Complexity and approximation in routing and scheduling

Complexity and Approximation in Routing and Scheduling

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Technische Universiteit Eindhoven, op gezag van de
Rector Magnificus, prof.dr. R.A. van Santen, voor een
commissie aangewezen door het College voor
Promoties in het openbaar te verdedigen
op maandag 24 mei 2004 om 16.00 uur

door

Renatus Augustinus Sitters

geboren te Alkmaar

Dit proefschrift is goedgekeurd door de promotoren:

prof.dr. J.K. Lenstra
en
prof.dr. J. Sgall

Copromotor:
dr. L. Stougie

CIP-DATA LIBRARY TECHNISCHE UNIVERSITEIT EINDHOVEN

Sitters, Renatus A.

Complexity and approximation in routing and scheduling /
by Renatus A. Sitters. - Eindhoven : Technische Universiteit Eindhoven, 2004.
Proefschrift. - ISBN 90-386-0882-9
NUR 919
Subject headings : combinatorial optimisation / computational complexity /
scheduling / routing / algorithms
2000 Mathematics Subject Classification : 68Q17, 68Q25, 68W25, 90B35, 68M20

Acknowledgments

I thank Leen Stougie who has been an enormous help during my PhD research. His enthusiasm motivated me and people around me which created a great atmosphere for research. He advised me in many ways and most of this research was done with him. It am very happy to have Leen as advisor and supervisor.

I thank Jan Karel Lenstra for his advise, and comments on my writings.

I thank everybody that I worked together with during my research. In particular I thank Willem de Paepe, Maarten Lipmann, Xiwen Lu, Tjark Vredeveld, and Nicole Megow. Together with Leen, we spent much time on research in Eindhoven, but also Berlin and during many conferences in Europe. I enjoyed it a lot. I am also grateful to have worked with Jiri Sgall, Stefano Leonardi, Cor Hurkens, Judith Keijsper and Rudi Pendavingh. I thank Stefano and Alberto Marchetti Spaccamela for inviting me to Rome. I am grateful for the financial support I received from AMORE and DONET which gave me the opportunity to visit many conferences and meet researchers abroad.

Finally, I thank my girlfriend Corina, and my family and friends for their love and support.

René, April 2004.

Contents

Chapter 1

Introduction

1.1 Combinatorial optimization

Imagine you are the driver of a school bus. Day after day you bring the same group of thirty children from their homes to school and back again. In the morning and afternoon the trip starts and ends at the school where the school-bus remains overnight. Through your experience you found a route that seems quite efficient, but at some day you decide to look for the best possible tour. You decide that best possible means a tour of minimum length. You will find out that this is not so easy. In fact, it is very unlikely that you will solve this problem without the use of modern software for solving *combinatorial optimization problems*.

Definition 1 *An optimization problem* Π *is a three-tuple* $(\mathcal{I}, \mathcal{S}, f)$ *where*

1 $\mathcal{I}$ *is the set of problem instances,*

2 *every instance* $I \in \mathcal{I}$ *has a set of solutions* $\mathcal{S}(I)$*,*

3 *every solution* $x \in \mathcal{S}(I)$ *has a value* $f(x, I) \in \mathbb{R}$*.*

The objective is to determine for a given instance I *if there exists a solution, and if so, either to find a solution that has smallest value amongst all solutions for* I*, in which case we call* Π *a* minimization problem*, or to find a solution that has largest value amongst all solutions for* I*, in which case we call* Π *a* maximization problem.

In this chapter we assume that an optimization problem is a minimization problem, since we only consider such problems in this thesis. Optimization problems

in which the solution set is finite for any instance are called *combinatorial* optimization problems. Since we only consider optimization problems with a finite solution set, we will often omit the word combinatorial or simply write *problem*.

Notice the distinction we make between a problem and an instance of a problem. In an instance we actually have the information from which we can obtain an optimal solution; a problem is a set of instances, usually implicitly defined. The explicit problem of the bus driver is an instance of the well-studied *traveling salesman problem* (TSP). Instances of TSP are specified by a set of points with their pairwise distances, where the set $\mathcal{S}$ is the set of all possible tours through all points, and the function f adds to each tour its length.

In this thesis we consider two types of optimization problems: *machine scheduling problems* and *routing problems*.

1.2 Machine scheduling problems

Machine scheduling concerns the allocation of activities to resources over time, so as to optimize some objective function. Applications are found in areas like production planning, computer control, personnel scheduling, and maintenance scheduling. In production planning we can think of machines being the resources and operations that have to be performed on the machines being the activities. In the sequel we refer to the resources as *machines* and to the activities as *jobs*.

Graham et al. [45] classified the frequently used machine scheduling problems by distinguishing three problem characteristics: the *machine environment, the job characteristics*, and the *objective function*. We mention some basic features.

The simplest machine environment is the *single machine* in which each of n jobs J_j $(j = 1 \dots n)$ has to be processed during p_j time units on the machine. We assume that a machine can work on only one job at a time. In the *parallel* machine environment the processing time of a job is reciprocal to the speed of the machine, and any job can be processed by at most one machine at a time. *Identical parallel machines* operate all at the same constant speed. *Uniform parallel machines* have their own constant speed and *unrelated parallel machines* process at a job-dependent speed.

Examples of job characteristics are release dates, deadlines, and the possibility of allowing *preemption*. If preemption is allowed, then an operation may be interrupted and resumed at the same moment on a different machine or at a later moment on any machine. If preemption is not allowed, then any job must be processed on one machine without interruption.

A schedule is an allocation of the jobs to time intervals on the machines; it is feasible if all requirements imposed by the machine environment and the job characteristics are met. The objective is usually to minimize a non-decreasing function of the *completion times* $C_1, \dots, C_n$ of the jobs, e.g. the makespan

($\max_j C_j$), or the total weighted completion time ($\sum_{j=1}^{n} w_j C_j$), where each job has a given weight w_j indicating its importance.

1.3 Routing problems

Routing problems concern the design of routes in a metric space. In this thesis we only consider the routing of servers through a metric space so as to serve requests. The traveling salesman problem is a typical example of such a routing problem: Each request is a point in the metric space, and a request is served when the server's position matches the request point. The job characteristics that we mentioned for machine scheduling also apply to routing problems. For example, a request might be preempted if it has a processing time, i.e., the server must stay in the request point for a given time in order to serve the request.

Many of the popular objective functions used in scheduling problems are also common for routing problems. For example, the objective in the *traveling repairman problem* (TRP) is to minimize the sum of the (weighted) completion times. Instances of this problem are similar to TSP instances with the addition that one of the given points is labelled as the *origin*. The completion time of a request is the length of the partial tour from the origin to the request point. If the bus driver uses the TRP objective for the design of the afternoon ride (when he drops off the children), then the optimal tour minimizes the total travel time of the children. We see that the repairman is more client oriented than the salesman, who minimizes his own travel time.

There is no true distinction between scheduling and routing problems. For example, a routing problem with processing times and a single point as its metric space is basically a machine scheduling problem.

1.4 Complexity theory

We can solve the problem of the bus driver by going through all possible tours and take the one of minimum length. In general, an optimal solution for an instance of the traveling salesman problem with n points can be found by this *complete enumeration* of all possible solutions. A step-by-step procedure for a combinatorial optimization problem, which either outputs some solution or outputs that it cannot find a solution, is called an *algorithm*. In this thesis we only consider *deterministic* algorithms. Such algorithms always return the same answer for any fixed instance. In *randomized* algorithms, the steps are defined by the instance and by the value of random variables. Such algorithms may give different answers for multiple runs on a fixed instance.

Definition 2 *An algorithm* solves *a problem* Π *if for any instance* I *with a non-empty solution set* $\mathcal{S}(I)$*, the output is an optimal solution of* I*. An algorithm that solves a problem* Π *is called an* exact *algorithm for* Π*.*

We can easily program a computer to run the complete enumeration algorithm for the traveling salesman problem. However, an instance with only 30 points, such as the bus-driver's problem, already has so many possible solutions that we would not live long enough to see the answer! It seems fair to say that complete enumeration does not really solve the TSP problem. Informally, we call an algorithm *efficient* if it is guaranteed to run in an acceptable amount of time, for any instance of the problem. A generally used standard to formalize efficiency of algorithms is that of *polynomial boundedness*.

Definition 3 *We say that an algorithm is* efficient *if for any instance* I *the number of computational steps required to solve* I *is bounded by a polynomial in the size* $|I|$ *of the instance.*

The *size* $|I|$ is the number of characters used to represent an instance. For example, instances of TSP can be written as a sequence of $n(n-1)/2$ numbers, representing the pairwise distances. Each number is represented by a sequence of characters (digits). The way in which instances are represented is part of the problem definition.

A *decision promlem* is given by a set of instances, where each insctance either has the value 'yes' or the value 'no'. The problem is to decide if the given instance is a yes-instance. For example, given $2n$ integers, can we partition them in two sets with equal sum? Given a minimization problem Π, we can define a decision problem D_Π with parameter b that asks for each instance I of Π and value of b, if it has a solution with value at most b. An algorithm solves the decision problem if it answers this question correctly for any instance (I, b) of the decision problem. If we efficiently solve a problem Π, then we efficiently solve the associated decision problem. Conversely, an algorithm that solves the decision problem efficiently can be used, under a mild restriction on the objective function, to find the optimal value of an instance efficiently.

We denote by P the class of decision problems that can be solved by an efficient algorithm. For many natural problems, such as TSP, it is conjectured but not yet proven that their decision problem does not belong to P. On the other hand, if an instance of TSP has a solution with value at most b, then we can easily verify this if such a solution is given to us. We denote by NP the class of decision problems for which we can efficiently verify any yes answers of the decision problem. For a formal definition of the classes P and NP we

refer to [37]. The class P is contained in NP. A famous open question in mathematics is whether or not $P = NP$.

We say that a problem $D_1 \in NP$ *polynomially transforms* to a problem $D_2 \in NP$ if we have a function $\mathcal{F}$ that maps any instance I_1 of D_1 to an instance I_2 of D_2 such that $\mathcal{F}$ can be computed in polynomial time, and I_1 is a yes instance of D_1 if and only if I_2 is a yes instance of D_2. More generally, we say that D_1 *polynomially reduces* to D_2 if we can solve D_1 efficiently if solving any instance of D_2 is counted as a single step.

Definition 4 *A decision problem D is NP-complete if all problems in NP polynomially transform to D.*

It is unlikely that we can efficiently solve any NP-complete problem, since this would imply that we can efficiently solve all problems in NP. Since polynomial transformations are transitive we can prove NP-completeness of a decision problem D_1 by showing that:

(i) D_1 is a member of NP;
(ii) there is an NP-complete problem D_2 that polynomially transforms to D_1.

Cook [27] showed that the *satisfiability* problem is NP-complete by giving an explicit transformation to SAT for any problem in NP. See [37] for a definition of the satisfiability problem. Since Cook's proof, many decision problems have been classified as NP-complete using the two conditions above [37].

We say that an optimization problem is *NP-hard* if any exact algorithm for this problem will solve an NP-complete problem at the loss of at most a polynomial factor in running time. Hence, an optimization problem is NP-hard if its decision problem is NP-complete.

Many optimization problems contain numbers, representing distances, capacities or weights. A decision problem is called *strongly* NP-complete, or NP-complete *in the strong sense*, if it is NP-complete even if we restrict ourselves to instances in which all numbers are polynomially bounded in the input size. We use the term *ordinarily* NP-complete if we can show NP-completeness of a problem but we do not know if it is also strongly NP-complete. Conversely, we say that a problem is *pseudo-polynomially solvable* if we can solve it in a time polynomial in the input size and the value of the largest integer.

1.5 Approximation algorithms

Algorithms that are not guaranteed to find an optimal solution could still be considered 'good' if the output value is never far from the optimal value. Given

an instance I for minimization problem Π we denote the optimal value of instance I by $\textsc{Opt}(I)$. If I has no feasible solution, then we set $\textsc{Opt}(I) = \infty$. We denote the value of the solution given by A on input I by $A(I)$. If no solution is returned, then we set $A(I) = \infty$.

