instances to determine the choices for the parameters of the two stage approach. In this subsection, we present a summary of the computational results for a large set of instances. We have used all single mode instances from the PSPlib, and have set the algorithm as discussed in the previous subsection with a multi pass, 10 second time limited, neighborhood search. For each instance, we construct a schedule with a deadline on 100% and 90% of $C_{\max}$. Table 4.6 summarizes the test results. The computational experiments were performed on a computer with a Intel Centrino processor running at 2.0 GHz. We used Delphi 7 to code the algorithm and CPLEX 9.1 to solve the ILP's. The details of the computational tests can be found on a website ([33]).

If we set the upper bound equal to $C_{\max}$ and solve an instance with zero cost, this means that we have an optimal (or best known) schedule for the RCPSP. We see that our method solves about 60% of the instances with zero cost for the 30, 60, and 90 job instances, but only 13% of the 120 job instances. The 30, 60, and 90 job instances are generated with similar characteristics, whereas the 120 job instances have a relative lower resource availability and are therefore tighter and more difficult. The average objective value, the amount of used irregular capacity, is only a small fraction of the total work content. In the solutions of the instances with 30 jobs, on average, 0.14% of the required capacity is satisfied with irregular capacity. This percentage goes up to 0.33% for the instances with 90 jobs. For the tighter 120 job instances, the percentage of work done with irregular capacity is 0.80%. These percentages are very low, and therefore we can conclude that the achieved schedules have a high quality.

Setting the deadline to 90% of C_{max}, we can no longer verify whether or not optimal schedules are found. As expected, the objective values increase. However,

			b=1.0				b=0.9		
Jobs	Instances	Work Content	Maximum $CT(s)$	Average $CT(s)$	Objective value	# with cost=0	Maximum $CT(s)$	Average $CT(s)$	Objective value
30	480	2,310.7	26.1	2.9	3.2	294	55.1	4.5	58.6
60	480	4,562.9	241.7	17.2	13.4	276	303.4	24.7	124.4
90	480	6,812.2	932.7	53.0	22.6	282	845.3	73.9	180.8
120	600	9,073.5	2,163.4	308.3	72.4	75	2,798.2	362.2	326.6

Table 4.6: Summary of the results.

the 10% reduction of the time horizon results in schedules that have only 2.5% of the total work content in irregular capacity for the 30 job instances, and up to 3.5% for the 120 job instances. So completing a project 10% earlier does not have to be too costly.

The computational time grows as the number of jobs grows, but the relative resource availability seems more important. The instances that have a low resource availability require more computational time than instances with high availability. The instances that require a lot of computational time are exactly those for which the corresponding RCPSP instances are also difficult and where the best found makespan is often not proven to be optimal.

4.6 Extensions of the TCPSP

In practice a more elaborate modeling of the project might be required, such as including multi-mode scheduling of jobs and time-lags on precedence relations. In this section we indicate how to extend the presented heuristic in these cases.

4.6.1 Multi-mode TCPSP

One possible extension of the TCPSP is by allowing multiple modes for scheduling the jobs. Then, for each job J_j a set M_j of different execution modes is given. Each mode $m \in M_j$ has a specified processing time p_{jm} and during the processing of job J_j in mode m it requires q_{jmk} units of resource R_k.

The presented solution approach can easily be extended to the multi-mode TCPSP. In the randomized sampling one not only has to select a job, but also a corresponding mode. This mode can be fixed until a feasible schedule is reached. A detailed description of the solution approach for the multi-mode variant is given in [32].

Once the multi-mode is incorporated, the presented model cannot only deal with working in overtime, and hiring in regular time and overtime, but also with the possibility to outsource. Outsourcing can be included by introducing a mode for each job which represents the outsourcing. We introduce for such an outsource mode a processing time and a resource requirement for an artificial resource that has to be hired. The processing time of this mode and the cost for hiring correspond to the outsourcing.

4.6.2 Including time-lags in the model

Within the new concept of time chains of Section 4.3.1, using time-lags on precedence relations can lead to problems. They are not properly defined, since it is not clear whether the time-lags only refer to regular time units or also to overtime time units. Consider the following example. If the time-lag is a consequence of a lab test that has to be done between two processes, the opening hours of the lab determine to which time units the time-lag applies. To overcome this problem, it is possible to introduce a dummy job and a dummy resource for each time-lag. With the proper resource requirements and resource availability for each dummy job and dummy resource, time-lags can have any desired property with respect to regular and overtime time units. Therefore, our approach can also deal with time-lags.

4.7 Concluding remarks

In this chapter, a new scheduling methodology is presented for scheduling jobs with strict deadlines. First jobs are scheduled for only a fraction for their required duration and then the fraction for which the jobs are scheduled is gradually increased. This idea is applied to find solutions for the TCPSP with hiring and working in overtime.

Since there are no benchmark instances for the TCPSP, we used benchmark instances from the RCPSP to get insight in the quality of the achieved schedules. It turned out that a large amount of the instances are solved to optimality. Decreasing the deadline of a project by 10% results in schedules that have far less than 10% of the work done with irregular capacity. Thus, we can state that the schedules generated by the two stage heuristic are of high quality. The computational tests also show that there is a lot of flexibility in the developed method. The flexibility is not only due to the parameter setting, but also due to the possibility to choose where to spend the computational effort. Therefore, we believe this method is very suited for practical use.

The computational tests demonstrate the effectiveness of the fractional scheduling method. We believe this methodology may also work well for other scheduling problems with strict deadlines. Moreover, the method may work well for any scheduling problems with a non-regular objective function, like earliness/ tardiness problems.

5 Time-constrained project scheduling with adjacent resources

5.1 Introduction

In this chapter we develop a decomposition method for project scheduling problems with adjacent resources. Adjacent resources are resources for which the units assigned to a job are required to be in some sense adjacent. Possible examples of adjacent resources are dry docks, shop floor spaces, and assembly areas. We focus on the Time-Constrained Project Scheduling Problem (TCPSP) with one 1-dimensional adjacent resource. However, the presented concepts and methods can be easily extended to more general models, e.g. multiple 1-dimensional adjacent resources or 2-dimensional adjacent resources.

The Time-Constrained Project Scheduling Problem (TCPSP) with an adjacent resource is defined as follows. We are given a set of jobs, a set of renewable resources and one 1-dimensional adjacent resource. Each job is characterized by a release date, processing time, deadline and its resource requirements, and has to be scheduled without preemption. The processing of the jobs is further restricted by precedence relations. The adjacent resource is a special type of resource that is characterized by two properties. First, the resource units of the adjacent resource are somehow topologically ordered (in this case ordered on a line) and the resource units assigned to a job have to be neighbored/adjacent and reassignment is not allowed. Second, motivated by the occurrence of adjacent resources in real life problems, we consider the more general case that the adjacent resource is not required only by a single job

but by groups of jobs (called job groups or simply groups). As soon as a job of such a job group starts, the assigned adjacent resource units are occupied, and they are not released before all jobs of that group are completed. In the considered model, it is only possible to hire additional capacity for the renewable resources, and not for the adjacent resource. The objective is to find a feasible assignment of the job groups to the adjacent resources and a feasible job schedule that minimizes the cost of hiring additional capacity.

The consideration of adjacent resources in the above mentioned form is motivated by a cooperation with a Dutch consultancy company. They encountered at several of their clients adjacent resource requirements. Since the project scheduling models in the literature do not cover these requirements, the company either assigns the adjacent resources in advance based on simple rules or they relax the adjacency requirements and repair the achieved solutions afterwards. However, since both approaches do not lead to satisfactory solutions, the company strives to incorporate adjacent resources in their planning software for project scheduling. One practical application is from the ship building industry that we use to illustrates the adjacency requirements. In this problem the docks form 1-dimensional adjacent resources, and all jobs related to building a single ship form a job group. Clearly, the part of the dock assigned to one ship has to satisfy the adjacency requirement. As soon as the construction of a ship starts, the assigned part of the dock is occupied until the construction is finished and the ship is removed from the dock. Removal or repositioning of a partially assembled ship is in practice too cumbersome and time consuming and therefore not an option. The other resources required to build the ships (like machines, equipment and personnel) can be modeled as renewable resources. The capacity of the dock is fixed but the capacity of renewable resources can be increased, e.g. by hiring additional personnel.

Adjacent resources have some relation with other special resource types considered in the literature. Spatial resources, as introduced in [14], are also resources which are not only required by a single job but by a group of jobs. However, no adjacency of the assigned resource units is required. Make-to-order assembly problems under assembly area constraints, see e.g. [37, 47], form a special case of project scheduling problems with spatial resources where each job group requires exactly one unit of the spatial resource. In this case the adjacency requirement is automatically fulfilled. Without the adjacency requirement on the resources, the spatial resource can also be modeled with the concept of cumulative resources, see e.g. [3, 61, 62]. Cumulative resources are, for example, used to incorporate storage facilities into project scheduling problems. When a job group starts the cumulative resource is depleted by a given amount, and replenished as soon as a job group completes. In Section 5.3.1 we show why an adjacent resource cannot be modeled as a cumulative resource.

The literature that does consider an adjacency requirement on resources, only considers the special case in which exclusively adjacent resources are considered and groups consist of a single job. In this case the scheduling problem can be seen as

a 2-dimensional packing problem. Examples of this can be found in literature on berth allocation at container terminals [31, 59], reconfigurable embedded platforms [23, 77], and check-in desks at airports [20]. In [35] such packing problems are modeled by introducing a mode for each possible placement of a job on the adjacent resource. Consequently, one has to solve a multi-mode project scheduling problem with possibly an exponential number of modes (see Section 5.3.2).

Relaxing the group and adjacency requirements, the considered problem reduces to the TCPSP as considered in Chapter 4 and [34]. The study of such types of time-constrained project scheduling problems started with [60] and [15]. In recent years, many results on project scheduling have been published, however, most of them on the resource oriented variant, the Resource-Constrained Project Scheduling Problem. For an overview see e.g. [18, 36, 50, 79].

Summarizing, the concepts of job groups and adjacency requirement on resources have been treated in the literature, but never in a combined manner. To the best of our knowledge this work is the first to consider this combination.

The outline of this chapter is as follows. In Section 5.2 we formally state the time-constrained project scheduling problem with one 1-dimensional adjacent resource. Additionally, we provide an illustrative example which we use throughout this chapter. Before we present the developed decomposition method, we discuss in Section 5.3 why existing modeling and solution techniques for related problems are not applicable when we are dealing with an adjacent resource. Section 5.4 describes the decomposition method. In this approach, first the groups are assigned to the adjacent resource and then the jobs are scheduled. The solution of the group assignment problem implies additional precedence relations between the jobs. Once these precedence relations are added, the scheduling of the jobs can be done with a method for the TCPSP, e.g. the method of Chapter 4. In Section 5.4.3 we introduce objective functions for the group assignment problem in order to steer the assignment to a promising one. Section 5.5 reports on computational tests. In Section 5.6 we give some concluding remarks.

5.2 The TCPSP with adjacent resources

In this section, we start by giving a detailed description of the Time-Constrained Project Scheduling Problem with Adjacent Resources (TCPSP with adjacent resources). As mentioned before, we restrict ourselves to a single 1-dimensional adjacent resource and, thus, each job group has a requirement for exactly one adjacent resource. However, the presented concepts and methods can be easily extended to more general models, e.g. multiple 1-dimensional adjacent resources or higher dimensional adjacent resources. In Section 5.2.2, we show how an instance of the scheduling problem can be represented by an Activity-on-Node network (AoN network) and at the end of this section, we give an example project with its corresponding AoN network and a corresponding solution to illustrate the problem. We use this illustrative

example throughout this chapter.