Definition 5 *An algorithm A is an* α-approximation algorithm *for a minimization problem Π if for any instance I with $\textsc{Opt}(I) < \infty$, the value of the solution returned by A is at most α times the optimal value for instance I.*

The value α is called the *performance guarantee* of algorithm A. It is at least 1 for a minimization problem and solves a problem if and only if $\alpha = 1$.

A family of algorithm $\{A_\epsilon\}_{\epsilon>1}$ is called a *polynomial time approximation scheme* (PTAS) for a minimization problem Π if A_ϵ is an efficient ϵ-approximation algorithm for every fixed $\epsilon > 1$.

We briefly mention how we can classify optimization problems by their approximability in a similar way as we did above for their solvability. For an extensive overview on complexity classes and approximation we refer to the pioneering paper by Papadimitriou and Yannakakis [67], and the books by Vazirani [85] and Ausiello et al. [9]. The class APX is formed by all optimization problems that have an efficient α-approximation algorithm for some constant α. The APX-complete problems are the hardest problems within APX with respect to so called *approximation preserving reductions*. The class of APX-complete problems is non-empty and an APX-complete problem does not have a PTAS unless $P{=}NP$ [8]. Hence, assuming $P \neq NP$, we can show the inapproximability of an optimization problem by giving an approximation preserving reduction from an APX-complete problem.

Papadimitriou and Yannakakis [67] defined the class $Max\text{-}SNP$ to find evidence of non-approximability. We omit any definition of this class and only mention that there is no $Max\text{-}SNP$-hard problem with a PTAS [8], unless $P{=}NP$.

1.6 On-line algorithms

In many practical applications optimization is a continuous process. Think about routing taxis in a city or assigning airplanes to gates. We simply cannot afford to wait until we have all information (if it is finite at all) before making choices about the solution. Many classical *off-line* optimization problems have a natural on-line variant. In the *on-line* traveling repairman problem, as described in [55], the instances are similar to the off-line traveling repairman problem with release dates. We require that at any moment t, the algorithm is ignorant of the requests released after time t. In other words, the tour followed by the server until time t is a function of the requests released until time t only.

We distinguish two frequently used models for on-line problems: problems in which the requests (or jobs) arrive over time, as in on-line TRP, and problems in which requests (or jobs) arrive one by one. In the first case the algorithm is free to choose among the available requests or it might even remain idle for some time. The latter case does not involve time. Any request is revealed as soon as the preceding request has been served. Any step made by the algorithm at the release of a request is a function of the requests released until then. There are many other ways to model optimization problems as on-line problems.

The performance of an on-line algorithm is often measured through its *competitive ratio*.

Definition 6 *An algorithm A is c-competitive for a minimization problem Π if there exists a constant γ such that for all inputs I*

$$A(I) \leq c \cdot \textsc{Opt}(I) + \gamma.$$

If γ is at most zero, then we say that A is strictly c-competitive.

An algorithm is called *competitive* if it is c-competitive for some constant c. The *competitive ratio* of an algorithm A is the infimum value c for which A is c-competitive. The competitive ratio of an on-line minimization problem Π is the infimum over all values c for which there exists a c-competitive algorithm.

A strictly c-competitive algorithm is a c-approximation algorithm. In practice we usually seek efficient on-line algorithms, i.e. algorithms requiring polynomial time. However, in on-line optimization theory we usually make no requirements or assumptions about the running time of algorithms. Most on-line problems do not have 1-competitive algorithms. The usual goal in on-line optimization is to find a c-competitive algorithm for some on-line optimization problem, and in addition to show that no on-line algorithm can be better than c-competitive.

An on-line problem is often described as a game between an on-line player (the algorithm) and an evil *adversary*, who tries to make the relative performance of the on-line player as bad as possible. The adversary knows the algorithm of the on-line player and chooses the worst instance for this algorithm.

1.7 Outline of the thesis

The research is focused on the computational complexity and competitive analysis of some elementary machine scheduling and routing problems. Here we briefly mention the main results.

In Chapter 2 we show that minimizing total completion time on unrelated parallel machines is NP-hard if preemption is allowed. The result is surprising

since the non-preemptive case can be solved in polynomial time. There are very few scheduling problems for which allowing preemption makes the problem harder.

In Chapter 3 we consider on-line algorithms for minimizing total (weighted) completion time on identical parallel machines. We generalize existing results and improve upon the best known competitive ratio for a preemptive single-machine problem.

Routing problems are the subject of Chapters 4 and 5. We prove NP-hardness of the traveling repairman problem on the tree metric. Many routing problems are much easier for trees than for a general metric space. Nevertheless, the tree metric is a useful model for many application in network routing. The traveling repairman problem has been well-studied, but most research is done on approximation algorithms for various metric spaces, including the tree metric. Our negative result implies that an efficient exact algorithm for this problem is highly unlikely to exist.

In Chapter 5 we consider the competitiveness of *metrical service systems*. This model contains many well-studied on-line routing problems as a special case, e.g. the *paging problem* and the *k-server problem*. We show a non-trivial sufficient condition for competitiveness of any metrical service system and provide an implicit algorithm. Although our algorithm attains huge competitive ratios in general, it is the first provably competitive algorithm for an important on-line routing problem: the *general two-server problem*. Our structural theorem gives new insight into the competitive analysis of metrical service systems.

Chapter 2

Complexity of preemptive minsum scheduling on unrelated machines

2.1 Introduction

Suppose that m machines M_i $(i = 1, \ldots, m)$ have to process n jobs J_j $(j = 1, \ldots, n)$. Each job can be processed by any of the machines but by only one at time. Each machine can process at most one job at a time. The time it takes to process job J_j completely on machine M_i is given by a positive integer p_{ij}. Preemption is allowed. We consider two optimality criteria: minimizing the sum of completion times $\sum_{j=1}^{n} C_j$, and minimizing the sum of unit lateness penalties $\sum_{j=1}^{n} U_j$. In the latter case each job J_j has a given due date d_j, that is, the moment in time by which it should be completed. We say that a job is *late* if $d_j < C_j$ and the penalty U_j is 1 if job J_j is late, and 0 otherwise. $\sum_{j=1}^{n} C_j$ is also called the total completion time. $\sum_{j=1}^{n} U_j$ is the number of late jobs.

With the two optimality criteria we have defined two scheduling problems. In the notation introduced by Graham et al. [45] they are $R|pmtn|\sum U_j$ and $R|pmtn|\sum C_j$. The 'R' indicates that we have unrelated parallel machines, i.e., no relation between the mn processing times p_{ij} is presumed. The number of machines m is defined as part of the problem instance. If the number of machines m is fixed, the notation 'Rm' is used. The abbreviation '$pmtn$' indicates that preemption is allowed. The third field indicates which optimality criterion is used. We will use this notation in this chapter as a name for the problems.

Lawler [57] proved that the problem of preemptively minimizing the number of late jobs on identical parallel machines, is ordinarily NP-hard. Hence,

$R|pmtn|\sum U_j$ is ordinarily NP-hard. We show that $R|pmtn|\sum U_j$ is NP-hard in the strong sense.

If no preemption is allowed ($R||\sum C_j$), an optimal schedule can be found in polynomial time by solving a weighted bipartite matching problem (Horn [49], Bruno et al. [17]). Consider the jobs that are to be processed on some machine M_i, and for simplicity assume that these are $J_l, J_{l-1} \ldots, J_1$. The total completion time of these jobs, if scheduled in this order, is $lp_{l1}+(l-1)p_{i,l-1}+\ldots+p_{i1}$. In general, the contribution of a job J_j is kp_{ij} if it is scheduled kth last on machine M_i. Define the weighted bipartite matching problem with the set of machines as one set of the partition and the other set is $\{(i,k) \mid 1 \leq i \leq m, 1 \leq k \leq n\}$, where the pair (i,k) indicates the kth last job position on machine M_i. The cost of matching job J_j to the pair (i,k) is kp_{ij}. Any optimal schedule corresponds to a minimum weighted matching and vice versa. The weighted matching problem can be solved in $O(n^3)$ time.

In the case of uniform parallel machines and no release date constraints, Gonzalez [44] shows that an optimal preemptive schedule can be found in polynomial time. We say that the machines are uniform if $p_{ij} = p_j/s_i$ for given *processing requirement* p_j of job J_j, and speed s_i of machine M_i. On the other hand, preemptive scheduling to minimize total completion time under release date constraints on two identical machines is NP-hard. We show that the problem $R|pmtn|\sum C_j$ is NP-hard in the strong sense.

$R\|\|\sum C_j$	$O(n^3)$	[17],[49]
$R\|pmtn\|\sum C_j$	NP-hard*	[◇]
$R\|\|\sum U_j$	NP-hard*	[36]
$R\|pmtn\|\sum U_j$	NP-hard*	[◇],[79]
$R\|p_{ij} \in \{p_j, \infty\}\|\sum C_j$	$O(n^3)$	[17],[49]
$R\|pmtn, p_{ij} \in \{p_j, \infty\}\|\sum C_j$	$O(n^3)$	[◇],[79]
$R\|p_{ij} \in \{p_j, \infty\}\|\sum U_j$	NP-hard*	[36]
$R\|pmtn, p_{ij} \in \{p_j, \infty\}\|\sum U_j$	NP-hard	[57]

[◇] this chapter
* NP-hard in the strong sense.

Table 2.1: Complexity status of some unrelated machine problems.

For almost all scheduling problems the preemptive version of the problem is not harder to solve than the non-preemptive version. For at least two scheduling problems this empirical law does not hold true. Brucker, Kravchenko, and Sotskov [16] showed that the preemptive *job shop* scheduling problem with two machines and three jobs ($J2|n=3, pmtn|C_{\max}$) is ordinarily NP-hard, whereas Kravchenko and Sotskov [54] showed that the non-preemptive version can be

solved in $O(r^4)$ time, where r is the maximum number of operations of a job. The other exception is that of finding an optimal preemptive schedule for equal length jobs on identical parallel machines. The optimality criterion is the sum of weighted lateness penalties. This problem was proven to be ordinarily NP-hard by Brucker and Kravchenko [15]. In the same paper they give a $O(n \log n)$ time algorithm for the non-preemptive version. Recently they even proved strong NP-hardness for the preemptive problem [14]. $R|pmtn|\sum C_j$ is the third problem type on this short list.

A special case of the unrelated machine model is the model in which each job J_j has a fixed processing time p_j but the job can only be processed on a job-specific subset of the machines. This is modelled by defining the processing time of job J_j on machine M_i as either p_j or infinite. We denote this problem as $R|pmtn, p_{ij} \in \{p_j, \infty\}|\sum C_j$. If preemption is not allowed, then the problem can be solved efficiently [49, 17]. We show that within this model, the problem of minimizing total completion time, when preemption is allowed, can be solved in polynomial time by showing that there exists an optimal schedule that is non-preemptive.

2.2 *NP*-hardness of preemptive minsum scheduling

We prove that preemptive scheduling on unrelated machines is NP-hard in the strong sense for both the sum of unit lateness penalties objective and the total completion time objective. We will polynomially transform the 3-DIMENSIONAL MATCHING PROBLEM to the decision problems of either of the two optimization problems. The 3-DIMENSIONAL MATCHING PROBLEM was proven to be NP-hard by Karp [50].

3-DIMENSIONAL MATCHING (3DM)

Instance: Sets $U = \{u_1, \ldots, u_m\}$, $V = \{v_1, \ldots, v_m\}$, and $W = \{w_1, \ldots, w_m\}$, and a subset $S \subset U \times V \times W$ of size $n \geq m$.
Question: Does S contain a perfect matching, that is, a subset S' of cardinality m that covers every element in $U \cup V \cup W$?

As a preliminary we define, for each instance of the 3DM problem, three *basic sets of machines* and one *basic set of jobs* that we shall use in both NP-hardness proofs in this chapter.

Given an instance of 3DM we define one machine U_i for each element u_i of the set U. The set of these m machines is denoted by U as well. In the

same way we define the sets of machines V and W. We use the following notation for the set S: $S = \{(u_{\alpha_j}, v_{\beta_j}, w_{\gamma_j}) |\ j = 1, \ldots, n\}$. For each element $s_j = (u_{\alpha_j}, v_{\beta_j}, w_{\gamma_j})$ of S we define one job which we denote by J_j. The set of these n jobs is denoted by J. Their processing time is small on the three machines that correspond to the related triple, and is large on all other machines. To be specific: let $s_j = (u_{\alpha_j}, v_{\beta_j}, w_{\gamma_j})$ be an element of S, then the processing time of job J_j is $3p$ on machine U_{α_j}, $\frac{3}{2}p$ on machine V_{β_j}, and p on machine W_{γ_j}, where $p \in \mathbb{R},\ p \geq 2$. The processing time is K on any of the other $3m-3$ machines, where $K \in \mathbb{R},\ K \geq 6p$. The numbers K and p will be chosen appropriately in each of the two NP-hardness proofs.