5.2.1 Formal model description

For a project, we are given a set $\mathcal{J}$ of n jobs, i.e. $\mathcal{J} = \{J_1, \ldots, J_n\}$. Each job J_j has a release date r_j, a processing time p_j, and a deadline d_j. W.o.l.g. we assume that all these and following input parameters are integer. Preemption of jobs is not allowed. The time horizon is divided into T time buckets, $t = 0, \ldots, T-1$, where time bucket t represents the time interval $[t, t+1)$ and $T = \max\{d_1, \ldots, d_n\}$. Thus, for each job J_j, time bucket r_j is the first and time bucket $d_j - 1$ the last in which the processing of job J_j can take place. We assume the time windows for each job to be large enough to process the job, i.e. $d_j - r_j \geq p_j$, since otherwise no feasible schedule exists. The processing of the jobs is further restricted by precedence relations, which are given by sets $P_j \subset \mathcal{J}$, denoting all direct predecessors of job J_j. With each precedence relation $J_i \in P_j$ there is an associated non-negative time lag τ_{ij} indicating that there have to be at least τ_{ij} time buckets between the completion of job J_i and the start of job J_j. We assume w.l.o.g. that all release dates and deadlines are consistent with the precedence relations, i.e. $r_i + p_i + \tau_{ij} \leq r_j$ and $d_j - p_j - \tau_{ij} \geq d_i$ for all $J_i \in P_j$. (If this is not the case, it can be achieved by a simple preprocessing.)

For the processing of the jobs there is a set $\mathcal{R}$ of renewable resources, $\mathcal{R} = \{R_1, \ldots, R_K\}$, and one 1-dimensional adjacent resource $\bar{R}$ available. Each renewable resource $R_k \in \mathcal{R}$ has a capacity $Q_{k,t}$ in time bucket t, and the adjacent resource has capacity $\bar{Q}$ in all time buckets. Job J_j has a resource requirement q_{jk} for renewable resource R_k during its processing. Additionally, we are given a set $\mathcal{G}$ of job groups, i.e. $\mathcal{G} = \{G_1, \ldots, G_m\}$. A job group $G_g \in \mathcal{G}$ represents a subset of the jobs ($G_g \subset \mathcal{J}$) and has a requirement of $\bar{q}_g$ adjacent resource units. The assigned resource units to group G_g are occupied from the first moment a job in G_g starts, and are released as soon as all jobs in G_g are completed. In principle a job can belong to any number of job groups.

In the considered model we assume the adjacent resource to have a fixed capacity. However, we do allow an increase of the capacity of the renewable resources. Increasing the capacity of renewable resource R_k in time bucket t by one unit, incurs a cost of c_{kt}. The objective is to find a feasible assignment of groups to the adjacent resource units, and at the same time a feasible schedule of jobs on the renewable resources, such that the total costs of increasing the capacity of the renewable resources is minimized.

Note that in contrast to Chapter 4, we only consider hiring of additional renewable resource capacity but no work in overtime to be able to meet the deadlines.

5.2.2 Activity-on-Node representation

Representing an instance of a project scheduling problem by an Activity-on-Node (AoN) network is a well known modeling technique from the literature, see e.g. [62].

Job	Cleaning crew (R_1)	Painting crew (R_2)	Safety inspector (R_3)	Processing time	Release date	Deadline	Direct predecessors	Job group
J_1	1	0	0	2	0	7	$\emptyset$	G_1
J_2	0	1	0	4	2	11	$\{J_1\}$	G_1
J_3	1	0	0	2	1	7	$\emptyset$	G_2
J_4	0	1	0	4	3	11	$\{J_1, J_3\}$	G_2
J_5	1	0	0	2	4	8	$\emptyset$	G_3
J_6	0	2	0	3	6	11	$\{J_5\}$	G_3
J_7	0	0	1	2	4	11	$\emptyset$	G_3

Table 5.1: Example project: Job characteristics.

In such a network nodes correspond to jobs (or sometimes called activities, hence the name) and arcs to precedence relations. Two dummy jobs having zero processing time and no resource requirements, J_0 and J_{n+1}, are added to represent the start and completion of the project, respectively. So, $\{J_0, J_1, \ldots, J_{n+1}\}$ is both the node and the job set. Each precedence relation $J_i \in P_j$ is represented by an arc from node J_i to node J_j with weight τ_{ij}. The release date of job J_j is modeled by an arc from node J_0 to J_j with weight equal to r_j.

The job groups and the deadlines can be incorporated into the AoN network construction for project scheduling problems as follows. To be able to identify the start and completion of the job groups, we add dummy jobs with zero processing time and no resource requirement. For each group G_g we introduce a start job J_g^s and a completion job J_g^c, respectively. Job J_g^s is a predecessor of all jobs in G_g with zero time lag, and has release date $\min_{J_j \in G_g}\{r_j\}$ and deadline $\min_{J_j \in G_g}\{d_j - p_j\}$. All jobs in G_g are predecessors of J_g^c. J_g^c has release date $\max_{J_j \in G_g}\{r_j + p_j\}$ and deadline $\max_{J_j \in G_g}\{d_j\}$. To model the deadlines of the jobs, we fix the scheduling of job J_{n+1} at time T (the largest deadline). The deadline of job J_j now can be modeled by an arc from node J_j to J_{n+1} with weight equal to $T - d_j$.

5.2.3 Example project

To emphasize the impact of an adjacent resource on the scheduling problem, consider the following illustrative example project from the yacht building industry. There are 3 yachts, each to be cleaned and painted in the next few days. The third yacht is also due for safety inspection. There is one adjacent resource, the dock ($\bar{R}$) with a length of 30 meters ($\bar{Q} = 30$). There are three renewable resources: cleaning crews (R_1), painting crews (R_2), and safety inspectors (R_3). There is one crew for cleaning ($Q_{1,t} = 1$), one crew for painting ($Q_{2,t} = 1$), and no safety inspector ($Q_{3,t} = 0$) available during the entire project horizon. These capacities can (and have to) be extended in some time periods by hiring personnel, e.g. the safety inspector. Table 5.1 displays the characteristics of the 7 jobs. All time lags are of zero length. Job group G_g corresponds to yacht g, and all yachts have to be placed on the one available dock. The lengths of the yachts are given by $\bar{q}_1 = 10$, $\bar{q}_2 = 10$ and $\bar{q}_3 = 20$.

Figure 5.1 shows the AoN network corresponding to this example project. All precedence relations implied by transitivity are omitted from the AoN network. Figure 5.2 gives a feasible solution for this project. The schedules for the renewable resources R_1 to R_3 indicate that for time buckets 4, 5, 8, 9, and 10 an additional painting crew and for time buckets 7 and 8 a safety inspector has to be hired. For the adjacent resource there are two different schedules displayed in Figure 5.2. Both schedules for $\bar{R}$ satisfy the timing constraints (both schedules for $\bar{R}$ have the exact same start and completion times) and both are consistent with the schedules for the renewable resources R_1 to R_3, but only the second satisfies the adjacency requirement. For the renewable resources it suffices to specify the start of each job, but for the adjacent resources the specific assignment of resource units is also necessary. The necessity of specifying the assignment of the groups to the adjacent resources is also the topic of the next section.

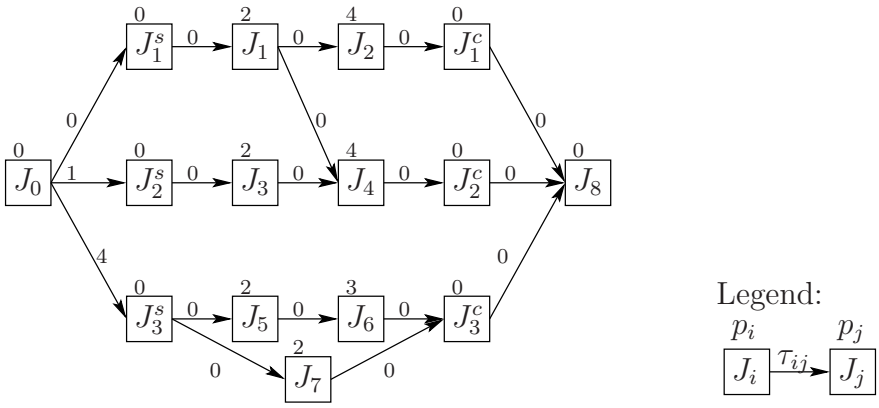


Figure 5.1: Example project: Activity-on-Node representation.

5.3 Failing modeling techniques

In this section we review two modeling techniques, *cumulative resource* modeling and *multi-mode* modeling, and comment on the use of *sequential planning heuristics* for scheduling problems having job groups. These techniques and methods seem at first glance useful for solving problems with adjacent resources. However, as we show, the additional computational complexity introduced by the adjacency requirement causes these techniques and methods to fail. The TCPSP with adjacent resources contains several elements which make the problem NP-hard. If we remove the adjacent resources and the notion of groups, we get the TCPSP which is NP-hard even if all processing times are equal to 1. However, by assuming all time windows to be

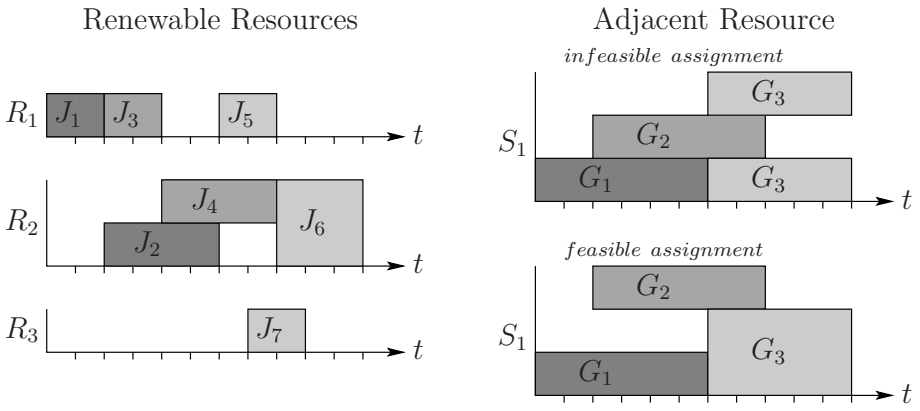


Figure 5.2: Example project: Gantt-charts of a solution.

large enough, i.e. $r_j + p_j \leq d_j$ for all J_j, and assuming unlimited hiring possibilities, at least there always exists a feasible solution for the TCPSP. In contrast to this, with the addition of only one adjacent resource the problem of deciding whether or not a feasible solution exists, turns out to be NP-complete. Furthermore, it is NP-complete to decide whether or not given group start and completion times that respect the adjacent resource capacity constraints can be extended to a feasible solution respecting also the adjacency requirements without changing the start and completion times of the job groups (see Section 5.3.1). In Section 5.3.2 we discuss why multi-mode modeling should not be used and in Section 5.3.3 we discuss what the pitfalls are when jobs are planned sequentially in the presence of job groups.

5.3.1 Cumulative resources modeling

Relaxing the adjacency requirement gives a problem that can be modeled with a cumulative resource replacing the adjacent resource. At the start of a group the cumulative resource is depleted and at completion the resource units are again available. Cumulative resources are used to model, for example, inventory levels [62]. This is, however, a proper relaxation of the problem and not a different formulation, i.e. there is no guarantee that a solution for the 'problem with a cumulative resource' can be transformed into a solution for the 'problem with an adjacent resource'. (The example on page 18 illustrates this result.)

Modeling groups with cumulative resources, relaxing the adjacency requirements and solving it as such, gives us start and completion times of the jobs and groups, but no assignment of particular adjacent resource units to groups. It is only guaranteed that in each time bucket at most $\bar{Q}$ units of the adjacent resource are used. Determining whether there exist a feasible assignment of adjacent resources given these

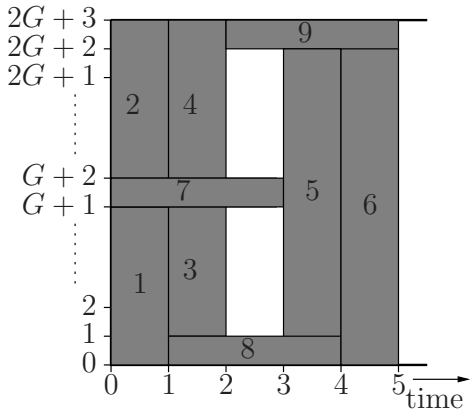


Figure 5.3: Construction in NP-hardness proof.

Group g	1	2	3	4	5	6	7	8	9
s_g	0	0	1	1	3	4	0	1	2
c_g	1	1	2	2	4	5	3	4	5
$\bar{q}_g$	G+1	G+1	G	G+1	2G+1	2G+2	1	1	1

Table 5.2: Construction in NP-hardness proof.

start and completion times, is a strongly NP-complete problem, see [20]. To give some intuition behind this result, we sketch the construction of the NP-completeness proof as given in [20].

Consider 9 groups with start times, completion times and adjacent resource requirements as in Table 5.2, and one 1-dimensional adjacent resource with capacity $2G + 3$, for some value G. Figure 5.3 displays the unique (some symmetry excluded) feasible solution to this assignment problem. The solution has two gaps of size G in time interval $[2, 3]$. By introducing new groups with resource requirements equal to the values of a PARTITION problem (see Section 1.3), assigning start time 2 and completion time 3 to these groups, and defining G to be half of the sum of these values, we have reduced the PARTITION problem to the group scheduling problem where the start and completion times are already given.

As a consequence, there is no guarantee that the start and completion times found with a solution method based on cumulative resource modeling, are such that there exists a feasible assignment of the groups to the adjacent resource units. It is even an NP-complete problem to determine whether there exists such a feasible assignment.

5.3.2 Multi-mode representation

An other possible approach to model the adjacency requirements is to represent each possible placement of a group on the adjacent resource by a different mode, as done in [35]. To explain this construction, we assume that each job group consists of exactly one job. We introduce for each possible placement a mode for the job (job group), where a placement is an interval of adjacent resource units of the required length. To be precise, we introduce a set of renewable resources Z_l (with $l = 1, \ldots, \bar{Q}$) all with a capacity of 1. Each resource Z_l represents one resource unit of the adjacent resource. For job j with adjacent resource requirement $\bar{q}_j$ we introduce modes m_{ij} (with $i = 1, \ldots, \bar{Q} - \bar{q}_j + 1$). Mode m_{ij} represents job j being placed on adjacent resource units i to $i + \bar{q}_j - 1$. Thus, the resource requirement of job j in mode m_{ij} for resource Z_l is 1 if $i \leq l \leq i + \bar{q}_j - 1$ and 0 otherwise.

In Figure 5.4 we illustrate the multi-mode construction for one 1-dimensional adjacent resource with capacity 3 and 2 jobs requiring 2 units of the adjacent resource, scheduled in different modes. So, with the introduction of these modes and additional renewable resources we have eliminated the 1-dimensional adjacent resource from the formulation.

The problem with this multi-mode representation is that the number of new resources and modes we have to introduce depends on the input data of a specific problem instance, and not on the input size of the problem. More precisely, with this transformation the instance size grows exponential, since the adjacent resource capacity is encoded in size $\log \bar{Q}$ in the original formulation, and after the transformation we need to specify $O(\bar{Q})$ modes for each job. Thus, this problem transformation is not a polynomial time transformation. From a computational time perspective, but also from a practical point of view, this should be avoided.

A second problem occurs when a constructive planning heuristic is used in combination with such a multi-mode representation. Then, not finding a feasible solution does not mean that the instance is infeasible (see Section 5.3.3). The decomposition method presented in Section 5.4 does not have this drawback and gives proof of infeasibility if it occurs.

5.3.3 Sequential planning heuristic

A sequential planning heuristic includes the jobs one by one into a schedule. This approach is the basis of almost all constructive heuristics for project scheduling problems. However, there is a large pitfall when we consider instances of the TCPSP with job groups, whether it be with an adjacent resource or a cumulative resource. As soon as the first job of a group is selected to be scheduled next, the group has to be assigned to the adjacent resource (or a given amount of the cumulative resource is depleted). These resource units stay assigned (or depleted) until the last job of the group is completed. However, in general it is hard to estimate when this will be. Thus, it is unclear how long other jobs may be delayed.

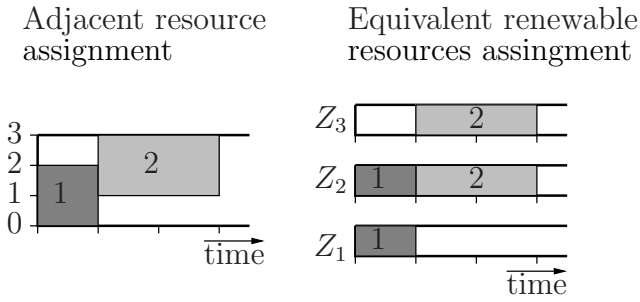


Figure 5.4: Adjacent resource by multi-mode modeling.

As a consequence, there is no mechanism to predict whether or not starting a certain group will cause jobs of other groups to miss their deadline. Thus, for a partially created schedule one cannot ensure that it can be extended to a complete feasible schedule.

For the decomposition method described next, this is not an issue. The assignment of the groups to adjacent resources is done such that a feasible job schedule exists.

5.4 The decomposition method

The considerations in the previous section indicate that for the TCPSP with adjacent resources no direct and simple heuristic can be found, since the problem of finding a feasible assignment of the adjacent resource for a given timing is already NP-complete. Furthermore, this fact also implies that it does not make sense to first treat the timing of the jobs and then the assignment of adjacent resources. Even more, it indicates that the problem of getting a feasible assignment of the adjacent recourse units to groups should play a central role and be treated first. Therefore, in this section, we present a decomposition method which considers first the feasibility of the assignment to the adjacent resource, and second the timing of the jobs. By considering the assignment of the groups first, we can use a sequential planning heuristic to schedule the jobs in the second stage and do not run into the problems mentioned in the previous section.

Since already the feasibility question of the assignment of the adjacent resource is NP-complete, we choose to use an exact approach in this first stage. It is based on an ILP formulation.

The outline of the decomposition method for the TCPSP with adjacent resources (which we refer to as the *original problem*) is as follows. The decomposition method separates the adjacent resource assignment from the problem of scheduling the jobs. The first step is to determine an assignment of the groups to the adjacent resource

units, and an ordering between those groups that are assigned to at least one common adjacent resource unit. We call this the Group Assignment Problem (GAP). Let F_{GAP} denote the set of all feasible solutions of the GAP. For a solution $a \in F_{\text{GAP}}$, we have an ordering of the groups that are assigned to the same adjacent resource. This ordering implies additional precedence relations in the original problem, i.e. if group g and group h share an adjacent resource unit in a solution of the GAP and g is scheduled before h then no job of group h can start before the completion of all jobs in group g. After adding these implied precedence relations, the adjacent resources do not have to be considered anymore. The resulting problem is a TCPSP without adjacent resources (which we refer to as the *resulting TCPSP*). Denote this resulting TCPSP for $a \in F_{\text{GAP}}$ by TCPSP(a). The second step is to find a low cost solution of the resulting TCPSP, which can be done by employing existing methods, e.g. the method proposed in Chapter 4.

By the above construction we can rewrite the TCPSP with adjacent resources as

$$\min_{a \in F_{\text{GAP}}} Opt\left(\text{TCPSP}(a)\right) \quad ,$$

where $Opt(\cdot)$ denotes the optimal value of the resulting TCPSP.

In the following we first describe the GAP. We design the GAP in such a way that the original problem has a feasible solution if and only if the GAP has a feasible solution and such that each feasible solution of the GAP can be extended to a feasible solution of the original problem. Afterwards, we present an ILP formulation of the GAP which can be used to solve it. Finally, we treat the resulting TCPSP problem resulting after fixing the assignment of the adjacent resource units according to a solution of the GAP.

5.4.1 The group assignment problem (GAP)

The feasibility of the original problem depends on the adjacent resource capacity and requirements, and on the timing constraints of the jobs. It does not depend on the renewable resources, since we assume that we can hire unlimited additional capacity of the renewable resources. For each group G_g we have to determine an adjacent resource assignment and a start and completion time, denoted by s_g and c_g, respectively. The start and completion time of a group implies a time window $[s_g, c_g]$ in which it should be possible to process all jobs of group G_g, respecting the timing constraints of the jobs. Note that the processing time of a group is not a priori fixed.

The start and completion time of a group have to be consistent with the release dates and deadlines of the jobs in this group. This gives rise to the following definitions: Earliest Start Time of a group $EST_g := \min_{J_j \in G_g}\{r_j\}$, Latest Start Time of a group $LST_g := \min_{J_j \in G_g}\{d_j - p_j\}$, Earliest Completion Time of a group $ECT_g := \max_{J_j \in G_g}\{r_j + p_j\}$, and Latest Completion Time of a group $LCT_g := \max_{J_j \in G_g}\{d_j\}$. In order to guarantee processing of all jobs in a group,

the start and completion of a group should be such that $s_g \in [EST_g, LST_g]$ and $c_g \in [ECT_g, LCT_g]$.

It is however not possible to choose s_g and c_g independently within the mentioned intervals. For example, if we choose the start time of a group G_g to be large, then choosing the completion time of group G_g equal to ECT_g might not be feasible. The reason for this is that there can be a path in the AoN network from J_g^s to J_g^c such that adding all processing times and time lags on this path exceeds the value $c_g - s_g$. This means the time window $[s_g, c_g]$ is not large enough to process the jobs in G_g. We define therefore a minimum processing time $p_g^{\min}$ for group G_g. To obtain the value of $p_g^{\min}$, we schedule J_g^s as late as possible (i.e. set $s_g = LST_g$) and given this start time of J_g^s we schedule J_g^c as early as possible. The difference between these start and completion times gives $p_g^{\min}$. The renewable and adjacent resource capacity is not considered, only timing constraints play a role. Note that $p_g^{\min}$ is determined by some critical path in the AoN network or by the release dates and deadlines. Only in the first case the minimum processing time requirement forms an additional constraint.

The above constraints focus on a single group. But there are also restrictions between different groups. Whenever there is a path in the AoN network from J_g^s to J_h^c we call group G_g a predecessor of group G_h, and by $P'_h \subset \mathcal{G}$ we denote the set of predecessors of group G_h. Again, if the start of group G_g is chosen large, it might not be possible to schedule the completion of group G_h at ECT_h. This gives rise to the definition of start-completion time lags τ'_{gh} between groups, i.e. $s_g + \tau'_{gh} \leq c_h$ for $G_g \in P'_h$. As before with the calculation of $p_g^{\min}$, to calculate τ'_{gh}, we schedule group G_g to start as late as possible, and given this start time, we schedule group G_h as early as possible. The difference between these values gives τ'_{gh}. (In particular $\tau'_{gg} = p_g^{\min}$.) Since we have start-completion time lags, it is possible that G_g is a predecessor of G_h and simultaneously that G_h is a predecessor of G_g. This implies that these groups have to be scheduled in parallel for some duration.

The Group Assignment Problem (GAP) can now be formally stated as follows: *We are given a set $\mathcal{G}$ of groups, i.e. $\mathcal{G} = \{G_1, ..., G_m\}$, precedence relations among the groups, and an adjacent resource $\bar{R}$. The adjacent resource has a capacity of $\bar{Q}$ resource units. Each group $G_g \in \mathcal{G}$ has to be assigned to $\bar{q}_g$ adjacent resource units for its entire duration. Group G_g has to start between EST_g and LST_g, and to complete between ECT_g and LCT_g with a duration of at least p_g^{min}. Whenever there is a precedence relation between two groups G_g and G_h, i.e. $G_g \in P'_h$, group G_g has to start at least τ'_{gh} time units before the completion of G_h. A solution of the GAP is an assignment of the groups to the adjacent resource units, and a schedule of the groups that respect the time windows and the precedence relations.*

The following shows that the feasibility of the GAP and the original problem are equivalent.

Observation 5.1. *The Group Assignment Problem has a feasible solution if and only if the TCPSP with Adjacent Resources has a feasible solution.*

Proof. Suppose that the original problem has a feasible solution. This solution of the TCPSP with adjacent resources specifies the adjacent resource assignment of the groups and the start times of the jobs. From the start times of the jobs we can derive the start and completion times of the groups. By definition, these start and completion times satisfy all the time restrictions for the groups in the GAP. So, the GAP has a feasible solution.