We use the following terminology. We say that a machine is a *slow* machine for job J_j if the processing time of job J_j is K on that machine. In any other case we say that the machine is *fast* for the job. The following lemma relates the perfect 3-dimensional matching property to properties of the scheduling instance given by the sets U, W, W and J.

Lemma 1 *Let I be an instance of 3DM and let the corresponding sets U, V, W and J as defined before. If we add the restriction that none of the V-machines can process a job before time $t = 1$ and none of the W-machines can be used for processing before time $t = 2$, then the following holds for every preemptive schedule:*

(i) *$C_j \geq p + 1$ for every job $J_j \in J$,*

(ii) *if there are m jobs J_j for which $C_j < (p+1) + \frac{1}{12}$, then I contains a perfect matching.*

Proof. (i) The completion time of job J_j is minimized if it is at any time scheduled on the fastest available machine. Schedule job J_j on machine U_{α_j} from $t = 0$ to $t = 1$, on machine V_{β_j} from $t = 1$ to $t = 2$, and on machine W_{γ_j} from time $t = 2$ onwards. Scheduled in this way $C_j = p+1$. Any other schedule will give a strictly larger completion time. We call this the *optimal* schedule job J_j.

(ii) We say that the schedule of job J_j is *nearly optimal* if J_j is scheduled for more than 0.5 time units on machine U_{α_j} in the interval $[0, 1]$, for more than 0.5 time units on machine V_{β_j} in the interval $[1, 2]$, and for more than 0.5 time units on machine W_{γ_j} in the interval $[2, 3]$. If m jobs receive simultaneously a nearly optimal schedule, then a perfect matching exists. Now consider the completion time of a job J_j that is processed at any time on the fastest available machine with the restriction that machine U_{α_j} is available between time 0 an 1 for exactly $t \leq 0.5$ time units. At time 2, job J_j has been processed for a fraction $t/3p + (1-t)/K + 1/1.5p \leq 11/12p$. Hence, the completion time is at least $2 + (1 - 11/12p)p = p + 1 + 1/12$. Similarly, we can show that the completion

time of J_j is at least $p+1+1/6$ if it is scheduled for at most 0.5 time units on machine V_{β_j} in the interval $[1,2]$, or for at most 0.5 time units on machine W_{γ_j} in the interval $[2,3]$. We conclude that the completion time of a job is at least $p+1+1/12$ if its schedule is not nearly optimal. □

We use this construction with the optimal schedule of the J-jobs to prove Theorems 1 and 2. In the lemma we made the restriction that sets of machines can only be used from some point in time onwards; in the NP-hardness proofs we have to find a way to enforce this. It turns out that this is easy for the $\sum U_j$ objective and more complicated for the $\sum C_j$ objective.

2.2.1 Minimizing the number of late jobs

Formally, the decision problem of $R|pmtn|\sum U_j$ is:

Instance: A number α, a set $\{M_1, M_2, \ldots, M_m\}$ of m unrelated machines, a set $\{J_1, J_2, \ldots, J_n\}$ of n independent jobs, and a set $\{p_{ij} | i = 1 \ldots m, j = 1 \ldots n\}$ where p_{ij} is the processing time of job J_j on machine M_i.
Question: Does there exist a preemptive schedule for which $\sum_{j=1}^{n} U_j \leq \alpha$?

Theorem 1 *$R|pmtn|\sum U_j$ is strongly NP-hard.*

Proof. Let I be an instance of 3DM with the notation as defined before. The scheduling instance consists of the basic sets of machines U, V and W, and the basic set of jobs J, defined before, and additionally we define a set A of jobs as follows. For each machine of the set V we introduce one job with processing time 1 on that specific machine, and with processing time K on any of the other $3m-1$ machines. The due date is 1 for all these jobs. For each machine of W we define one job with processing time 2 on that specific machine, and with processing time K on any of the other $3m-1$ machines. The due date is 2 for all these jobs. We define the set J of jobs related to S, with their processing times as before. We define the due dates of the jobs in the set J as $p+1$. We choose the values $p = 3m+3$ and $K = 6p$.

We claim that for any schedule, the number of late jobs is less than or equal to $n-m$, if and only if a perfect matching exists. Notice that, since the number of jobs is $n+2m$, this is the same as claiming that the number of early jobs is at least $3m$ if and only if a perfect matching exists.

If there exists a perfect matching, then it is possible to schedule the jobs such that $3m$ jobs are early: Schedule the $2m$ A-jobs such that they are early (there is only one way to do this), and give the m J-jobs that correspond to the elements in the 3-dimensional matching their optimal schedule as described in the proof of Lemma 1.

Now observe that it is impossible to have more than m early J-jobs. For a J-job to be early, it must be scheduled on its fast W-machine for a time of at least $p-2$. So if there are at least $m+1$ early J-jobs, then the total processing time required on the W-machines is at least $(m+1)(p-2) = (m+1)(3m+1)$. For any m this is strictly larger than $m(3m+4)$, which is the total available processing time on the W-machines before the due date, $3m+4$.

We conclude that if at least $3m$ jobs are early, then these jobs are the $2m$ A-jobs and exactly m of the J-jobs. From Lemma 1(ii) it follows that in this case a perfect matching exists. □

It is not obvious that the decision problem is in the class NP since we have to exclude the possibility that there is a schedule with a superpolynomial number of preemptions that has a strictly smaller objective value than any schedule with a polynomial number of preemptions. Lawler and Labetoulle [58] show that for any monotone, non-decreasing objective function $f(C_1, C_2, \ldots, C_n)$, an optimal schedule can be constructed by solving a linear program, if the completion times of the jobs in an optimal schedule are given. The number of preemptions for this schedule is $O(m^2 n)$. Verifying the feasibility of a schedule with $O(m^2 n)$ preemptions requires polynomial time, whence Theorem 1 implies that the decision problem of $R|pmtn|\sum U_j$ is NP-complete.

2.2.2 Minimizing the sum of completion times

Formally, the decision problem of $R|pmtn|\sum C_j$ is:

Instance: A number α, a set $\{M_1, M_2, \ldots, M_m\}$ of m unrelated machines, a set $\{J_1, J_2, \ldots, J_n\}$ of n independent jobs, and a set $\{p_{ij} | i = 1 \ldots m, j = 1 \ldots n\}$ where p_{ij} is the processing time of job J_j on machine M_i.
Question: Does there exist a preemptive schedule for which $\sum_{j=1}^{n} C_j \leq \alpha$?

Theorem 2 *$R|pmtn|\sum C_j$ is strongly NP-hard.*

Proof. Let I be an instance of 3DM with the notation as defined before. The scheduling instance contains the basic sets of machines U, V and W, and the basic set of jobs J. Additionally we define one machine which we denote by Z. The processing time on the Z-machine is p for any job from J. The value of p is set to $p = 2$. The value of K (which was defined as the processing time on slow machines) will be specified later. We also define two additional sets A and B of jobs. For each V-machine we define M A-jobs with processing time $\frac{1}{M}$ on that specific machine and processing time K on any other machine. The value of M will be specified later. For each W-machine we define $2M$ A-jobs with processing time $\frac{1}{M}$ on that specific machine and with processing

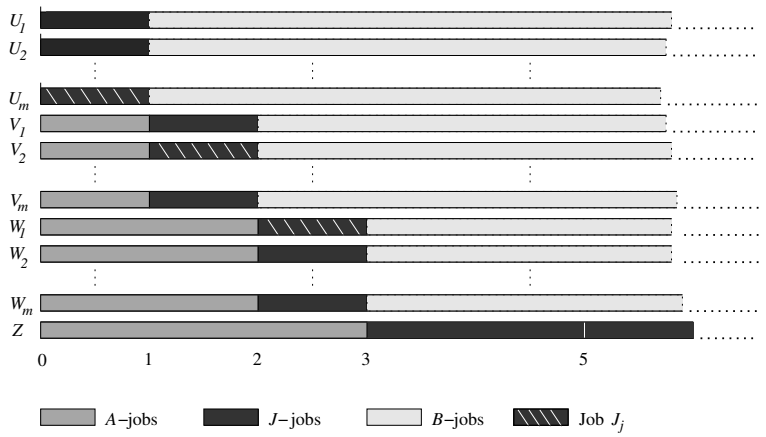

Figure 2.1: Sketch of the schedule σ_{3DM}.

time K on any other machine. For the Z-machine we define $3M$ A-jobs with processing time $\frac{1}{M}$ on the Z-machine and processing time K on any of the other $3m$ machines. This makes the total number of jobs in the set A $(3m+3)M$. The A-jobs are meant to keep the V-machines busy until time 1, the W-machines busy until time 2, and the Z-machine busy until time 3. Finally, we define the B-jobs. For each U-machine we define $\frac{1}{3}(n-m)$ B-jobs with processing time $2n+4$ on that specific machine, and processing time K on any other machine. (Without loss of generality we may assume that $\frac{1}{3}(n-m)$ is integer.) For each V-machine we define $\frac{2}{3}(n-m)$ B-jobs with processing time $2n+4$ on that specific machine and processing time K on any other machine. For each W-machine we define $(n-m)$ B-jobs with processing time $2n+4$ on that specific machine and processing time K on any other machine. The B-jobs are introduced to ensure that, in an optimal schedule, a limited subset of the J-jobs is scheduled on the U-, V-, and W-machines.

If a perfect matching exists then the jobs can be scheduled as shown in Fig. 2.1. All A-jobs are scheduled as early as possible on their fast machines. The m jobs from J that correspond to the perfect matching are scheduled as in the proof of Lemma 1. The completion time of these jobs is 3. All other J-jobs are scheduled after the A-jobs on the Z-machine. Each B-job is scheduled on its unique fast machine. The B-jobs are placed directly after the other jobs. This schedule is denoted by σ_{3DM}. The sum of completion times in σ_{3DM} is denoted by $C_{\sigma_{3DM}}$. The value of $C_{\sigma_{3DM}}$ is clearly a polynomial in m, n, M, and $1/M$. The expression is omitted here.

We write C_σ for the total completion time of any feasible schedule σ, and we write $C_\sigma(Y)$ for the sum of completion times of any subset of jobs Y in σ.

Now we show that if no perfect matching exists, then for any preemptive schedule σ the total completion time is strictly larger than $C_{\sigma_{3DM}}$. In fact we will show that $C_\sigma \geq C_{\sigma_{3DM}} + \frac{1}{48}$.

We introduce two restrictions to the scheduling problem.

- *Restriction 1:* No job may be processed on a slow machine.
- *Restriction 2:* All A-jobs have to be scheduled as in σ_{3DM}.

With these two restrictions we define three scheduling problems.

- *Problem 1:* The original problem with Restrictions 1 and 2.
- *Problem 2:* The original problem with Restriction 1.
- *Problem 3:* The original problem.

For each of the three problems we show a lower bound on the total completion time in case no perfect matching exists for the 3DM instance I. The essence of the reduction is contained in the proof for Problem l. It is intuitively clear that a lower bound for Problem 2 or 3 can be made arbitrarily close to a lower bound for Problem 1 if M and K are chosen large enough. We give explicit values for the numbers M and K and prove that these values suffice, i.e., we show that processing the A-jobs different or using slow machines will not invalidate the reduction.

Problem 1: Let σ be a feasible schedule for Problem 1. We prove that $C_\sigma \geq C_{\sigma_{3DM}} + \frac{1}{12}$ if no perfect matching exists. Since the total processing time of the jobs for which the Z-machine is fast is at most $2n + 3$, and the processing time of a B-job is $2n+4$, we may assume that the completion time of any J-job is strictly smaller than the completion time of any B-job. Otherwise we could decrease the completion time such a J-job by processing it as early as possible on the Z-machine.