Now suppose that the GAP has a feasible solution. From the GAP solution we have an adjacent resource assignment and start and completion times of the groups. Since the renewable resources can be hired, the feasibility of the original problem only depends on the timing constraints on the jobs. By definition of the time windows of the groups, their minimum duration, and the time lags it is ensured that feasible start times of jobs exists, i.e. by scheduling the jobs as early as possible. □

As mentioned in the previous section, the problem of finding a feasible solution for the overall problem is already NP-complete. Since this feasibility is equivalent to the feasibility of the GAP, the given decomposition has the advantage that the hard feasibility question is already treated at an early stage.

ILP formulation of the GAP

To find a solution for the GAP we model it as an ILP. For this, we define the following variables. The variable $a_g \in [0, \bar{Q} - \bar{q}_g]$ gives the adjacent resource assignment of group G_g, i.e. group G_g is assigned to the interval $[a_g, a_g + \bar{q}_g]$. Variables $s_g \in [EST_g, LST_g]$ and $c_g \in [ECT_g, LCT_g]$ are the start and completion time of group G_g as before. Finally, binary variables x_{gh}, y_{gh}, z_{gh} are used in the modeling to avoid overlap in the adjacent resource assignment.

The GAP is represented by the following set of constraints (directly followed by an explanation):

$$c_g - s_g \geq p_g^{\min} \qquad \forall G_g \qquad (5.1)$$

$$c_h - s_g \geq \tau'_{gh} \qquad \forall G_g \in P'_h \qquad (5.2)$$

$$\bar{Q} \cdot (z_{gh} + y_{gh}) \geq a_h + \bar{q}_h - a_g \qquad \forall g < h \qquad (5.3)$$

$$\bar{Q} \cdot (1 + z_{gh} - y_{gh}) \geq a_g + \bar{q}_g - a_h \qquad \forall g < h \qquad (5.4)$$

$$T \cdot (1 - z_{gh} + x_{gh}) \geq c_h - s_g \qquad \forall g < h \qquad (5.5)$$

$$T \cdot (2 - z_{gh} - x_{gh}) \geq c_g - s_h \qquad \forall g < h \ . \qquad (5.6)$$

The constraints (5.1) and (5.2) are clear from the definition of $p_g^{\min}$ and τ'_{gh}. What remains it to ensure that no two groups using a common adjacent resource unit overlap in time. It is sufficient to check this for each index pair (g, h) with $g < h$. Whenever the groups overlap on the adjacent resource, the right hand sides of (5.3) and (5.4) are both larger than 0, implying that $z_{gh} = 1$ independent if $y_{gh} = 0$ or $y_{gh} = 1$. If they do not overlap, at least one of the right hand sides is at most 0. Due

to the free choice for the variable y_{gh} the variable z_{gh} is now unrestricted. Whenever the groups overlap in time the right hand sides of (5.5) and (5.6) are both larger than 0, implying that $z_{gh} = 0$. Again, due to the variable x_{gh} the variable z_{gh} is unrestricted otherwise. Thus, constraints (5.3)-(5.6) ensure that no two groups have a conflict, since it is impossible for groups G_g and G_h to overlap in time and on the adjacent resource simultaneously (the two possibilities lead to different values of z_{gh}).

It is not necessary to restrict the variables s_g, c_g, and a_g to be integer. If the obtained solution contain a non-integer value for one of these variables we can round the value down without violating the constraints, since all input parameters are integers. No objective function is needed since we are at this stage only looking for a feasible assignment.

Adding cutting planes

The ILP given by (5.1)-(5.6) gives a complete description of the GAP. By adding additional valid inequalities (cutting planes) we can reduce the computation time required to solve the ILP. We propose two types of cutting planes.

For a subset of groups that have a cumulative adjacent resource requirement more than $\bar{Q}$, we know that at least two group have a resource overlap in any solution. We call such a subset a *resource conflicting set*. A *minimum resource conflicting set* is a resource conflicting set that does not remain a resource conflicting set if any of the groups is removed from it. Let S be such a minimum resource conflicting set. Then we can add the following constraint to our model:

$$\sum_{G_g, G_h \in S, g<h} z_{gh} \geq 1 \quad \forall S \ . \tag{5.7}$$

Similarly we can look at a *minimum time conflicting set*. When a set of groups have a cumulative minimum duration larger than T, we know that at least two group share time units, and thus not adjacent resource units. This implies that not all z_{gh} values can be 1. Let S' be such a minimum time conflicting set. We can add:

$$\sum_{G_g, G_h \in S', g<h} z_{gh} \leq \frac{1}{2}|S'|\big(|S'|-1\big) - 1 \quad \forall S' \ . \tag{5.8}$$

As we show in Section 5.5, adding constraints (5.7) and (5.8) to the ILP formulation (5.1)-(5.6), can significantly reduce the computational time needed to solve the GAP.

5.4.2 Deriving and solving the resulting TCPSP

In this section we treat the problem that remains after assigning the groups to the adjacent resource. We show how the solution of the GAP can be incorporated into

the AoN network, such that the group structure and adjacent resource do not have to be considered anymore when searching for a low cost schedule for the jobs.

A solution of the GAP gives us for each group G_g a start time (s_g) and a completion time (c_g), and an adjacent resource assignment ($[a_g, a_g + \bar{q}_g]$). We could impose these start and completion time of a group on the jobs within that group by redefining the release dates and deadlines of the jobs, i.e. $\bar{r}_j := \max\{r_j, s_g\}$ and $\bar{d}_j := \min\{d_j, c_g\}$ for all $J_j \in G_g$. However, this would unnecessarily restrict the resulting TCPSP, i.e. hiring a lot of renewable resources might be unavoidable, since these aspects did not play a role in the GAP. It is sufficient to keep the order in which two groups G_g and G_h are assigned to the same adjacent resource unit. Thus, if group G_g and G_h share an adjacent resource unit and group G_g completes before group G_h starts (all jobs of group G_g complete before any job of group G_h starts), we add a precedence relation from job J_g^c to J_h^s. In this way, different solutions for the resulting TCPSP may result in different start and completion times of the groups, however the assignment of the groups to the adjacent resources remains valid to all these solutions. Therefore, adding precedence relations restricts the resulting TCPSP less than imposing the start and completion times from the GAP solution, and still guarantees feasibility of the adjacent resource assignment. Note that the release dates and deadlines of jobs may require adjustment such that they are consistent with the added precedence relations. The dummy jobs J_g^s and J_g^c now can be removed from the AoN network. When removing a dummy job all predecessors of the dummy job become predecessors of all the successors of the dummy job.

In Figure 5.5(a) the Activity-on-Node network of the resulting TCPSP is given for the feasible adjacent resource assignment presented in Figure 5.2 for the example in Section 5.2.3. After removing the dummy start and completion nodes, the AoN network reduces to the network given in Figure 5.5(b).

To find a solution of the resulting TCPSP any known method from the literature can be employed. We employ the method of [34] since it fits the resulting TCPSP best. The method is designed to deal with hiring of renewable resources and work in overtime. Since our model does not include working in overtime, the method boils down to the following two stage procedure. The first stage of the heuristic constructs schedules by means of randomized sampling. In this stage many schedules are constructed by adding jobs one by one into the schedule. Jobs can only be put into the schedule if all their predecessors have been scheduled. The selection of the job to be scheduled next is based on a randomization which is biased to jobs which get close to their deadline. The second stage consists of a neighborhood search. This neighborhood search repeatedly removes a small subset of the jobs from the schedule and reinsert them by means of an ILP solution. This last technique is based on the work of [64].

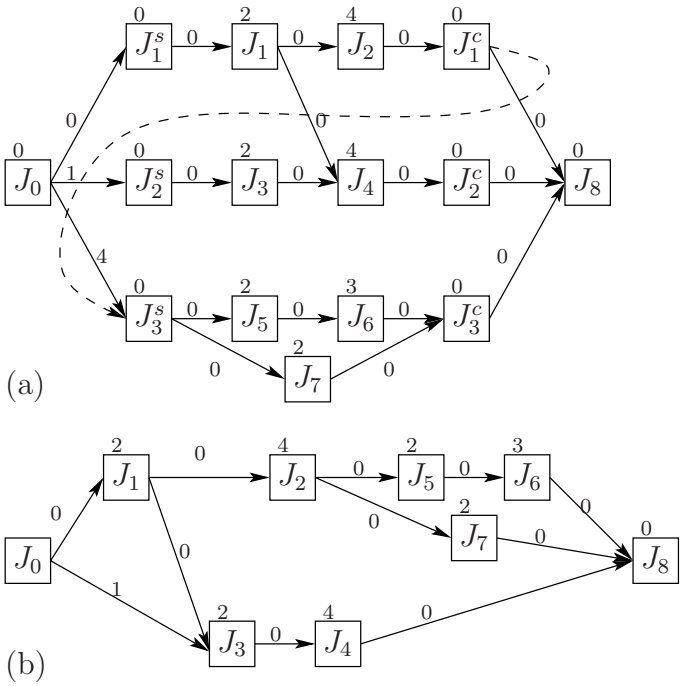


Figure 5.5: Example project: Activity-on-Node network of the resulting TCPSP.

5.4.3 Objective function

Up to now, we considered just an arbitrary feasible solution $a \in F_{\mathrm{GAP}}$ of the GAP. However, if we choose different feasible solutions of the GAP we obtain different resulting TCPSP's. The following example illustrates how different solutions of the GAP imply different precedence relations in the resulting TCPSP, and hence different solutions for the original problem.

Consider the example project as given in Section 5.2.3, but now with the duration of job J_4 reduced to 2, i.e. $p_4 = 2$. In Figure 5.6 two different assignments of the adjacent resource together with the corresponding schedules of the renewable resources are given. Since a second painting crew has to be hired in time buckets 4 and 5 for Assignment 1 and not for Assignment 2, Assignment 2 results in a better solution of the resulting TCPSP. Therefore, it is a better solution of the original problem.

This raises the question whether it is possible to predict somehow the quality of the overall solution by assigning some quality measure to the assignments within the ILP. The above example illustrates that it may be beneficial to have long durations

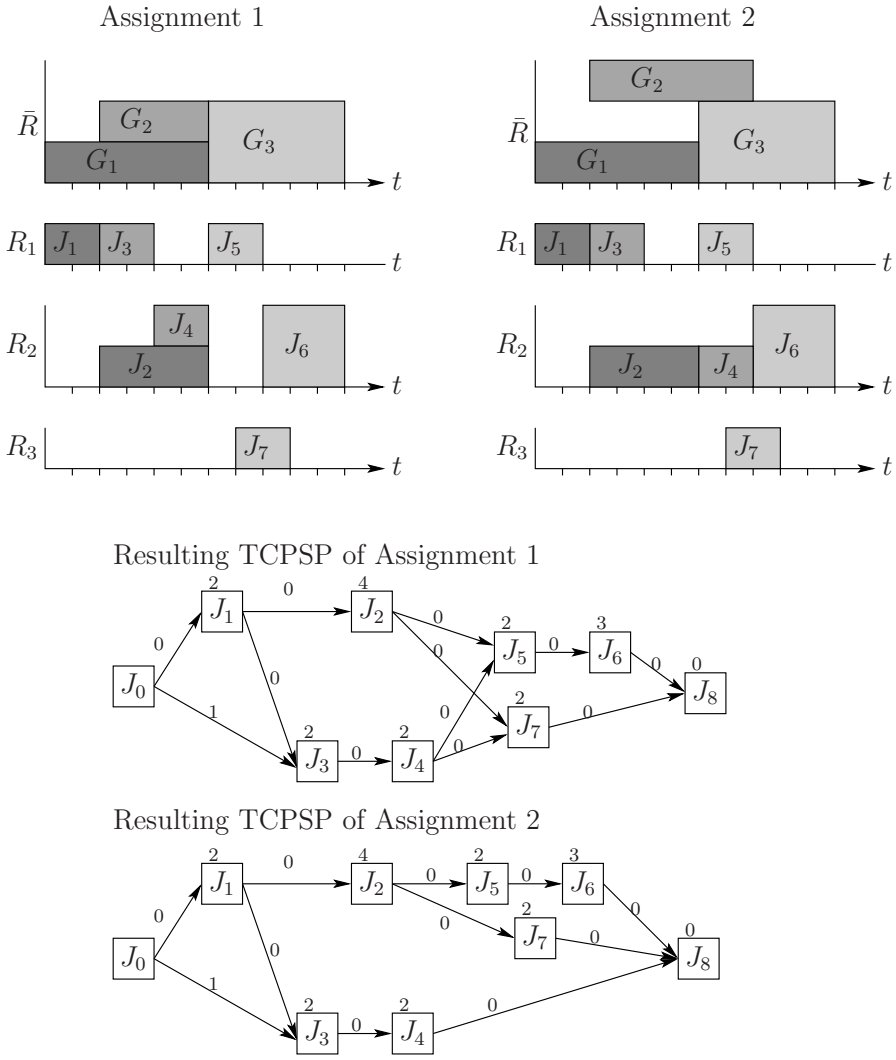


Figure 5.6: Example project: Two alternative assignments

for the groups. An assignment of groups with long durations allows more flexibility in scheduling the jobs in the resulting TCPSP. One may expect to find better job schedules when there is more room for the jobs. To get a GAP solution with long group durations, we propose the following two objective functions for the GAP.