Let T_{U_i} $(i = 1, \ldots, m)$ be the time that is spent on processing J-jobs on machine U_i. Define T_{V_i}, T_{W_i} $(i = 1, \ldots, m)$ and T_Z in a similar way. Notice that in schedule σ_{3DM} we have $T_{U_i} = T_{V_i} = T_{W_i} = 1$ and $T_Z = 2(n - m)$. Let $B(U_i)$ $(i = 1, \ldots, m)$ be the set of B-jobs that have machine U_i as their fast machine and define $B(V_i)$ and $B(W_i)$ in a similar way.

$$\begin{aligned}
&C_\sigma(B(U_i)) \geq C_{\sigma_{3DM}}(B(U_i)) + \tfrac{1}{3}(n-m)(T_{U_i} - 1) && (i = 1, \ldots, m),\\
&C_\sigma(B(V_i)) \geq C_{\sigma_{3DM}}(B(V_i)) + \tfrac{2}{3}(n-m)(T_{V_i} - 1) && (i = 1, \ldots, m),\\
&C_\sigma(B(W_i)) \geq C_{\sigma_{3DM}}(B(W_i)) + (n-m)(T_{U_i} - 1) && (i = 1, \ldots, m).
\end{aligned}$$

Adding the three equations above yields:

$$C_\sigma(B) \geq C_{\sigma_{3DM}}(B) + (n-m)\left(\frac{1}{3}\sum_{i=1}^{m} T_{U_i} + \frac{2}{3}\sum_{j=i}^{m} T_{V_i} + \sum_{i=1}^{m} T_{W_i} - 2m\right).$$

By the choice of the processing times of the J-jobs we have

$$T_Z = 2n - \frac{1}{3}\sum_{i=1}^{m} T_{U_i} - \frac{2}{3}\sum_{i=1}^{m} T_{V_i} - \sum_{i=1}^{m} T_{W_i}.$$

Combining the two equations above yields

$$C_\sigma(B) \geq C_{\sigma_{3DM}}(B) + (n-m)\left(2(n-m) - T_Z\right). \tag{2.1}$$

Let $C_\sigma(J)_1$ be the sum of the m smallest completion times amongst the J-jobs in the schedule σ, and let $C_\sigma(J)_2$ be the sum of the $n-m$ largest completion times among the J-jobs. In a similar way we define $C_{\sigma_{3DM}}(J)_1$ and $C_{\sigma_{3DM}}(J)_2$. Let $c_1 \leq c_2 \leq \ldots \leq c_n$ be an ordering of the completion times of the J-jobs in σ. The largest completion time c_n is at least $3 + T_Z$, and the second largest is at least $3 + T_Z - 2$, and so on. In general we have $c_j \geq 3 + T_Z - 2(n-j)$. We obtain:

$$\begin{array}{rcl} C_\sigma(J)_2 & = & \sum_{j=m+1}^{n} c_j \\ & \geq & \sum_{j=m+1}^{n} (3 + T_Z - 2(n-j)) \\ & = & \sum_{j=m+1}^{n} (3 + 2(j-m) + T_Z - 2(n-m)) \\ & = & C_{\sigma_{3DM}}(J)_2 + (T_Z - 2(n-m))(n-m). \end{array} \tag{2.2}$$

Combining inequalities (2.1) and (2.2), and using $C_{\sigma_{3DM}}(A) = C_\sigma(A)$ and $C_{\sigma_{3DM}}(J)_1 = 3m$ we obtain

$$\begin{array}{rcl} C_\sigma & = & C_\sigma(A) + C_\sigma(B) + C_\sigma(J)_1 + C_\sigma(J)_2 \\ & \geq & C_{\sigma_{3DM}}(A) + C_{\sigma_{3DM}}(B) + C_\sigma(J)_1 + C_{\sigma_{3DM}}(J)_2 \\ & = & C_{\sigma_{3DM}} - C_{\sigma_{3DM}}(J)_1 + C_\sigma(J)_1 \\ & = & C_{\sigma_{3DM}} - 3m + C_\sigma(J)_1. \end{array}$$

From Lemma 1 we have $C_\sigma(J)_1 \geq 3m + \frac{1}{12}$, implying $C_\sigma \geq C_{\sigma_{3DM}} + \frac{1}{12}$.

Problem 2: Let σ be any feasible schedule for Problem 2. We prove that $C_\sigma \geq C_{\sigma_{3DM}} + \frac{1}{24}$ for an appropriately large value of M if there does not exists a perfect matching.

Let $\delta(V_i)$ $(1 \leq i \leq m)$ be the total time between time 0 and 1 during which machine V_i does not process A-jobs. Similarly, let $\delta(W_i)$ and $\delta(Z)$ be the total time in respectively the time interval $[0,2]$ and $[0,3]$ during which machine W_i and Z do not process their A-jobs. The last A-job on machine V_1 completes earliest at time $1 + \delta(V_1)$, and the second last at time $1 + \delta(V_1) - \frac{1}{M}$, and so on. Hence, compared to the schedule σ_{3DM}, at least $\lceil \delta(V_1)M \rceil$ A-jobs on machine V_1 are delayed by $\delta(V_1)$, increasing the total completion time by at least $\delta(V_1)^2 M$. Denote $\delta = \delta(V_1) + \ldots + \delta(V_m) + \delta(W_1) + \ldots + \delta(W_m) + \delta(Z)$.

$$\begin{array}{rcl} C_\sigma(A) & \geq & C_{\sigma_{3DM}}(A) + \sum_{i=1}^{m} \delta(V_i)^2 M + \sum_{i=1}^{m} \delta(W_i)^2 M + \delta(Z)^2 M \\ & \geq & C_{\sigma_{3DM}}(A) + \delta^2 M/(2m+1). \end{array} \tag{2.3}$$

If this total time δ is used for processing a J-job, then at most a fraction $\delta/2$ of this job can be processed during this time. For B-jobs this fraction is even smaller. Now consider the problem in which all processing times of J- and B-jobs are multiplied by a factor $1-\delta/2$, slow machines may not be used, and the machines from V, W and Z may not be used until time 1, 2 and 3 respectively. From Problem 1 it follows that the sum of completion times of the J- and B-jobs in this scaled problem, and thus also in the original problem, is at least $(1-\frac{1}{2}\delta)(C^* + \frac{1}{12})$, where we write C^* for $C_{\sigma_{3DM}}(J) + C_{\sigma_{3DM}}(B)$. Together with (2.3) this gives the following inequality:

$$\begin{array}{rcl} C_\sigma & \geq & C_{\sigma_{3DM}}(A) + \delta^2 M/(2m+1) + (1-\frac{1}{2}\delta)(C^* + \frac{1}{12}) \\ & = & C_{\sigma_{3DM}} + \delta^2 M/(2m+1) - \frac{1}{2}\delta(C^* + \frac{1}{12}) + \frac{1}{12} \end{array} \tag{2.4}$$

Standard calculus tells us that (2.4) implies

$$C_\sigma \geq C_{\sigma_{3DM}} - \frac{1}{16}(C^* + \frac{1}{12})^2(2m+1)/M + \frac{1}{12} \tag{2.5}$$

The number M has not been specified yet. If we choose $M = \frac{3}{8}(C^* + \frac{1}{12})^2(2m+1)$, then $C_\sigma \geq C_{\sigma_{3DM}} + \frac{1}{24}$. Notice that M is well-defined since $C^* = C_{\sigma_{3DM}}(J) + C_{\sigma_{3DM}}(B)$ does not depend on M.

Problem 3: Let σ be any schedule for Problem 3. If a perfect matching does not exist, we prove we will prove that $C_\sigma \geq C_{\sigma_{3DM}} + 1/48$ for $K \geq 48(2m+4)N(C_{\sigma_{3DM}} + \frac{1}{48})$, where N is the total number of jobs in our scheduling instance. Suppose that some parts of jobs are scheduled on slow machines. If the total number of processing units scheduled on slow machines exceeds $(C_{\sigma_{3DM}} + \frac{1}{48})/K$, then the statement clearly holds. From σ we define a new schedule in three steps. First, remove all the work that is scheduled on slow machines. Secondly, shift the remaining schedule forward in time over a time $1/(48N)$. That is, all work is postponed by $1/(48N)$. Thirdly, reschedule the removed work on fast machines between $t = 0$ and $t = 1/(48N)$. This is possible since the total processing time of this work is at most $(2m+4)(C_{\sigma_{3DM}} + \frac{1}{48})/K \leq 1/(48N)$ if completely scheduled on fast machines. In this new schedule no job is scheduled on a slow machine, and the increase in the total sum of completion times is at most $1/48$. Using the lower bound for Problem 2 we obtain

$$C_\sigma + \frac{1}{48} \geq C_{\sigma_{3DM}} + \frac{1}{24} \quad \Rightarrow \quad C_\sigma \geq C_{\sigma_{3DM}} + \frac{1}{48}.$$

We conclude that a perfect matching exists if and only if there exist a schedule σ for which $C_\sigma \leq C_{\sigma_{3DM}} + \frac{1}{48}$. □

From Section 2.2.1 we know that the decision problem of $R|pmtn|\sum C_j$ is in NP. Hence, Theorem 2 implies that the decision problem is NP-complete.

2.3 Identical machines with job-specific availability

A special case of the unrelated machine model is the model in which each job J_j has a fixed processing time p_j but with the restriction that it can only be processed on a job-specific subset of the machines. This is modelled by setting the processing time of job J_j to p_j on machine M_i if J_j can be processed on M_i and to ∞ otherwise.

McNaughton [62] proved already in 1959 that for identical machines there is no preemptive schedule with a finite number of preemptions for which the total completion time is strictly less than that of the optimal non-preemptive schedule. As we mentioned earlier there is an optimal schedule with at most $O(m^2n)$ preemptions [58], implying that McNaugthon's restriction to a finite number of preemptions may be removed. We generalize this theorem of McNaughton to the restricted unrelated machine model.

Theorem 3 *Let I be an instance of the problem $R|pmtn|\sum C_j$. If there are numbers $\{p_1, \ldots, p_n\}$ such that $p_{ij} \in \{p_j, \infty\}$ for all $i \in \{1, \ldots, m\}$ and $j \in \{1, \ldots, n\}$ then there exists an optimal schedule that is non-preemptive.*

Proof. Suppose the theorem is not true. Then there exists an instance with the smallest number of jobs for which its optimal preemptive schedule has a strictly smaller total completion time than any non-preemptive schedule. Let this be instance I with m machines and n jobs. Let σ^* be an optimal schedule among the non-preemptive schedules and let σ be a feasible schedule with $C_\sigma < C_{\sigma^*}$. Since we can assume that the number of preemptions is finite [58], we assume that σ has the smallest number of preemptions among all feasible schedules σ for I for which $C_\sigma < C_{\sigma^*}$. Without loss of generality we assume that no machine remains empty in σ.

Let T_i $(1 \leq i \leq m)$ be the completion time of machine M_i and let J_i be the job that is processed last on this machine. W.l.o.g. we assume $T_1 \leq \cdots \leq T_m$. We distinguish between the case in which all jobs J_i $(1 \leq i \leq m)$ are different and the case in which at least two are equal.

Case 1: All jobs J_i are different. Define the instance I' from I by removing the jobs $J_1, \ldots, J_m$ from the instance. By induction there exists an optimal schedule $\sigma^{'}$ for I' that is non-preemptive. From $\sigma^{'}$ we construct a non-preemptive schedule $\sigma^{''}$ for I by simply adding each job J_i at the end on

machine M_i. Since $\sigma^{'}$ is optimal it does not have idle time implying

$$\begin{aligned} C_{\sigma''} &= C_{\sigma'} + \sum_{i=1}^{m} C_{\sigma''}(J_i) \\ &= C_{\sigma'} + \sum_{j=1}^{n} p_j. \end{aligned} \tag{2.6}$$

Since $C_\sigma(J_i) \geq T_i$ and $\sum_{i=1}^{m} T_i \geq \sum_{j=1}^{n} p_j$ we have $\sum_{i=1}^{m} C_\sigma(J_i) \geq \sum_{j=1}^{n} p_j$. The optimality of $\sigma^{'}$ for $I^{'}$ now implies

$$\begin{aligned} C_\sigma &\geq C_{\sigma'} + \sum_{i=1}^{m} C_\sigma(J_i) \\ &\geq C_{\sigma'} + \sum_{j=1}^{n} p_j \\ &= C_{\sigma''}. \end{aligned} \tag{2.7}$$

Hence we constructed a non-preemptive schedule $\sigma^{''}$ the total completion time of which is at most the total completion time of the optimal preemptive schedule schedule σ. A contradiction.

Case 2: There are indices j and k, $1 \leq j < k \leq m$, such that job J_j is the same job as J_k. We make a small change in the schedule as follows. In the current schedule, some part of job J_j is processed on machine M_j up to time T_j, and then job J_j is completed on machines $M_{j+1}, \ldots, M_m$. In the new schedule, we continue processing job J_j on machine M_j after time T_j until its completion. Note that this reduces the number of preemptions of job J_j while its completion time does not increase. For all other jobs the number of preemptions and completion time remains the same. Again we obtain a contradiction. □

Since the non-preemptive problem can be solved in $O(n^3)$ time even in the more general case of unrelated machines [17], [49], we have the following corollary.

Corollary 1 *The problem $R|pmtn|\sum C_j$ can be solved in $O(n^3)$ time if we restrict to instances with $p_{ij} \in \{p_j, \infty\}$ for all $i \in \{1, \ldots, m\}$ and $j \in \{1, \ldots, n\}$ and some set $\{p_1, \ldots, p_n\}$.*

If we replace the $\sum C_j$ objective by the $\sum U_j$ objective, then the non-preemptive version is NP-hard in the strong sense since the problem is strongly NP-hard on identical parallel machines [36]. Simalarly, Lawler [57] proved that minimizing $\sum U_j$ on identical machines is ordinarily NP-hard if preemption is allowed. Hence, the preemptive version of our problem is ordinarily NP-hard.

McNaughton proved his theorem mentioned above for the total weighted completion time objective. Thus, NP-hardness of minimizing total completion time on identical parallel machines implies NP-hardness of the weighted version. It follows immediately that the weighted version of the problem of Corollary 1 is also NP-hard. However, it is not true that preemption is redundant for these problems as illustrated by the following example.

Example 1 We define an instance with machines M_1 and M_2 and jobs J_1, J_2 and J_3. The processing times are $p_{11} = 1, p_{21} = \infty, p_{12} = \infty, p_{22} = 1, p_{13} = p_{23} = 2$, and the weights are $w_1 = w_2 = 1, w_3 = 2$. One can easily check that the value of any non-preemptive schedule is at least 8. Now schedule job J_1 first on machine M_1. Schedule J_2 between time 1 and 2 on machine M_2 and schedule J_3 from 0 to 1 on machine M_2 and from 1 to 2 on machine M_1. The value of this preemptive schedule is 7. □

2.4 Postlude

Recently [81], we were able to prove that the problems $R|pmtn|\sum U_j$ and $R|pmtn|\sum C_j$ are $Max\text{-}SNP$-hard, using an *L-reduction* from the *maximal 3-dimensional matching* problem. The complexity of both problems is still open for a fixed number of machines, even for $m = 2$. Another open question is whether $R|\text{pmtn}, p_{ij} \in \{p_j, \infty\}|\sum U_j$ solvable in pseudopolynomial time or NP-hard in the strong sense.

Chapter 3

On-line scheduling so as to minimize total completion time

3.1 Introduction

We consider the problem of on-line minimizing total (weighted) completion on identical parallel machines with release time constraints. Formally, an instance I is given by n jobs J_j $(j = 1, \ldots, n)$ and m machines M_i $(i = 1, \ldots, m)$. Each job J_j $(j = 1, \ldots, n)$ has a given processing time p_j, release date r_j and weight w_j. In any feasible schedule a machine can process only one job at a time and a job must be processed without interruption on one machine. The processing of a job is not allowed to start before its release date. Given a schedule we write S_j for the start time of job J_j, and C_j for its completion time. The objective is to minimize $\sum_{j=1}^{n} w_j C_j$. We write $\textsc{Opt}(I)$ for the optimal value for instance I.

In Section 3.2 we consider the non-preemptive unweighted problem, i.e., all weights are 1. This is joint work with Leen Stougie and Xiwen Lu [60]. In Section 3.3 we consider the weighted problem with and without preemption. For each of the three problems we present an algorithm and a proof of its competitiveness. The competitive ratios for the non-preemptive problems do not improve on existing ratios. However, we generalize existing algorithms and present a new algorithm which is 1.56-competitiveness for the preemptive single-machine problem, improving the best known ratio of 2.

Our proof technique is similar in any of the three proofs and is based on work of Anderson and Potts [3]. We describe the general idea as follows. Given an on-line optimization problem we look for a simple property of a solution

that is sufficient for (near)-optimality. In the on-line algorithm we try to satisfy this property as much as possible and in the analysis we analyze how much we have to change the instance and/or solution such that the property holds. In this chapter we give some simple properties of parallel machine schedules for which we conjecture that they ensure near-optimality. We use these properties to express the competitive ratios of our algorithms.

3.2 Minimizing total completion time

Hoogeveen and Vestjens [48] (see Lemma 2) show that no deterministic on-line algorithm can have a competitive ratio smaller than 2 for the problem on a single machine. Several single machine algorithms with competitive ratio matching this lower bound have been given in the literature. Phillips, Stein and Wein [70] presented a 2-competitive algorithm based on the optimal preemptive schedule. Hoogeveen and Vestjens used the idea of shifted release dates to obtain a 2-competitive algorithm. The same idea was used by Stougie (cited in [86]) who obtained a third algorithm. However, 2-competitiveness of this algorithm was never proved. The algorithm that we present in this section generalizes these last two algorithms.

For the problem on m identical parallel machines, Chekuri et al. [21] gave an on-line algorithm that is $3 - 1/m$-competitive. They construct a preemptive schedule on a single machine, and use the order of completion times of the jobs in this schedule obtain a non-preemptive schedule on identical parallel machines. A general lower bound of 1.309 on the competitive ratio of any algorithm for this problem is given by Vestjens [86].

For some special cases an optimal schedule can be found in a very simple way. If all jobs are released at time zero, then the parallel-machine problem is solved by list scheduling the jobs in order of increasing processing times [26], i.e. jobs are scheduled one by one in the order of the list and as early as possible. This algorithm is known as the *shortest processing time* (SPT) rule. Schrage [73] showed that an optimal schedule for the unweighted preemptive single-machine problem with release date constraints is obtained if at any moment the job with the shortest remaining processing time is processed. This algorithm is known as the *shortest remaining processing time* (SRPT) rule.

The computational complexity of the off-line problems is well understood. Lenstra et al. [59] proved NP-hardness in the strong sense for the non-preemptive single-machine problem. The preemptive problem is ordinarily NP-hard for two machines [28]. A polynomial time approximation scheme for the (preemptive) problem on identical parallel machines was given by Afrati et al. [1]. This scheme even applies to the case of a fixed number of unrelated machines with weighted completion time objective.

We present the single-machine lower bound since it indicates a property that any on-line algorithm for minimizing total completion time should have.

Lemma 2 *(Hoogeveen and Vestjens [48])*
For the single-machine problem there is no deterministic on-line algorithm with competitive ratio $(2-\epsilon)$*, for any* $\epsilon > 0$.

PROOF. Assume that an algorithm A is $2-\epsilon$-competitive for some $\epsilon > 0$. We will prove that there exists an instance, depending on A, that contradicts this assumption. We imagine that an adversary builds the instance on-line based on the steps of A. The adversary releases a job J_1 with processing time $p_1 = 1$ at time 0. If, according to A, job J_1 starts later than $1-\epsilon$, then no more jobs are given and the competitive ratio becomes $(S_1+1)/1 > 2-\epsilon$, where S_1 is the starting time of J_1. A contradiction. If, on the other hand, $S_1 \leq 1-\epsilon$, then $n-1$ jobs are presented at time $1/2 + S_1/2$, each with processing time 0. The total completion time of the schedule produced by A is at least $n(S_1+1)$, whereas the optimal schedule has total completion time $n(1/2 + S_1/2) + 1$. Hence, the competitive ratio tends to 2 if n tends to infinity, which contradicts the assumption that A is $2-\epsilon$-competitive. □

In order to be 2-competitive on a single machine, no job J_j can start before time p_j, but the processing of a job cannot be postponed too long. The algorithms of Hoogeveen and Vestjens [48] and Stougie (cited in [86]) both use a similar approach: At the release of any job J_j it is assigned a new release date $q_j \geq r_j$; the SPT rule is applied to the jobs with these shifted release dates, always scheduling the shortest available job. Hoogeveen and Vestjens define $q_j = \max\{r_j, p_j\}$ while Stougie chooses $q_j = r_j + p_j$. Both algorithms are 2-competitive, although a proof has never been published for the latter algorithm. The DELAYED-SPT algorithm defined below generalizes these algorithms.

DELAYED-SPT:
At the release of a job J_j shift its release date r_j to q_j , where q_j is an arbitrary number in the interval $[\max\{r_j, p_j\}, r_j + p_j]$. Apply the SPT rule with the delayed release dates.

We analyze the performance of this algorithm for identical parallel machines. The SPT rule is readily applicable on identical parallel machines. Also SRPT has a simple extension to identical parallel machines: At any moment process the m jobs with smallest remaining processing time among the available jobs. Another preemptive rule is the *preemptive shortest processing time* rule. The PREEMPTIVE-SPT rule is similar to SRPT except that the order of the jobs in the list is not based on remaining processing time but on the original processing

time. Although this rule does not provide optimal schedules in general for the total completion time objective, we use it below in the analyzeis of DELAYED-SPT.

Definition 7 *Given any schedule σ for some instance I, we say that σ is an* SPT schedule *with respect to I if it is the output of the SPT rule applied to I (assuming that ties are broken in favor of σ). We define an* SRPT schedule *and a* PREEMPTIVE-SPT schedule *in a similar way.*

Let $\mathcal{I}_1$ be the set of instances for which there exists an SRPT schedule in which no job is preempted. We define α as the maximum, taken over all $I \in \mathcal{I}_1$, of the ratio between the value of the SRPT schedule of I and OPT(I).

Similarly, let $\mathcal{I}_2$ be the set of instances for which there exists a PREEMPTIVE-SPT schedule in which no job is preempted. We define β as the maximum, taken over all $I \in \mathcal{I}_2$, of the ratio between the value of the PREEMPTIVE-SPT schedule of I and OPT(I).

Notice that any PREEMPTIVE-SPT schedule in which no job is preempted is also an SRPT-schedule, implying $\mathcal{I}_2 \subseteq \mathcal{I}_1$, whence $\alpha \geq \beta$.

Theorem 4 *Algorithm* DELAYED-SPT *is 2α-competitive for the problem of minimizing total completion time on identical parallel machines with jobs arriving over time. If $q_j = \max\{r_j, p_j\}$ for all j, then* DELAYED-SPT *is 2β-competitive.*

PROOF. Let σ be the schedule produced by the DELAYED-SPT algorithm for some instance I with job set J, and let $Z(\sigma)$ be its total completion time. We relate the value $Z(\sigma)$ to the optimal value OPT(I) through a modified instance I'. This instance is defined from the original instance I and schedule σ. First, we show that σ is an optimal schedule for the modified instance. Next we use any optimal schedule for I to bound OPT(I') from above. These two observations together prove the theorem.

The modified instance I' is defined from I and from the on-line schedule σ. For each job $J_j \in J$ we define one job J'_j with parameters r'_j and p'_j. The processing time remains the same: $p'_j = p_j$, and the release date is changed to $r'_j = \min\{S_j, 2r_j + p_j\}$, where S_j is the starting time of job J_j in σ.

First, we prove that σ satisfies the SRPT rule with respect to the modified instance I'. Assume that at time t one of the machines is processing a job J_k and job J_j is available for instance I', i.e. $r'_j \leq t$ and J_j is not processed at time t, hence $r'_j = 2r_j + p_j$. If $q_j \leq S_k$, then we must have $p_j \geq p_k$ since otherwise the SPT-rule would have been violated at time S_k. Notice that $S_k + p_k \leq 2S_k$ since job J_k did not start before time p_k. Therefore,

$q_j > S_k$ implies that the remaining processing time of job J_k at time t is $S_k + p_k - t \leq 2S_k - t < 2r_j + 2p_j - t = r'_j + p_j - t \leq p_j$.