The first objective function simply maximizes the total absolute group duration:

$$\text{ABS} := \text{maximize} \sum_{G_g \in \mathcal{G}} (c_g - s_g) \quad .$$

The second objective function maximizes the total group duration relative to the minimum duration of the group:

$$\text{REL} := \text{maximize} \sum_{G_g \in \mathcal{G}} \left(\frac{c_g - s_g}{p_g^{\min}} \right) \quad .$$

For the assignments given in Figure 5.6 both measures have a preference to Assignment 2, i.e. for Assignment 1 we have ABS $= 6{+}4{+}5 = 15$ and REL $= \frac{6}{6}{+}\frac{4}{4}{+}\frac{5}{5} = 3$, and for Assignment 2 we have ABS $= 6 + 6 + 5 = 17$ and REL $= \frac{6}{6} + \frac{6}{4} + \frac{5}{5} = 3\frac{1}{2}$.

One may think of various other objective criteria for assignments. However, our goal is not to investigate many different objectives, but to see if the objectives influence the overall outcome and if one objective dominates the other. Therefore, in next section we study the effect of adding these objective functions to the GAP on the quality of the solution found for the original problem.

5.5 Computational tests

In this section we report on computational experiments for the presented decomposition approach. The aim of this section is to show that the presented decomposition approach gives a flexible approach to handle the TCPSP with adjacent resources. Since no solution methods for this problem are known and since also the possible relaxations of the problem (e.g. relaxing the adjacency constraints of the adjacent resources) do not lead to useful lower bounds, it is hard to judge the overall quality of the achieved solutions. However, we show that by using different objective functions for the GAP we obtain different solutions for the original problem. The test results show no dominance between the proposed GAP objective functions.

The computations are performed on a computer with a dual 3.40GHz processor and 1.00GB of RAM memory. The ILP's are solved with CPLEX version 11.1.

5.5.1 Generating instances

For the generation of the instances we make use of the project generator ProGen, described in [51, 52, 53], which generates instances for the RCPSP. This generator uses various parameters to govern the properties of the precedence network and the

resource requirements of the instances generated. In this work, we do not fully explore all possible parameter variations but restrict ourself to the parameter values that are mentioned in [53] as leading to instances that are interesting to study. Further below we mention the precise settings.

The instances of the TCPSP with one 1-dimensional adjacent resource are generated in three steps. To construct an instance, we first generate a set of groups with their adjacent resource requirement and the precedence relations among them, and determine the capacity of the adjacent resource. In the second step, we generate a set of renewable resources, for each group a set of jobs with corresponding renewable resource requirements and precedence relations, and the renewable resource capacities. In the final step, we convert the precedence relations between groups into precedence relations between jobs of those groups. Whenever there is a precedence relation from group G_g to group G_h we add a precedence relation from a randomly selected job from G_g to a randomly selected job from G_h.

We generate 4 different sets of instances that vary in the number of groups and the number of jobs per group. However, for all instances we let the total number of jobs be equal to 120. This allows us to somehow compare the computation times. In the four different sets of instances the number of groups are 8, 10, 12, and 15 with each group having 15, 12, 10 and 8 jobs, respectively. For each set of instances we generate 100 instances.

The parameters of ProGen are chosen as follows. The network complexity (the average number of non-redundant arcs per node) of the group and job networks is 1.5 with an allowed deviation of 0.3. Each group requires up to 10 units of the adjacent resource and the resource strength (a measure of the resource scarceness through a scaling in the convex combination of the minimum and maximum resource demand) of the adjacent resource is 0.2. The number of renewable resources varies between 5 and 10 and each job requires up to 5 of these resources for processing. The renewable resource factor (the average portion of resources required per job) is 0.5 and the renewable resource strength is 0.2. The processing times of the jobs are drawn uniformly from $\{1, \ldots, 15\}$. For details on these measures we refer to [53].

In order to transform these instances to instances of the time-constrained project scheduling problem, we define the cost of hiring one additional renewable resource unit in one time bucket to be 1, i.e. $c_{kt} = 1$ for all k, t. Furthermore, we define the release date of the project as 0 and derive a deadline that applies to all jobs. The deadline of an instance is determined in the same way as projects generated by ProGen have due dates. Let MP denote the minimum project length, which is defined by the longest path in the AoN network, and let $\bar{T}$ be the sum of all processing times. $\bar{T}$ is thereby an upper bound on the project length. Now we define the project deadline as $MP + \delta\left(\bar{T} - MP\right)$. The scaling parameter δ is called the due date factor. The smaller this value the smaller the deadline becomes, and as a consequence the resource requirements play a more important role. We vary the value of δ between 0.05 and 0.20. Varying δ while keeping the remaining data fixed leads to different instances, which we treat as different versions of the same instance.

Using these general release date and deadline, release dates and deadlines of the jobs are derived by making them consistent with the precedence relations.

The parameter values are chosen in this way to achieve instances that are useful for testing, that is, instances that are not easy to solve. By varying the due date factor we can move from sets with many instances having 0 cost solutions (large due date factor) to sets with many infeasible instances (small due date factor). Due to the adjacency requirements of the adjacent resource, there is no guarantee that there exist feasible solutions for the generated instances. However, infeasibility of an instance can already be determined after solving the GAP.

5.5.2 Solving the GAP

In the first series of tests we concentrate on the group assignment problem (GAP) only. For the four sets of instances (100 instances with 8, 10, 12, and 15 groups, respectively), we explore the effect of varying the due date factor on the number of feasible solutions, and the effect of using cutting planes in combination with the proposed objective functions on the computation time.

For each of the 400 instances we solve the GAP 48 times; 4 different due date factors, 3 different objective functions (NO meaning without objective, ABS and REL) each with 4 different settings of the cutting planes. For these computations we have set a time limit of 1800s on the computation time spent on one instance. Tables 5.3, 5.4, 5.5 and 5.6 summarize the results for the instances with 8, 10, 12 and 15 groups per instance, respectively. They display the number of feasible and infeasible instances and the number of instances for which the time limit of 1800s is exceeded. Additionally, they contain the average computation time per instance, and, more specific, also the average computation time for the feasible and infeasible instances separately. Table 5.6 contains only the test results for due date factors 0.15 and 0.20. For smaller due date factors, the computation time required becomes large, i.e. more than half of the tests exceed the time limit of 1800s.

As mentioned earlier and now demonstrated by the computational tests, a decrease of the due date factor leads to a growing fraction of infeasible instances in the set. By increasing the due date factor from 0.05 to 0.10, the number of feasible instances grows a lot, but still half of the instances are infeasible in the set with 8 groups in each instance. Further increasing the due date factor to 0.15 seems to lead to interesting instances for testing the resulting TCPSP, i.e. we expect many of the instances to be feasible but also that hardly any of the instances has a feasible job schedule that does not hire additional renewable resources.

Regarding the cutting planes we can draw a firm conclusion. It is best to employ the cutting planes of (5.7), called CP1, and not the cutting planes of (5.8), called CP2. The improvement obtained by adding CP1 is in particularly large for the computation time spent on the infeasible instances. Obviously, the computation time increases as the number of groups increases.

We test instances with up to 15 groups. The results show that the computation

time spent becomes large, but by tuning CPLEX this can be dealt with. We point out that for practical applications 15 groups is already quite a lot. We end this section by a note on the use of CPLEX.

Note. *It is surprising to see that CPLEX 11.1 is able to determine the feasibility of an instance faster if there is an objective function in the GAP. CPLEX's search strategies are influenced by the objective function.*

For a subset of instances that exceed the time limit we have tested different settings of CPLEX 11.1. For each of these instances we are able to reduce the required computation time to a fraction of a second by changing the settings. However, the optimal setting differs from instance to instance. We use therefore the default settings for all computations.

5.5.3 Solving the resulting TCPSP

In the second series of tests we compare the cost of the resulting TCPSP when different objective function are used in the GAP. In these tests we use a due date factor of 0.15 and use only the cutting planes of (5.7), that is CP1, as motivated in the previous section. For each of the 100 instances of the four sets, we solve the GAP and the resulting TCPSP 3 times, once without an objective for the GAP and next with the two proposed objectives. As a consequence of the chosen cost for hiring extra renewable resources, the objective value of the resulting TCPSP equals the amount of resource units hired in total. Solving the resulting TCPSP takes on average 35 seconds and never more than 3 minutes.

Table 5.7 displays the results of these tests. Note that only one instance in the set of 15 groups exceeds the time limit of 1800s in the GAP. When we consider the cost of the TCPSP solution, we see that all GAP objectives result in about the same averages values (second to last column).

To see whether the different GAP solutions found by the three objectives result in large differences, we can compare them with the overall best found solution. The rightmost column of Table 5.7 gives the average values of these best found solution. There is a significant difference between the average best found solution and the average solution found by using just one specific GAP objective. So, the overall solution found heavily depends on the GAP solution found.

In Table 5.8 we count the number of times the best found solution is due to either the use of no objective, the ABS objective or the REL objective. The results show that the use of either objective is equally good, the number of times they lead to the best found solution is about the same. So, no choice of objective function dominates the other.

The above results show that it is hard to predict just on the basis of the GAP solution what the objective value of the resulting TCPSP will be, at least with the presented GAP objectives. Therefore, it is worth to generate not only one solution for the GAP, but to generate a number of different GAP solutions by using different objectives, and solve for each of them the corresponding TCPSP.

Number of groups	Due date factor	GAP objective	Using cuts	Time	Number feasible	Time feasible	Number infeasible	Time infeasible	Number time exceeded
8	0.05	NO	CP1 & CP2	0.01	1	0.02	99	0.01	0
			CP1	0.01	1	0.02	99	0.01	0
			CP2	24.07	1	0.00	98	6.20	1
			-	20.95	1	0.09	98	3.00	1
		ABS	CP1 & CP2	0.01	1	0.02	99	0.01	0
			CP1	0.01	1	0.02	99	0.01	0
			CP2	0.33	1	0.02	99	0.33	0
			-	0.17	1	0.03	99	0.17	0
		REL	CP1 & CP2	0.01	1	0.02	99	0.01	0
			CP1	0.00	1	0.02	99	0.00	0
			CP2	0.16	1	0.02	99	0.16	0
			-	0.23	1	0.02	99	0.23	0
8	0.10	NO	CP1 & CP2	25.78	50	0.15	49	15.72	1
			CP1	26.88	50	0.11	50	53.64	0
			CP2	317.51	49	48.70	36	65.67	15
			-	271.58	50	1.13	38	144.76	12
		ABS	CP1 & CP2	0.41	50	0.08	50	0.74	0
			CP1	0.54	50	0.08	50	1.00	0
			CP2	3.48	50	1.05	50	5.90	0
			-	3.45	50	0.25	50	6.64	0
		REL	CP1 & CP2	0.35	50	0.10	50	0.59	0
			CP1	0.46	50	0.08	50	0.83	0
			CP2	6.21	50	0.55	50	11.88	0
			-	12.07	50	0.20	50	23.94	0
8	0.15	NO	CP1 & CP2	18.32	95	0.33	4	0.01	1
			CP1	3.63	95	0.05	5	71.60	0
			CP2	29.26	95	11.85	4	0.02	1
			-	18.51	95	0.54	4	0.01	1
		ABS	CP1 & CP2	0.07	95	0.06	5	0.33	0
			CP1	0.08	95	0.06	5	0.48	0
			CP2	0.55	95	0.33	5	4.82	0
			-	0.35	95	0.13	5	4.52	0
		REL	CP1 & CP2	0.14	95	0.06	5	1.78	0
			CP1	0.13	95	0.06	5	1.45	0
			CP2	0.69	95	0.32	5	7.82	0
			-	2.21	95	0.15	5	41.43	0
8	0.20	NO	CP1 & CP2	0.02	99	0.02	1	0.09	0
			CP1	0.02	99	0.02	1	0.13	0
			CP2	0.05	99	0.04	1	1.11	0
			-	0.03	99	0.03	1	0.34	0
		ABS	CP1 & CP2	0.02	99	0.02	1	0.06	0
			CP1	0.02	99	0.02	1	0.05	0
			CP2	0.05	99	0.05	1	0.42	0
			-	0.03	99	0.03	1	0.16	0
		REL	CP1 & CP2	0.02	99	0.02	1	0.06	0
			CP1	0.02	99	0.02	1	0.05	0
			CP2	0.06	99	0.06	1	0.58	0
			-	0.03	99	0.03	1	0.11	0

Table 5.3: Solving the GAP: 8 groups

Number of groups	Due date factor	GAP objective	Using cuts	Time	Number feasible	Time feasible	Number infeasible	Time infeasible	Number time exceeded
10	0.05	NO	CP1 & CP2	39.93	8	36.63	90	1.11	2
			CP1	41.87	8	0.14	90	6.51	2
			CP2	710.02	6	13.20	56	45.04	38
			-	700.20	8	0.53	57	123.07	35
		ABS	CP1 & CP2	1.08	8	0.78	92	1.11	0
			CP1	0.40	8	0.48	92	0.39	0
			CP2	17.22	8	52.48	92	14.15	0
			-	12.21	8	6.51	92	12.71	0
		REL	CP1 & CP2	0.17	8	0.39	92	0.15	0
			CP1	0.24	8	0.21	92	0.24	0
			CP2	15.43	8	11.06	92	15.81	0
			-	9.91	8	8.43	92	10.03	0
10	0.10	NO	CP1 & CP2	682.16	51	26.87	12	20.37	37
			CP1	632.38	53	19.61	13	76.79	34
			CP2	849.35	50	6.67	3	0.34	47
			-	838.70	50	20.46	5	369.19	45
		ABS	CP1 & CP2	77.94	55	4.90	44	130.10	1
			CP1	33.78	55	1.69	45	73.00	0
			CP2	363.48	55	34.14	33	390.00	12
			-	330.29	55	20.42	34	356.04	11
		REL	CP1 & CP2	44.60	55	5.36	45	92.55	0
			CP1	25.62	55	1.28	45	55.37	0
			CP2	323.58	55	25.06	33	284.24	12
			-	336.13	55	9.72	30	202.62	15
10	0.15	NO	CP1 & CP2	0.21	100	0.21	0	-	0
			CP1	8.25	100	8.25	0	-	0
			CP2	72.73	96	0.76	0	-	4
			-	18.26	99	0.26	0	-	1
		ABS	CP1 & CP2	0.15	100	0.15	0	-	0
			CP1	0.07	100	0.07	0	-	0
			CP2	0.45	100	0.45	0	-	0
			-	0.60	100	0.60	0	-	0
		REL	CP1 & CP2	0.13	100	0.13	0	-	0
			CP1	0.08	100	0.08	0	-	0
			CP2	1.47	100	1.47	0	-	0
			-	0.24	100	0.24	0	-	0
10	0.20	NO	CP1 & CP2	0.08	100	0.08	0	-	0
			CP1	0.06	100	0.06	0	-	0
			CP2	0.12	100	0.12	0	-	0
			-	0.08	100	0.08	0	-	0
		ABS	CP1 & CP2	0.08	100	0.08	0	-	0
			CP1	0.04	100	0.04	0	-	0
			CP2	0.12	100	0.12	0	-	0
			-	0.11	100	0.11	0	-	0
		REL	CP1 & CP2	0.07	100	0.07	0	-	0
			CP1	0.04	100	0.04	0	-	0
			CP2	0.13	100	0.13	0	-	0
			-	0.10	100	0.10	0	-	0

Table 5.4: Solving the GAP: 10 groups

Number of groups	Due date factor	GAP objective	Using cuts	Time	Number feasible	Time feasible	Number infeasible	Time infeasible	Number time exceeded
12	0.05	NO	CP1 & CP2	252.44	5	0.71	81	0.49	14
			CP1	252.26	5	0.35	81	0.29	14
			CP2	1371.09	3	2.98	21	14.16	76
			-	1381.93	2	0.28	21	0.40	77
		ABS	CP1 & CP2	58.53	7	125.91	92	48.98	1
			CP1	27.58	7	2.62	92	27.55	1
			CP2	506.60	7	277.06	75	217.60	18
			-	399.02	6	1.28	76	98.60	18
		REL	CP1 & CP2	59.96	8	249.46	92	43.48	0
			CP1	18.30	8	36.96	92	16.68	0
			CP2	487.97	5	1.78	73	125.86	22
			-	504.26	6	55.32	70	98.48	24
12	0.10	NO	CP1 & CP2	756.68	55	1.09	3	2.04	42
			CP1	705.06	58	5.22	3	0.80	39
			CP2	865.61	52	3.06	0	-	48
			-	922.57	51	79.53	0	-	49
		ABS	CP1 & CP2	442.67	69	41.56	11	490.81	20
			CP1	285.46	70	22.83	17	208.70	13
			CP2	640.04	66	59.04	1	706.55	33
			-	650.06	63	49.25	4	625.74	33
		REL	CP1 & CP2	396.34	68	20.07	13	382.22	19
			CP1	249.77	70	10.96	21	381.40	9
			CP2	680.08	61	9.22	3	881.63	36
			-	715.34	60	35.86	2	490.74	38
12	0.15	NO	CP1 & CP2	0.70	100	0.70	0	-	0
			CP1	0.31	100	0.31	0	-	0
			CP2	0.71	100	0.71	0	-	0
			-	0.49	100	0.49	0	-	0
		ABS	CP1 & CP2	0.44	100	0.44	0	-	0
			CP1	0.22	100	0.22	0	-	0
			CP2	0.71	100	0.71	0	-	0
			-	0.40	100	0.40	0	-	0
		REL	CP1 & CP2	0.42	100	0.42	0	-	0
			CP1	0.24	100	0.24	0	-	0
			CP2	0.78	100	0.78	0	-	0
			-	0.48	100	0.48	0	-	0
12	0.20	NO	CP1 & CP2	0.46	100	0.46	0	-	0
			CP1	0.21	100	0.21	0	-	0
			CP2	0.36	100	0.36	0	-	0
			-	0.18	100	0.18	0	-	0
		ABS	CP1 & CP2	0.31	100	0.31	0	-	0
			CP1	0.14	100	0.14	0	-	0
			CP2	0.40	100	0.40	0	-	0
			-	0.17	100	0.17	0	-	0
		REL	CP1 & CP2	0.32	100	0.32	0	-	0
			CP1	0.14	100	0.14	0	-	0
			CP2	0.50	100	0.50	0	-	0
			-	0.20	100	0.20	0	-	0

Table 5.5: Solving the GAP: 12 groups

Number of groups	Due date factor	GAP objective	Using cuts	Time	Number feasible	Time feasible	Number infeasible	Time infeasible	Number time exceeded
15	0.15	NO	CP1 & CP2	96.35	95	6.68	0	-	5
			CP1	19.87	99	1.89	0	-	1
			CP2	95.37	95	5.65	0	-	5
			-	18.89	99	0.90	0	-	1
		ABS	CP1 & CP2	4.73	100	4.73	0	-	0
			CP1	0.85	100	0.85	0	-	0
			CP2	4.82	100	4.82	0	-	0
			-	1.64	100	1.64	0	-	0
		REL	CP1 & CP2	4.19	100	4.19	0	-	0
			CP1	0.80	100	0.80	0	-	0
			CP2	5.44	100	5.44	0	-	0
			-	1.01	100	1.01	0	-	0
15	0.20	NO	CP1 & CP2	5.40	100	5.40	0	-	0
			CP1	0.96	100	0.96	0	-	0
			CP2	4.13	100	4.13	0	-	0
			-	0.33	100	0.33	0	-	0
		ABS	CP1 & CP2	3.90	100	3.90	0	-	0
			CP1	0.50	100	0.50	0	-	0
			CP2	3.60	100	3.60	0	-	0
			-	1.03	100	1.03	0	-	0
		REL	CP1 & CP2	3.42	100	3.42	0	-	0
			CP1	0.58	100	0.58	0	-	0
			CP2	3.63	100	3.63	0	-	0
			-	0.49	100	0.49	0	-	0

Table 5.6: Solving the GAP: 15 groups

Number of groups	GAP objective	Number feasible	Average time GAP(feasible)	Average Time resulting TCPSP	Average TCPSP objective value	Average value best found solution
8	NO	95 (5 infeasible)	0.05	42.42	54.43	37.6
	ABS	95 (5 infeasible)	0.05	40.59	57.28	
	REL	95 (5 infeasible)	0.06	40.92	56.85	
10	NO	100	8.30	36.53	56.17	29.99
	ABS	100	0.07	35.37	50.21	
	REL	100	0.08	36.89	43.27	
12	NO	100	0.26	34.05	53.79	33.23
	ABS	100	0.17	32.51	54.34	
	REL	100	0.19	33.18	53.41	
15	NO	99 (1 time exceeded)	1.89	30.51	60.04	32.64
	ABS	100	0.84	30.67	60.87	
	REL	100	0.81	30.57	53.65	

Table 5.7: Solving the TCPSP: Comparing Objectives

8 Groups			10 Groups		
NO	ABS	REL	NO	ABS	REL
62	58	62	34	47	44
12 Groups			15 Groups		
NO	ABS	REL	NO	ABS	REL
39	32	44	39	34	37

Table 5.8: Solving the TCPSP: Times best

5.6 Concluding remarks

We presented a decomposition approach for the Time-Constrained Project Scheduling Problem with adjacent resources, which focuses in an early stage on the issue of a feasible assignment of groups to adjacent resource units. This is important since the NP-completeness of the feasibility problem forms the main obstacle to develop fast heuristic solution approaches for the overall problem. The presented approach detects infeasibility of an instance of the TCPSP with adjacent resources by solving the corresponding GAP in the first step. In case an instance is feasible, the first step gives also a solution for the group assignment and the order of the groups on the adjacent resources, which can be extended to an overall feasible solution in the second step. The test results show no clear dominance among the presented GAP objective functions. Finding good solutions by one specific objective remains problematic, but by solving each instance multiple times with different GAP objective functions, the quality of the generated schedules improves significantly.

The presented method can easily be extended to include 2-dimensional adjacent resources and multiple adjacent resources, by modifying the ILP formulation. However, the computational time required to solve the GAP's will become a bigger issue.