We conclude that

$$Z(\sigma) \leq \alpha \textsc{Opt}(I'). \tag{3.1}$$

Now assume that $q_j = \max\{r_j, p_j\}$. Again, if $q_j \leq S_k$ then we must have $p_j \geq p_k$ since otherwise the SPT-rule would have been violated at time S_k. If $q_j > S_k$ then $2r_j + p_j = r'_j \leq t < S_k + p_k \leq 2S_k < 2q_j = 2\max\{r_j, p_j\}$. Thus, $q_j = \max\{r_j, p_j\} = p_j$ implying $p_k \leq S_k < q_j = p_j$. We conclude that

$$Z(\sigma) \leq \beta \textsc{Opt}(I'). \tag{3.2}$$

Now we show that the optimal value for I' is at most twice the optimal value for instance I. Consider any optimal schedule for I. We transform this schedule into a feasible schedule for I' as follows. If job J_j is scheduled between time t and $t + p_j$, then we process job J'_j on the same machine between time $2t + p_j$ and $2t + 2p_j$. Clearly, this is a feasible schedule for I' and all completion times are doubled. Hence,

$$\textsc{Opt}(I') \leq 2\textsc{Opt}(I). \tag{3.3}$$

Combining (3.1), (3.2) and (3.3) completes the proof. □

Since the SRPT rule produces an optimal schedule for the preemptive single-machine problem we have the following corollary.

Corollary 2 *On a single machine the value of the* Delayed-SPT *schedule is at most twice the value of an optimal preemptive schedule.*

Unfortunately, SRPT is not optimal for the preemptive problem on parallel machines. This is not surprising since Du et al. [28] proved that this problem is NP-hard. Phillips et al. [70] showed that the SRPT schedule is not worse than twice the optimal preemptive schedule. This bound combined with Theorem 4 implies that Delayed-SPT is 4-competitive for the non-preemptive identical parallel machines problem. Although this is larger than the $3-1/m$-competitive algorithm of Chekuri et al. [21], a proof of a smaller ratio for SRPT would yield a proof of a smaller ratio for our algorithm. Finding the competitive ratio of SRPT on identical parallel machines seems a very interesting problem on its own. We conjecture that this ratio is much smaller than 2.

The best ratio for Delayed-SPT that can be derived from our proof is 2β. Unfortunately we do not know of an upper bound for β smaller than 2. We conjecture that the values of α and β are close to the lower bounds provided in the following two propositions.

Proposition 1 *The worst-case ratio α between a non-preemptive SRPT schedule and the corresponding optimal non-preemptive schedule is at least $\frac{12}{11}$.*

PROOF. Define an instance with 2 machines and 5 jobs with parameters $p_1 = p_2 = p_4 = p_5 = 1$ and $p_3 = 2$ and $r_1 = r_2 = r_3 = 0$, and $r_4 = r_5 = 2$. A feasible SRPT-schedule is induced by the starting times $S_1 = S_2 = 0$, $S_3 = 1$, $S_4 = 2$, and $S_5 = 3$ and has total completion time 12. The optimal schedule starts job 1 and 3 at time 0 and has value 11. □

Proposition 2 *The worst-case ratio β between a* PREEMPTIVE-*SPT schedule without preemptions and the corresponding optimal non-preemptive schedule is at least $\frac{14}{13}$.*

PROOF. The instance of the proof above is adjusted a little bit. Take 2 machines and 5 jobs with parameters $p_1 = p_2 = 1$ and $p_3 = p_4 = p_5 = 2$, and $r_1 = r_2 = r_3 = 0$ and $r_4 = r_5 = 2$. A feasible PREEMPTIVE-SPT schedule is induced by the starting times $S_1 = S_2 = 0$, $S_3 = 1$, $S_4 = 2$, and $S_5 = 3$ and has total completion time 14. The optimal schedule starts job 1 and 3 at time 0 and has value 13. □

Both examples apply to any even number of machines. For odd numbers we have slightly weaker lower bounds. Proposition 2 implies that, through Theorem 4, we will not be able to prove a competitive ratio smaller than 28/13 for DELAYED-SPT. Additionally, an instance of the scheduling problem with only one job released at time 0 and processing time 1 shows that its ratio is at least 2 on any number of machines.

The interesting question remains if an on-line algorithm for the problem on identical parallel machines exists with competitive ratio strictly less than 2. The single-machine lower bound does not extend to parallel machines: If m jobs with processing time 1 are given at time zero, then we could gradually start processing them from time 0. In this way there are always some machines that are available or will become available soon. The best lower bound is 1.309 [48]. A competitive ratio strictly smaller than 2 would be a divergence from the general phenomenon in on-line scheduling in which competitive ratios of algorithms for multiple machine problems are higher than those for their single machine counterparts (see [77]).

3.3 Minimizing total weighted completion time

The problem of minimizing total weighted completion time on a single machine with release date constraints is a well-studied problem in scheduling theory. In the last decade many algorithm have been developed for several variants of the

off-line and the on-line problem. The complexity of the off-line problems is well-understood. The preemptive single machine problem is strongly NP-hard [56] and the non-preemptive single machine problem is strongly NP-hard even if all job weights are 1 [59]. The earlier mentioned polynomial time approximation scheme by Afrati et al. [1] applies also to the weighted problems that we consider in this section.

In Section 3.3.2 we consider the non-preemptive setting on identical parallel machines, and in Section 3.3.3 we consider the preemptive setting on a single machine. For the first problem we present a simple proof for 2-competitiveness of a deterministic on-line algorithm proposed by Anderson and Potts [3] and show how to extend this algorithm to parallel machines. We use the same proof technique in Section 3.3.3 to prove that a simple modification of Smith's ratio rule yields a 1.56-competitive algorithm for the preemptive problem. Preliminary to both results we introduce the *mean busy date relaxation*.

3.3.1 Mean busy date relaxation

The *mean busy date* of a job in a schedule is the average of all moments in which it is processed. Formally, let $\delta_j(t)$ be the indicator function of a given schedule σ, i.e. $\delta_j(t) = 1$ if job J_j is processed at time t and $\delta_j(t) = 0$ otherwise. The mean busy date M_j of job J_j is defined by

$$M_j = \frac{1}{p_j} \int_{r_j}^{C_j} \delta_j(t) t \cdot dt.$$

Lemma 3 *For any job J_j, we have $M_j \leq C_j - p_j/2$. On a single machine, equality holds if and only if job J_j is not preempted.*

Proof. If job J_j is not preempted then δ_j is 1 between $C_j - p_j$ and C_j implying $M_j = C_j - p_j/2$. Conversely, if the processing of J_j is interrupted, then clearly $M_j < C_j - p_j/2$. □

A lower bound on the total weighted completion time $\sum w_j C_j$ is the total weighted mean busy date $\sum w_j M_j$ plus $\frac{1}{2}\sum w_j p_j$. Several models to minimize this lower bound for the single machine have been investigated. Dyer and Wolsey [29] formulate it as an LP (linear program) with variables $y_j(t) \geq 0$, $(j = 1 \ldots n,\ t = 0, 1, \ldots, L)$, where L is an upper bound on the makespan of an optimal schedule. Intuitively, a machine can work on multiple jobs at the same moment. Each job J_j is processed in the interval $[t, t+1]$ with speed $y_j(t)$. Goemans [39] gave an LP in which M_j, $j = 1 \ldots n$, are the only variables, and proved that the optimum of this problem is equal to the optimum of the time indexed LP of Dyer and Wolsey.

The relaxation ($\mathcal{R}$) below is based on the model of Dyer and Wolsey. Since we we do not care about computational issues of the relaxation and want to use it only as a tool in our proofs, we choose a general formulation with functions $\delta_j(t) \geq 0$, ($j = 1 \ldots, n$, $t \in \mathbb{R}^+$). This enhances the notation and we do not have to consider integrality issues.

$$
\begin{array}{lll}
\text{minimize} & \sum_j \frac{w_j}{p_j} \int_{r_j}^{\infty} \delta_j(t) t \cdot dt & \\
\text{subject to} & \int_{r_j}^{\infty} \delta_j(t) \cdot dt = p_j & \text{for } j = 1, \ldots, n; \\
& \sum_j \delta_j(t) \leq 1 & \text{for all } t \geq 0; \\
& \delta_j(t) \geq 0 & \text{for } j = 1, \ldots, n \text{ and } t \geq 0.
\end{array}
\qquad (\mathcal{R})
$$

To avoid pathological situations we require that, in a feasible solution of ($\mathcal{R}$), any function $\delta_j(t)$ is piecewise constant with a finite number of pieces which all have a strictly positive length. We call any feasible solution a *pseudo schedule*.

We denote $\rho_j = w_j/p_j$. Given a set of jobs we say that job J_j has *highest priority* if there is no job J_k with $\rho_k > \rho_j$. We call a complete ordering of the jobs, in which $J_j \prec J_k$ for any two jobs with $\rho_j > \rho_k$, a *ρ-ordering*. Notice that a ρ-ordering can be obtained on-line, i.e. at time r_j we can fix the order of job J_j with respect to any job released before time r_j.

Smith [82] showed that if all release dates are zero, then an optimal solution for the preemptive single machine problem is obtained by sequencing jobs in non-increasing order of ratio w_j/p_j. This method is known as Smith's ratio rule or as the *weighted shortest processing time* (WSPT) rule. It becomes the SPT rule in the case of unit weights. For non-trivial release dates the WSPT rule is no longer optimal. Even if the rule is applied preemptively, i.e. the job on the machine is replaced by a job with higher ratio at the moment that such a job is released, then this Preemptive-WSPT-rule is not optimal for the preemptive problem. However, Goemans et al. showed that this rule is optimal for problem ($\mathcal{R}$) [42].

Preemptive-WSPT:
Maintain a ρ-ordering of the jobs. At any point in time schedule (preemptively) the highest priority job.

A schedule is called a PREEMPTIVE-WSPT schedule if it can be produced by the PREEMPTIVE-WSPT Algorithm for some ρ-ordering of the jobs.

Lemma 4 *(Goemans et al. [42])*
A pseudo schedule is optimal for $(\mathcal{R})$ *if and only if it is a* PREEMPTIVE-WSPT *schedule.*

For a given pseudo schedule σ we denote the objective value $\sum w_j M_j$ by $Z^M(\sigma)$ and we denote the optimal value of $(\mathcal{R})$ for a given instance I by $\text{OPT}^M(I)$. A generalization of the PREEMPTIVE-WSPT algorithm to m identical parallel machines is to schedule at any moment the m jobs with highest priority among the available jobs. The relaxation $(\mathcal{R})$ can easily be generalized to identical parallel machines. We denote this also by $(\mathcal{R})$ and by omit the details.

Proposition 3 PREEMPTIVE-WSPT *on parallel machines is not better than* $5/4$*-competitive with respect to the optimal value of* $(\mathcal{R})$*.*

PROOF. Consider the instance consisting of two machines and three jobs with parameters $p_1 = p_2 = 1$, $p_3 = 2$, $r_1 = r_2 = r_3 = 0$, and $w_1 = w_2 = 1$, $w_3 = 2$. PREEMPTIVE-WSPT might choose to start job 1 and 2 at time 0 and job 3 at time 1, which leads to objective value 5. The optimal schedule starts job 3 at time 0 and has value 4. □

If all weights are 1 then a PREEMPTIVE-SPT schedule is exactly a PREEMPTIVE-WSPT schedule. Thus, it also follows from Proposition 2 that PREEMPTIVE-WSPT is not optimal for identical parallel machines. On the other hand, an upper bound on the ratio of PREEMPTIVE-WSPT would yield an upper bound on the ratio β defined in Section 3.2.

3.3.2 Non-Preemptive Scheduling

The first constant factor approximation algorithm for the single machine was given in the paper of Phillips et al. [69] and had a factor $16 + \epsilon$. Many improvements followed [74][46][19][70]. Goemans [40] used the concept of α-points, which were introduced in [69]. For any $\alpha \in [0, 1]$ and any j he defined the α-point $t_j(\alpha)$ as the time at which αp_j units of job J_j have been processed in the preemptive WSPT schedule. For a fixed α the non-preemptive schedule is obtained by processing the jobs in order of their α-points. Goemans showed that this yields a deterministic on-line $1+\sqrt{2}$–competitive algorithm for $\alpha = 1/\sqrt{2}$. If α is chosen independently for each job from a specific distribution (see [42]), then a randomized 1.685-approximation is obtained.

The only 2-competitive algorithm for the non-preemptive single-machine problem was given by Anderson and Potts [3]. Although their proof technique

is elegant, the actual proof is rather long and technical. Even before publication in 2002, Queyranne [71] gave a different proof using an LP relaxation as a lower bound. His proof follows quite easily from properties of this relaxation, which was studied extensively by Goemans [39]. This emphasizes the value of this relaxation, although there does not seem to be a general method in the way the relaxation is applied.

We will combine the good ideas from both proofs. Like Anderson and Potts we define a modified instance I' and a modified schedule σ' from the original instance I and the corresponding on-line schedule σ. Instead of proving optimality of σ' for I', we prove that σ' is optimal for a *relaxation* of I'. This relaxation is related to the one used by Queyranne. Additionally, we give an upper bound on the optimal value for the relaxed modified instance. This part in particular seems much easier in our proof than in the original proof by Anderson and Potts, who bound the optimal value of the *unrelaxed* modified instance. The simplification of the proof allows a generalization of the originally proposed algorithm.

In Section 3.2 we showed that the DELAYED-SPT algorithm is best possible for the non-preemptive problem with unit weights on a single machine. A natural extension of this algorithm to the weighted problem is to shift the release dates to $\max\{r_j, p_j\}$ and then apply the WSPT-rule respecting the new release dates. Recently, Megow [63] showed that the the algorithm is 3-competitive on a single machine and 4-competitive on identical parallel machines. The following example shows that the competitive ratio on a single machine cannot be strictly smaller than 3.

Example 2 *There are two jobs with parameters* $(r_1, p_1, w_1) = (0, p, 1)$ *and* $(r_2, p_2, w_2) = (0, p+1, W)$*, where* p *and* W *are large numbers. The proposed algorithm will start job 1 at time* p *and job 2 at time* $2p$*, yielding an objective value of* $p + W(3p+1)$*. In the optimal solution job 2 starts at time* 0 *and job 1 starts at time* $p+1$ *and has value* $W(p+1) + 2p + 1$*. The competitive ratio tends to 3 if* W *tends to infinity.*

The 2-competitive algorithm from Anderson and Potts[3] is a slightly different extension of the DELAYED-SPT algorithm than the one we discussed above.

DELAYED-WSPT (Single machine):
Maintain a ρ-ordering of the jobs as described. At any moment t that the machine is idle consider the job J_j with highest priority among the available, yet unscheduled jobs. If $t \geq p_j$ then start processing job J_j, else wait.

We give an alternative proof of 2-competitiveness of DELAYED-WSPT, which was first proven by Anderson en Potts [3]

Theorem 5 *(Anderson and Potts[3])*
The DELAYED-WSPT *algorithm is 2-competitive on a single machine, with respect to the optimal preemptive schedule.*

PROOF. Let σ be the schedule produced by DELAYED-WSPT for an arbitrary instance I with job set J, and let $Z(\sigma)$ be its total weighted completion time. We relate the value $Z(\sigma)$ to the optimal value $\text{OPT}(I)$ of the preemptive problem through a modified instance I'. For this instance our objective is to minimize total weighted mean busy date. We modify σ such that it becomes optimal for I'.

We say that at time t the machine *is waiting for* job J_j if the machine is idle at time t and job J_j has highest priority among the available jobs at that moment, but the machine does not start job J_j, i.e. $t < p_j$. Notice that at any moment, either the machine is processing a job, or the machine is idle and no jobs are available, or the machine is idle and waiting for some specific job J_j. We denote by F_j the total time that the machine waited for job J_j.

The modified instance I' is defined by I and by the on-line schedule σ. For each job $J_j \in J$ we define one job J'_j with parameters r'_j, p'_j and w'_j. We define $p'_j = p_j + F_j$. If $F_j > 0$, i.e. the machine waited for job J_j then let s_j be the earliest moment at which it waited for J_j. If $F_j = 0$, then let $s_j = S_j$, i.e. the start time of job J_j. We define $r'_j = \min\{2r_j, s_j\}$. Notice that this implies $r'_j \geq r_j$. Finally, we define $w'_j = w_j p'_j / p_j$ so as to keep the ratio w'_j/p'_j equal to w_j/p_j, preserving the ρ-ordering on the jobs, i.e. $J_j \prec J_k \Leftrightarrow J'_j \prec J'_k$.

We extend schedule σ to a feasible schedule σ' for I' in the obvious way: job J'_j is processed at time t in σ' if either job J_j is processed or is waited for at time t in σ. We show that σ' is a PREEMPTIVE-WSPT schedule for I' and hence is optimal with respect to the mean busy date objective. Consider an arbitrary moment t. We have to show that σ' is processing the job with highest w_j/p_j-ratio amongst the jobs available w.r.t. I'. This follows immediately if σ' is idle at time t since in that case no jobs are available in σ and therefore neither in σ'. If σ' is processing a job while at the same time σ is waiting for this job, then clearly σ' is processing the highest priority job. Now assume that at time t job J_j and J'_j are processed in σ and σ' while job J'_k is available for σ'. We will prove that $J_j \prec J_k$, hence $J'_j \prec J'_k$.

If the machine waited for job J_k in σ, then it was doing so before time S_j. This immediately implies that $J_j \prec J_k$ since the algorithm gave priority to job J_j at time S_j. Now assume the machine did not wait for job J_k. Since job J'_k starts at S_k and is released strictly before it starts processing, we must have $r'_k < S_k$, implying $r'_k = \min\{2r_k, s_k\} = \min\{2r_k, S_k\} = 2r_k$. Since job J_j did not start before time p_j we know that $r'_k \leq t < S_j + p_j \leq 2S_j$, implying $r_k < S_j$. Again we conclude that $J_j \prec J_k$ since the algorithm gave priority to job J_j while job J_k was available.

Now we compare the values $Z(\sigma)$ and $Z^M(\sigma')$ (the total mean busy date of σ'). Let the indicator function $y_j(t)$ $(t \geq 0, j = 1 \ldots n)$ be 1 if σ is waiting for job J_j at time t and 0 otherwise. We define G_j as the contribution of the first F_j units of job J_j in σ' to the weighted mean busy date, i.e. $G_j = \rho_j \int y_j(t) t \cdot dt$. The mean busy date is then,

$$w'_j M'_j = \rho_j \int y_j(t) t \cdot dt + \rho_j \int_{C_j - p_j}^{C_j} t \cdot dt = G_j + w_j C_j - \frac{1}{2} w_j p_j.$$

Hence,

$$\textsc{Opt}^M(I') = Z^M(\sigma') = \sum_j G_j + Z(\sigma) - \frac{1}{2} \sum_j w_j p_j. \tag{3.4}$$

Next, we bound $\textsc{Opt}^M(I')$ from above. We use an optimal schedule for I and schedule σ to construct a pseudo schedule for I'. Let the function $x_j(t)$ $(t \geq 0, j = 1 \ldots n)$ be the indicator function of an optimal schedule for I. We will show that a pseudo schedule for I' is given by the function $\delta_j(t)$:

$$\delta_j(t) = \frac{1}{2} x_j(t/2) + \frac{1}{2} y_j(t/2), \text{ for all } t \geq 0.$$

Intuitively, we take the optimal schedule and slow down the processing by a factor 2. Parallel to that we process the jobs as in σ', restricted to the first F_j processing units of p'_j, and also slowed down by a factor 2. First, since $\sum \delta_j(t) = 1/2 \sum x_j(t) + 1/2 \sum y_j(t) \leq 1$, the machine is not overloaded. Second, no job starts before time $\min\{2r_j, 2s_j\} \geq \min\{2r_j, s_j\} = r'_j$. Third, each job J'_j is processed for exactly p'_j processing units.

$$\begin{aligned} \int_0^\infty \delta_j(t) \cdot dt &= \int_0^\infty \tfrac{1}{2} x_j(t/2) \cdot dt + \int_0^\infty \tfrac{1}{2} y_j(t/2) \cdot dt \\ &= \int_0^\infty x_j(t) \cdot dt + \int_0^\infty y_j(t) \cdot dt = p_j + F_j = p'_j. \end{aligned}$$

If we denote by M^*_j and C^*_j, respectively, the mean busy date and completion time of job J_j in the optimal schedule for I, then by Lemma 3 $M^*_j \leq C^*_j - p_j/2$. Now we are ready to bound the total weighted mean busy date of the constructed schedule.

$$\begin{aligned} \rho_j \int_0^\infty \delta_j(t) t \cdot dt &= \rho_j \int_0^\infty \tfrac{1}{2} x_j(t/2) t \cdot dt + \rho_j \int_0^\infty \tfrac{1}{2} y_j(t/2) t \cdot dt \\ &= 2\rho_j \int_0^\infty x_j(t) t \cdot dt + 2\rho_j \int_0^\infty y_j(t) t \cdot dt \\ &\leq 2\rho_j p_j (C^*_j - \tfrac{p_j}{2}) + 2G_j \\ &= 2w_j C^*_j - w_j p_j + 2G_j. \end{aligned}$$

Hence,

$$\textsc{Opt}^M(I') \leq 2\textsc{Opt}(I) - \sum_j w_j p_j + 2\sum_j G_j. \tag{3.5}$$

Combing (3.4) and (3.5) we obtain

$$Z(\sigma) \leq 2\textsc{Opt}(I) - \frac{1}{2}\sum_j w_j p_j + \sum_j G_j. \tag{3.6}$$

Since the machine does not wait for job J_j at a time $t \geq p_j$ we have

$$G_j \leq \rho_j \int_0^{p_j} t \cdot dt = \frac{1}{2} w_j p_j,$$

completing the proof. □

The proof of Theorem 5 above enables us to create some freedom in the algorithm in the way we did for the unweighted problem in Section 3.2. In order to stay 2-competitive it is sufficient if the following three conditions are satisfied: First, no job J_j starts before time p_j. Second, at the moment a job starts it has highest priority in the ρ-ordering among the available jobs. Third, the sum of the two series in equation (3.6) is at most zero. This last condition is clearly satisfied if the machine does not wait for a job J_j after time p_j. However, incorporating this condition directly in the algorithm itself would allow to wait for a job J_j even after time p_j. Define $G_j(t) = \rho_j \int_0^t y_j(s) s \cdot ds$, where, as before, $y_j(t)$ is 1 if the machine is waiting for job J_j at time t, and 0 otherwise. Now define $G(t) = \sum_j G_j(t)$, where the sum is taken over all jobs that have been released until time t. The conditions in the Delayed-WSPT algorithm then become: If $t < p_j$, then wait; if $G(t) \geq \sum_j w_j p_j/2$, then start processing job J_j; In the other case either wait or start processing J_j. These conditions give the most general form of the Delayed-WSPT that can be derived directly from our proof. However, if we prefer to have an easy expression in terms of the job parameters, then we can use the following observation:

$$\rho_j \int_{r_j}^{\sqrt{r_j^2+p_j^2}} t \cdot dt = \frac{1}{2} w_j p_j.$$

Delayed-WSPT (extended):
Maintain a ρ-ordering of the jobs as described. At any moment t that the machine is idle consider the highest priority job J_j among the available jobs. If $t < p_j$, then wait. If $p_j \leq t < \sqrt{r_j^2 + p_j^2}$ then either wait or start processing job J_j. In the other case start processing job J_j.

The single machine algorithm and the proof of Theorem 5 can easily be generalized to identical parallel machines. However, we can no longer prove 2-competitiveness since PREEMPTIVE-WSPT is not optimal for identical parallel machines.

DELAYED-WSPT:
Maintain a ρ-ordering of the jobs. At any moment t consider the k highest priority jobs among the available jobs, where k is the number of idle machines at time t. For any job J_j in this set apply the following rule. If $t \geq p_j$ then start processing job J_j, else wait.