For future research it would be interesting to see whether other GAP objectives are more successful. In particular, one might choose an objective function that is more dependent on the job characteristics. However, preliminary tests have shown that using weighted versions of the presented objectives does not improve the results (the weights of a group corresponds to the renewable resource requirements of the jobs in that group). Besides exploring a fixed objective, the presented decomposition can be the basis of a feedback between the GAP and the resulting TCPSP, where the outcome of the resulting TCPSP can influence the GAP objective before resolving. This may lead to a local search approach, where the weights and the different type of objectives of the GAP can be used as a solution space. Adapting the weights can be seen as some sort of intensification phase and the change of the objective as some sort of diversification phase of the search process. To make such an approach successful, intelligent ways of changing the weights based on the outcome of the TCPSP have to be developed.

Summarizing, the presented decomposition method forms a first promising approach for the TCPSP with adjacent resources and may form a good basis to develop better and more efficient methods.

Adapting the presented decomposition approach to the RCPSP with adjacent resources, requires quite some effort. Due to the fixed capacities of the renewable resources in a RCPSP, it will be more difficult to anticipate the effect of the additional precedence relations implied by the GAP solution. The start and completion times of a group in the GAP solution can differ a lot from that of the final solution. On top of that, asking for long group durations in the GAP solution to have flexibility in scheduling the jobs, is conflicting with minimizing the makespan for the RCPSP with adjacent resources.

Bibliography

[1] R.A. Baeza-Yates, J.C. Culberson, and G.J.E. Rawlins. Searching in the plane. *Information and Computation*, 106(2):234–252, 1993.

[2] B.S. Baker and J.S. Schwarz. Shelf algorithms for two-dimensional packing problems. *SIAM Journal on Computing*, 12(3):508–525, 1983.

[3] J.C. Beck. Heuristics for scheduling with inventory: Dynamic focus via constraint criticality. *Journal of Scheduling*, 5(1):43–69, 2002.

[4] W.W. Bein, L. Epstein, L.L. Larmore, and J. Noga. Optimally competitive list batching. *Lecture Notes in Computer Science (Algorithm Theory - SWAT2004)*, 3111:77–89, 2004.

[5] N.L. Biggs, E.K. Lloyd, and R.J. Wilson. *Graph Theory 1736-1936*. Clarendon Press, Oxford, 1976.

[6] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

[7] D.J. Brown, B.S. Baker, and H.P. Katseff. Lower bounds for on-line two-dimensional packing algorithms. *Acta Informatica*, 18(2):207–225, 1982.

[8] P. Brucker. *Scheduling Algorithms*. Springer Verlag, fourth edition, 2004.

[9] P. Brucker, A. Drexl, R. Möhring, K. Neumann, and E. Pesch. Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operations Research*, 112:3–41, 1999.

[10] W.T. Chan, F.Y.L. Chin, D. Ye, G. Zhang, and Y. Zhang. On-line scheduling of parallel jobs on two machines. *Journal of Discrete Algorithms*, 6(1):3–10, 2008.

[11] J. Chen and C.-Y. Lee. General multiprocessor task scheduling. *Naval Research Logistics*, 46(1):57–74, 1999.

[12] M. Chrobak and C. Kenyon. Competitiveness via doubling. *SIGACT News*, 37(4):115–126, 2006.

[13] M. Chrobak, C. Kenyon, J. Noga, and N.E. Young. Incremental medians via online bidding. *Algorithmica*, 50(4):455–478, 2008.

[14] R. De Boer. *Resource-Constrained Multi-Project Management, A hierarchical decision support system.* PhD thesis, University of Twente, 1998.

[15] R.F. Deckro and J.E. Herbert. Resource constrained project crashing. *OMEGA International Journal of Management Science*, 17(1):69–79, 1989.

[16] E. Demeulemeester. Minimizing resource avaiability costs in time-limited project networks. *Management Science*, 41(10):1590–1598, 1995.

[17] E. Demeulemeester and W. Herroelen. New benchmark results for the resource-constrained project scheduling problem. *Management Science*, 43(11):1485–1492, 1997.

[18] E.L. Demeulemeester and W.S. Herroelen. *Project Scheduling: A Research Handbook.* Kluwer Academic Publishers, Boston, 2002.

[19] X. Deng, C.K. Poon, and Y. Zhang. Approximation algorithms in batch processing. *Journal of Combinatorial Optimization*, 7(3), 2003.

[20] C.W. Duin and E. Van der Sluis. On the complexity of adjacent resource scheduling. *Journal of Scheduling*, 9(1):49–62, 2006.

[21] U. Faigle, W. Kern, and G. Turàn. On the performance of online algorithms for partition problems. *Acta Cybernetica*, 9(2):107–119, 1989.

[22] D.G. Feitelson, L. Rudolph, U. Schwiegelshohn, K.C. Sevcik, and P. Wong. Theory and practice in parallel scheduling. *Lecture Notes in Computer Science*, 1291:1–34, 1997.

[23] S.P. Fekete, E. Köhler, and J. Teich. Higher-dimensional packing with order constraints. *SIAM Journal on Discrete Mathematics*, 20:1056–1078, 2006.

[24] A. Fiat and G. Woeginger, editors. *Online algorithms: The state of the art*, volume 1442 of *Lecture Notes in Computer Science*. Springer Verlag, 1998.

[25] N. Gademann and M. Schutten. Linear-programming-based heuristics for project capacity planning. *IIE Transactions*, 37(2):153–165, 2005.

[26] G. Galambos and G.J. Woeginger. On-line bin packing - a restricted survey. *Mathematical Methods of Operations Research*, 42(1):25–45, 1995.

[27] M.R. Garey and D.S. Johnson. *Computers and intractability : a guide to the theory of NP-completeness*. W.H. Freeman and Company, 1979.

[28] R.L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45(9):1563–1581, 1966.

[29] R.L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal of Applied Mathematics*, 17(2):416–429, 1969.

[30] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

[31] Y. Guan, W-Q Xiao, R.K. Cheung, and C-L Li. A multiprocessor task scheduling model for berth allocation: heuristic and worst-case analysis. *Operations Research Letters*, 30(5):343–350, 2002.

[32] T.A. Guldemond, J.L. Hurink, J.J. Paulus, and J.M.J. Schutten. Time-constrained project scheduling. Working paper 180, Beta Research School for Operations Management and Logistics, Eindhoven, The Netherlands, 2006.

[33] T.A. Guldemond, J.L. Hurink, J.J. Paulus, and J.M.J. Schutten. Time-constrained project scheduling; details of the computational tests. http://tcpsp.ewi.utwente.nl/, 2006.

[34] T.A. Guldemond, J.L. Hurink, J.J. Paulus, and J.M.J. Schutten. Time-constrained project scheduling. *Journal of Scheduling*, 11(2):137–148, 2008.

[35] S. Hartmann. Packing problems and project scheduling models: An integrative perspective. *Journal of the Operational Research Society*, 51(9):1083–1092, 2000.

[36] W. Herroelen, B. De Reyck, and E. Demeulemeester. Resource-constrained project scheduling: a survey of recent developments. *Computers and Operations Research*, 25(4):279–302, 1998.

[37] K. Hess and R. Kolisch. Efficient methods for scheduling make-to-order assemblies under resource, assembly area and part availability constraints. *International Journal of Production Research*, 38(1):207–228, 2000.

[38] J.L. Hurink, A.L. Kok, J.J. Paulus, and J.M.J. Schutten. Time-constrained project scheduling with adjacent resources. Working paper 261, Beta Research School for Operations Management and Logistics, Eindhoven, The Netherlands, 2008.

[39] J.L. Hurink and J.J. Paulus. Special cases of online parallel job scheduling. Working paper 235, Beta Research School for Operations Management and Logistics, Eindhoven, The Netherlands, 2007.

[40] J.L. Hurink and J.J. Paulus. Online algorithm for parallel job scheduling and strip packing. *Lecture Notes of Computer Science (WAOA 2007)*, 4927:67–74, 2008.

[41] J.L. Hurink and J.J. Paulus. Online scheduling of parallel jobs on two machines is 2-competitive. *Operations Research Letters*, 36(1):51–56, 2008.

[42] B. Johannes. Scheduling parallel jobs to minimize the makespan. *Journal of Scheduling*, 9(5):433–452, 2006.

[43] D.S. Johnson, A. Demers, J.D. Ullman, M.R. Garey, and R.L. Graham. Worst-case performance founds for simple one-dimensional packing algorithms. *SIAM Journal on Computing*, 3(4):299–325, 1974.

[44] T. Kis. A branch-and-cut algorithm for scheduling of projects with variable intensity activities. *Mathematical Programming*, 103(3):515–539, 2005.

[45] R. Kolisch. *Project Scheduling under Resource Constraints Efficient Heuristics for Several Problem Classes*. Physica Verlag, 1995.

[46] R. Kolisch. Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation. *European Journal of Operations Research*, 90:320–333, 1996.

[47] R. Kolisch. Integrated scheduling, assembly area- and part-assignment for large scale make-to-order assemblies. *International Journal of Production Economics*, 64:127–141, 2000.

[48] R. Kolisch and A. Drexl. Adaptive search for solving hard project scheduling problems. *Naval Research Logistics*, 43(1):23–40, 1996.

[49] R. Kolisch and S. Hartmann. Heuristic algorithms for solving the resource-constrained project scheduling problem: Classification and computational analysis. In J. Weglarz, editor, *Handbook on Recent Advances in Project Scheduling*, pages 197–212. Kluwer, 1999.

[50] R. Kolisch and R. Padman. An integrated survey of deterministic project scheduling. *Omega*, 29(3):249–272, 2001.

[51] R. Kolisch and A. Sprecher. Project scheduling library - PSPlib. http://129.187.106.231/psplib/, 1997.

[52] R. Kolisch and A. Sprecher. PSPLIB a project scheduling problem library. *European Journal of Operational Research*, 96(1):205–216, 1997.

[53] R. Kolisch, A. Sprecher, and A. Drexl. Characterization and generation of a general class of resource-constrained project scheduling problems. *Management Science*, 41(10):1693–1703, 1995.

[54] C.Y. Lee and R. Uzsoy. Minimizing makespan on a single batch processing machine with dynamic job arrivals. *International Journal on Production Research*, 37(1):219–236, 1999.

[55] C.Y. Lee, R. Uzsoy, and L.A. Martin-Vega. Efficient algorithms for scheduling semiconductor burn-in operations. *Operations Research*, 40(4):764–775, 1992.

[56] J.K. Lenstra and E. Aarts. *Local Search in Combinatorial Optimization*. Princeton University Press, 2003.

[57] C.-L. Li, X. Cai, and C.-Y. Lee. Scheduling with multiple-job-on-one-processor pattern. *IIE Transactions*, 30(5):433–445, 1998.

[58] R.K-Y Li and R.J. Willis. Resource constrained scheduling within fixed project durations. *The journal of the operational research society*, 44(1):71–80, 1993.

[59] A. Lim. The berth planning problem. *Operations Research Letters*, 22(2-3):105–110, 1998.

[60] R.H. Möhring. Minimizing costs of resource requirements in project networks subject to a fixed completion time. *Operations Research*, 32(1):89–120, 1984.

[61] K. Neumann and C. Schwindt. Project scheduling with inventory constraints. *Mathematical Methods of Operations Research*, 56(3):513–533, 2002.

[62] K. Neumann, C. Schwindt, and J. Zimmermann. *Project Scheduling with Time Windows and Scarce Resources*, volume 508 of *Lecture notes in economics and mathematical systems*. Springer, 2002.

[63] Q. Nong, J. Yuan, R. Fu, L. Lin, and J. Tian. The single-machine parallel-batch on-line scheduling problem with family jobs to minimize makespan. *International Journal of Production Economics*, 111(2):435–440, 2008.

[64] M. Palpant, C. Artigues, and P. Michelon. LSSPER: Solving the resource-constrained project scheduling problem with large neighbourhood search. *Annals of Operations Research*, 131(1):237–257, 2004.

[65] J.J. Paulus, D. Ye, and G. Zhang. Optimal online-list batch scheduling. Working paper 260, Beta Research School for Operations Management and Logistics, Eindhoven, The Netherlands, 2008.

[66] M. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Prentice Hall, second edition, 2002.

[67] M. Pinedo. *Planning and Scheduling in Manufacturing and Services*. Operations Research and Financial Engineering. Springer, 2005.

[68] C.K. Poon and W. Yu. A flexible on-line scheduling algorithm for batch machine with infinite capacity. *Annals of Operations Research*, 133(1):175–181, 2005.

[69] C.K. Poon and W. Yu. On-line scheduling algorithms for a batch machine with finite capacity. *Journal of Combinatorial Optimization*, 9(2):167–186, 2005.

[70] K. Pruhs, J. Sgall, and E. Torng. Online scheduling. In Joseph Y-T. Leung, editor, *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, chapter 15, pages 15–1–15–41. CRC Press, 2004.

[71] F. Ridouard, P. Richard, and P. Martineau. On-line scheduling on a batch processing machine with unbounded batch size to minimize the makespan. *European Journal of Operational Research*, 189(3):1327–1342, 2008.

[72] J.F. Rudin III. *Improved Bound for the Online Scheduling Problem*. PhD thesis, The University of Texas at Dallas, 2001.

[73] M.W. Schäffter. Scheduling with forbidden sets. *Discrete Applied Mathematics*, 72(1-2):155–166, 1997.

[74] A. Schrijver. *Combinatorial Optimization - Polyhedra and Efficiency*. Springer Verlag, 2003.

[75] J. Sgall. On-line scheduling of parallel jobs. *Lecture Notes in Computer Science*, 841:159–176, 1994.

[76] D.D. Sleator and R.E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

[77] C. Steiger, H. Walder, and M. Platzner. Operating systems for reconfigurable embedded platforms: Online scheduling of real-time tasks. *IEEE Transactions on Computers*, 53(11):1393–1407, 2004.

[78] J. Turek, J.L. Wolf, and P.S. Yu. Approximate algorithms for scheduling parallelizable tasks. *Proceedings of the fourth annual ACM symposium on Parallel algorithms and architectures*, pages 323–332, 1992.

[79] J. Weglarz. *Project Scheduling: Recent Models, Algorihms and Applications*. Kluwer, Boston, 1999.

[80] D. Ye, X. Han, and G. Zhang. A note on online strip packing. *Journal of Combinatorial Optimization (to appear), DOI:10.1007/s10878-007-9125-x*, 2008.

[81] D. Ye and G. Zhang. On-line scheduling of parallel jobs. *Lecture Notes in Computer Science (SIROCCO 2004)*, 3104:279–290, 2004.

[82] D. Ye and G. Zhang. On-line scheduling of parallel jobs in a list. *Journal of Scheduling*, 10(6):407–413, 2007.

[83] G. Zhang, X. Cai, and C. Wong. Some results on resource constrained scheduling. *IIE Transactions*, 36(1):1–9, 2004.

[84] G. Zhang, X. Cai, and C.K. Wong. On-line algorithms for minimizing makespan on batch processing machines. *Naval Research Logistics*, 48(3):241–258, 2001.

Samenvatting

Het centrale thema van dit proefschrift, *Online Scheduling & Project Scheduling*, is scheduling, wat vertaald naar het Nederlands planning betekent. Binnen de toegepaste wiskunde houdt het veld van het operationele onderzoek zich onder andere bezig met planningsvraagstukken: Wat is de beste volgorde waarin de producten geproduceerd kunnen worden? Welke machines kunnen het best gebruikt worden voor verschillende processen? Deze vragen moeten beantwoord worden om een organisatie soepel en efficiënt te laten lopen. De schaarse productiemiddelen moeten zodanig ingezet worden dat de doelstellingen van de organisatie behaald worden.

Planning, het toewijzen van taken in de tijd, behelst twee dingen. Ten eerste kunnen taken op verschillende manieren uitgevoerd worden. Er moet bijvoorbeeld een keuze gemaakt worden welke machine of werknemer de betreffende taak uitvoert. Ten tweede is er de volgorde van uitvoeren. De productie van product A vindt plaats voor de productie van product B, of andersom. Hieruit blijkt al dat er veel mogelijkheden zijn. De gewenste oplossing van een planningsprobleem moet gekozen worden uit een zeer groot aantal mogelijke oplossingen. Elke potentiële oplossing is te waarderen door bijvoorbeeld de kosten in deze oplossing te meten. Echter, niet alle mogelijke oplossingen zijn toegestane oplossingen, want vanuit de probleemstelling worden er mogelijk beperkende voorwaarden opgelegd. Door bijvoorbeeld contractuele verplichtingen moet product A morgen al geleverd worden, en moet dus voor die tijd geproduceerd zijn. Een planning die hier niet aan voldoet is niet een toegestane oplossing. Van het grote aantal mogelijke oplossingen zoeken we een optimale oplossing. Een optimale oplossing is een planning met de laagst haalbare kosten die tevens aan alle voorwaarden voldoet.

Op de bovenstaande manier zoeken naar een optimale oplossing uit een groot aantal alternatieven, dat een kostenfunctie minimaliseert en aan de voorwaarden

voldoet, noemen we combinatorische optimalisatie. De theorie leert ons dat alle mogelijkheden één voor één te bekijken voor de meeste problemen niet tot de mogelijkheden behoord. Een ogenschijnlijk eenvoudig planningsprobleem heeft al snel zoveel mogelijke oplossingen dat een computer jaren nodig heeft om alle mogelijke oplossingen te beschouwen.

Het is aan de wiskundige om een snelle methode te ontwikkelen die tot een optimale oplossing leidt. Wanneer dit bewijsbaar niet mogelijk blijkt te zijn, kunnen alternatieve aanpakken gekozen worden. Approximatieve schema's geven oplossingen met een garantie op de kwaliteit; de gevonden oplossing heeft bijvoorbeeld gegarandeerd niet meer dan 10% meer kosten dan een optimale oplossing. Lokale zoekmethoden geven een oplossing die niet met een kleine verandering te verbeteren is. Een heuristiek geeft snel een oplossing door een vast ingenieus stappenschema te volgen in de hoop oplossingen van hoge kwaliteit te leveren, echter zonder garantie.

In het eerste deel van dit proefschrift worden online problemen behandeld. Een probleem noemen we online als de informatie over het op te lossen probleem beetje bij beetje bekend wordt bij degene die het op moet lossen. In het online model zoals behandeld in dit proefschrift, komen de in te plannen taken één voor één bij de planner binnen. Zodra de taak is ingepland komt de volgende taak binnen. Denk hierbij bijvoorbeeld aan een reserveringssysteem. Omdat de planner de toekomstige taken niet kent en eenmaal ingeplande taken niet kan herplannen, is het moeilijker een planning van goede kwaliteit te maken. In Hoofdstukken 2 en 3 worden algoritmes ontworpen die een garantie op de kwaliteit van de oplossing leveren. De kwaliteit van de geconstrueerde oplossing wordt gemeten met het competitieve ratio, dat is het ratio tussen de kosten van de geconstrueerde oplossing en de kosten van een optimale oplossing als alle taken vooraf bekend waren geweest. Tevens geven we in deze hoofdstukken ondergrenzen voor de competitieve ratio's; we laten zien dat er geen algoritme bestaat dat een competitief ratio lager dan een bepaalde waarde heeft.

Voor een aantal planningsproblemen geeft dit proefschrift verbeterde online algoritmes, dat wil zeggen online algoritmes met lagere competitieve ratio's dan de al bekende algoritmes, en verbeterde ondergrenzen voor deze ratio's. In een aantal gevallen levert dit een optimaal algoritme op, dat wil zeggen dat er bewijsbaar geen algoritme bestaat die een lager competitief ratio heeft.

Hoofdstuk 2 behandelt online planningsproblemen waar een taak meerdere machines gelijktijdig nodig heeft. Een applicatie hiervan is te vinden in computers met parallelle procesoren, waar een berekening wordt verdeeld over een aantal van de procesoren. Als hoofdresultaten zijn te noemen: een 6.6623-competitief algoritme ongeacht het aantal machines, en een onder grens van 2 op het competitieve ratio wanneer er twee procesoren zijn.

Hoofdstuk 3 gaat over online batch scheduling. Het model bevat één machine die een aantal taken gelijktijdig kan uitvoeren. Zo'n groep van taken noemen we een batch. De taken in een batch zijn echter pas klaar wanneer alle taken in de batch klaar zijn. Een toepassing wordt gevonden in ovens voor printplaten. De

printplaten moeten voor een gegeven minimum tijd in een oven en de oven kan een aantal printplaten gelijktijdig verwarmen. Het is echter schadelijk als de oven gedurende het verwarmingsproces geopend wordt. De printplaten die in één batch zitten gaan gelijktijdig in en gelijktijdig uit de oven. In dit hoofdstuk geven we voor elke mogelijke capaciteit van de batch een online algoritme en bewijzen dat deze optimaal is.

Het tweede deel van dit proefschrift richt zich meer op praktische planningsproblemen. Deze zogenaamde project-planningsproblemen bevatten zoveel mogelijk aspecten van de realiteit. In dit deel ontwerpen we vernieuwende heuristieken. Met rekenexperimenten tonen we aan dat de heuristiek in veel gevallen een goede oplossing geeft.

In Hoofdstuk 4 is een heuristiek ontworpen voor planningsproblemen waarin taken onderworpen zijn aan strenge tijdslimieten. Door op de juiste momenten extra personeel in te zetten en of in overuren te werken, kunnen de kosten laag worden gehouden. Het vernieuwende idee ligt in de manier van het opbouwen van de planningen. We beginnen met elke taak voor een kleiner deel, zeg 75%, in te plannen en gaan dit percentage langzaam opvoeren tot alle taken geheel ingepland zijn. De rekenresultaten laten zien dat deze aanpak waardevol is.

In Hoofdstuk 5 is een decompositie ontwikkeld die het mogelijk maakt om zogenaamde aaneengesloten middelen op te nemen in de oplossingsmethodes. Een voorbeeld van een aaneengesloten middel is een scheepsdok. Wanneer een schip van 10 meter lengte in een dok gezet wordt moet er een aaneengesloten deel van 10 meter beschikbaar zijn. De 'aaneengesloten zijn' voorwaarde blijkt een grote impact te hebben op de moeilijkheid van het planningsprobleem. In de decompositie wordt eerst een plaatsing van de taken op de aaneengesloten middelen gemaakt en vervolgens wordt gekeken naar de andere productiemiddelen. We laten zien dat een goede plaatsing van groot belang is.

Curriculum vitae

Jacob Jan Paulus was born on November 13, 1980, in Opsterland, the Netherlands. In 1999 he received his VWO diploma from C.S.G. Liudger in Drachten. From 1999 to 2005 he studied Applied Mathematics at the University of Twente, the Netherlands. His Master's thesis was on the subject of online matching and performed under the supervision of dr. W. Kern at the Discrete Mathematics and Mathematical Programming group. As part of his study he obtained a minor in Management and Governance and performed his practical training at the University of Auckland, New Zealand, on the subject of optimizing hydro-electric power stations. During his study he was active member and board member of different student societies.

In May 2005 Jacob Jan Paulus started as a Ph.D. student at the Discrete Mathematics and Mathematical Programming group at the University of Twente under the supervision of dr. J.L. Hurink. The project was funded by the Dutch BSIK/BRICKS project, theme Intelligent Systems 3: Decision Support Systems for Logistic Networks and Supply Chain Optimization. During his Ph.D. period, he was course instructor for Optimization Modeling and co-supervisor of various bachelor and master projects. As a Ph.D. student he was a member of EIMDA (Euler Institute for Discrete Mathematics and its Applications), BETA (Research School for Operations Management and Logistics), and the LNMB (Dutch Network on the Mathematics of Operations Research). From the latter two he obtained Ph.D. diplomas for successfully completing the required Ph.D. level courses. In 2006 he was the Ph.D. representative in the board of the LNMB. In spring 2008 he was a visiting researcher at Zhejiang University, Hangzhou, China.

Notes