Theorem 6 DELAYED-WSPT *is 2γ-competitive for identical parallel machines, where γ is the competitive ratio of* PREEMPTIVE-WSPT *for the preemptive version of the problem on identical parallel machines.*

PROOF. The proof is similar to the single machine proof. We say that the machines are waiting for job J_j at time t if job J_j is in the set of highest priority jobs at time t but does not start at time t (since $t < p_j$). The instance I' and σ' are defined in the same way. The equality $\text{OPT}^M(I') = Z^M(\sigma')$ changes to $\gamma\text{OPT}^M(I') \geq Z^M(\sigma')$. The equations (3.4) and (3.6) now become:

$$\gamma\text{OPT}^M(I') \geq Z^M(\sigma') = \sum_j G_j + Z(\sigma) - \frac{1}{2}\sum_j w_j p_j,$$

$$Z(\sigma) \leq 2\text{OPT}(I) - (\gamma - \frac{1}{2})\sum_j w_j p_j + (2\gamma - 1)\sum_j G_j.$$

Combining this with $\gamma \geq 1$ and $G_j \leq \frac{1}{2} w_j p_j$ completes the proof. □

We can prove a slightly better ratio than 2γ if we replace the condition $t \geq p_j$ by $t \geq (1+\epsilon)p_j$ for some small ϵ. We skip the calculations.

3.3.3 Preemptive Scheduling

The DELAYED-WSPT algorithm produces a non-preemptive schedule with value at most twice the value of an optimal preemptive schedule. This follows also from Queyranne's proof but is not mentioned in the paper of Anderson and Potts. Epstein and Van Stee [30] show that no deterministic on-line algorithm can have a competitive ratio smaller than 1.073 for the problem with preemption. Several other 2-competitive algorithms have been given for the preemptive single-machine problem (see [43],[63]). Schulz and Skutella [75] gave a randomized 4/3-competitive algorithm for the single machine. Megow and Schulz

proved that the competitive ratio of PREEMPTIVE-WSPT is 2 on identical parallel machines. In the worst-case instance this algorithm preempts jobs at the moment they are almost completed. To overcome this problem Megow [63] considers the *shortest weighted remaining processing time* (SWRPT) rule, in which at any moment the job with largest ratio between weight and remaining processing time is processed. Megow shows a lower bound of 1.16 on a single machine and conjectures that the competitive ratio is much smaller than 2. Megow et al. [64] show that this ratio is at most 2 on identical parallel machines.

Our algorithm has a parameter $c > 1$ and it applies the PREEMPTIVE-WSPT with the restriction that a job cannot be preempted at a time t if it can be completed before time ct.

DELAYED-PREEMPTIVE-WSPT[c]: (Single machine)
Maintain a ρ-ordering of the jobs. At any moment schedule the job that has highest priority, with the restriction that a job is never preempted at a moment t if its remaining processing time at that moment is not more than $(c-1)t$.

Theorem 7 *Algorithm* DELAYED-PREEMPTIVE-WSPT[c] *is c-competitive on a single machine for any* $c \geq \upsilon$*, where* υ *is the real root of* $2\upsilon^3 - 4\upsilon^2 + 2\upsilon - 1$ *(*$\upsilon \approx 1.56$*).*

PROOF. Let σ be the schedule produced by DELAYED-PREEMPTIVE-WSPT[c] for instance I with job set J, and let $Z(\sigma)$ be its total weighted completion time. We relate the value $Z(\sigma)$ to the optimal value $\text{OPT}(I)$ through a modified instance I'. For this instance our objective is to minimize total weighted mean busy date.

The modified instance I' is defined by I and by the on-line schedule σ. For each job $J_j \in J$ we define one job J'_j with parameters r'_j, p'_j and w'_j. Each of the three parameters depends on I and σ. The shifted release date is given by $r'_j = \min\{cr_j, S_j\}$, where S_j is the start time of job J_j in σ. Assume $c \geq 1$. We denote by u_j the moment at which job J_j is preempted for the last time in schedule σ and denote by F_j the total time that J_j is processed before time u_j. Now we define $p'_j = p_j + F_j$. We define $w'_j = w_j p'_j / p_j$ so as to keep the ratio w'_j/p'_j equal to w_j/p_j. We preserve the same ρ-ordering on the jobs, i.e. $J_k \prec J_j \Leftrightarrow J'_k \prec J'_j$.

The rest of the proof consists of three parts. First we bound $\text{OPT}^M(I')$ from above using an optimal schedule for I and using σ. Next we compare the values $Z(\sigma)$ and $\text{OPT}^M(I')$ and in the last part we combine the two results.

Let the function $x_j(t)$ $(t \geq 0, j = 1 \ldots n)$ be the indicator function of an optimal schedule for I. Let the function $y_j(t)$ $(t \geq 0, j = 1 \ldots n)$ be 1 if σ is processing job J_j at time $t \leq u_j$ and 0 otherwise. A feasible schedule for I' is

now given by the function $\delta_j(t)$:

$$\delta_j(t) = \frac{1}{c} x_j(t/c) + \frac{c-1}{c} y_j\left((c-1)t/c\right), \text{ for all } t \geq 0.$$

Intuitively, we take the optimal schedule and slow down the processing by a factor c. Parallel to that we process the jobs as in σ, slowed down by a factor $c/(c-1)$ and restricted to the first F_j processing units. We will show that $\delta_j(t)$ defines a pseudo schedule for I'. Since

$$\sum_j \delta_j(t) = \frac{1}{c} \sum_j x_j(t/c) + \frac{c-1}{c} \sum_j y_j((c-1)t/c) \leq 1/c + (c-1)/c = 1,$$

the machine is not overloaded. Second, no job starts before time $\min\{cr_j, \frac{c}{c-1} S_j\} \geq \min\{cr_j, S_j\} = r'_j$. Third, each job J'_j is completely processed.

$$\begin{aligned} \int_0^\infty \delta_j(t) \cdot dt &= \int_0^\infty \tfrac{1}{c} x_j(t/c) \cdot dt + \int_0^\infty \tfrac{c-1}{c} y_j((c-1)t/c) \cdot dt \\ &= \int_0^\infty x_j(t) \cdot dt + \int_0^\infty y_j(t) \cdot dt = p_j + F_j = p'_j. \end{aligned}$$

Now we define G_j as the contribution of the first F_j processing units of J_j in σ to the weighted mean busy date in σ, i.e. $G_j = \rho_j \int_0^\infty y_j(t) t \cdot dt$. If we denote by M_j^* and C_j^* the mean busy date and completion time of job J_j in the optimal schedule, then by Lemma 3 $M_j^* \leq C_j^* - p_j/2$. Now we are ready to bound the total mean busy date of the constructed schedule.

$$\begin{aligned} \rho_j \int_0^\infty \delta_j(t) t \cdot dt &= \rho_j \int_0^\infty \tfrac{1}{c} x_j(t/c) t \cdot dt + \rho_j \int_0^\infty \tfrac{c-1}{c} y_j((c-1)t/c) t \cdot dt \\ &= \rho_j c \int_0^\infty x_j(t) t \cdot dt + \rho_j \tfrac{c}{c-1} \int_0^\infty y_j(t) t \cdot dt \\ &\leq \rho_j c p_j (C_j^* - \tfrac{p_j}{2}) + \tfrac{c}{c-1} G_j \\ &= c w_j C_j^* - \tfrac{1}{2} w_j p_j + \tfrac{c}{c-1} G_j. \end{aligned}$$

Hence,

$$\textsc{Opt}^M(I') \leq c\textsc{Opt}(I) - \frac{c}{2} \sum_j w_j p_j + \frac{c}{c-1} \sum_j G_j. \tag{3.7}$$

Next, we relate $\textsc{Opt}^M(I')$ to the value $Z(\sigma)$ of the on-line schedule. For this purpose we define another modified instance I''. This instance is formed from I by shifting the release dates to $r''_j = \min\{cr_j, S_j\}$ while keeping the other

parameters intact. We prove that σ is a PREEMPTIVE-WSPT schedule for I'' and hence is optimal with respect to the mean busy date objective. Assume that at time t job J_j is processed, while job J_k is available for σ in the modified instance I''. We have to prove that $J_j \prec J_k$. Let v be the moment at which the segment of J_j that is currently on the machine started processing. If $r_k \leq v$ then clearly $J_j \prec J_k$ since the algorithm gave priority to job J_j at time v. If, on the other hand $r_k > v$, then the length of the remaining segment of job J_j at time r_k was more than $t - r_k \geq r''_k - r_k = (c-1)r_k$. Since the algorithm gave priority to job J_j at time r_k while the remaining processing time of J_j was more than $(c-1)r_k$, the ordering must be $J_j \prec J_k$. Hence,

$$Z^M(\sigma) = \text{OPT}^M(I'').$$

Now we relate the values $\text{OPT}^M(I')$ and $\text{OPT}^M(I'') = Z^M(\sigma)$. Consider the PREEMPTIVE-WSPT schedules of I' and I'', both with the same ρ-ordering on the jobs. We denote the former schedule by σ' and the latter is exactly σ, as we just showed. Notice that a PREEMPTIVE-WSPT schedule can be obtained in an off-line way by scheduling the jobs one by one as early as possible and in the fixed order. By this construction it is easy to see that the inequalities $p'_j \geq p_j$ (for all jobs J_j) imply that for any number $q \in [0, p_j]$, schedule σ' completes the first q processing units of job J'_j not earlier than schedule σ completes the first q processing units of job J_j. More specifically, σ' starts the last p_j processing units of job J'_j not earlier than time $C_j - (p_j - F_j)$. Let $y'(j)$ be the indicator function for σ'.

$$\begin{aligned}
w'_j M'_j &= \rho_j \int_0^\infty y'_j(t) t \cdot dt \\
&\geq \rho_j \int_0^\infty y_j(t) t \cdot dt + \rho_j \int_{C_j - p_j + F_j}^{C_j + F_j} t \cdot dt \\
&= G_j + \rho_j (p_j C_j + p_j F_j - \tfrac{1}{2} p_j^2) \\
&= G_j + w_j C_j + w_j F_j - \tfrac{1}{2} w_j p_j
\end{aligned}$$

Hence,

$$\text{OPT}^M(I') \geq \sum_j G_j + Z(\sigma) + \sum_j w_j F_j - \frac{1}{2} \sum_j w_j p_j. \tag{3.8}$$

Combining the inequalities (3.7) and (3.8) yields

$$Z(\sigma) \leq c\text{OPT}(I) - \frac{c-1}{2} \sum_j w_j p_j + \frac{1}{c-1} \sum_j G_j - \sum_j w_j F_j.$$

To complete the proof it suffices to show that

$$-\frac{c-1}{2}w_j p_j + \frac{1}{c-1}G_j - w_j F_j \leq 0, \quad \text{for all } j \in J. \tag{3.9}$$

Therefore, we consider any job J_j and delete for simplicity the index j in the remainder of the proof. Since job J_j is preempted at time u the remaining processing time $(p-F)$ must be more than $(c-1)u$ implying $u < (p-F)/(c-1)$. We use this inequality to bound G.

$$G \leq \rho \int_{u-F}^{u} t \cdot dt = \rho(uF - \frac{1}{2}F^2) < \frac{\rho(p-F)F}{c-1} - \frac{\rho F^2}{2}.$$

Substituting this in (3.9) we obtain

$$\begin{aligned} -\frac{c-1}{2}wp + \frac{1}{c-1}G - wF &< -\frac{c-1}{2}wp + \frac{\rho(p-F)F}{(c-1)^2} - \frac{\rho F^2}{2(c-1)} - F \\ &= -\frac{\rho(c+1)}{2(c-1)^2}F^2 - \frac{(c^2-2c)w}{(c-1)^2}F - \frac{(c-1)wp}{2} \\ &= wp\left(-\frac{(c+1)}{2(c-1)^2}H^2 - \frac{c^2-2c}{(c-1)^2}H - \frac{(c-1)}{2}\right). \end{aligned}$$

We substituted $F = pH$ in the last equality. Standard calculus shows that this expression is non-negative for any $H \in \mathbb{R}$ if $2c^3 - 4c^2 + 2c - 1 \geq 0$. The real root of this inequality is $1.5651\ldots$. □

We can extend Delayed-Preemptive-WSPT$[c]$ to identical parallel machines as has been done for the non-preemptive problem. We say that machine i and job J_j are blocked at time t if job J_j is processed on i at time t and its remaining processing time is at most $(c-1)t$.

Delayed-Preemptive-WSPT$[c]$:
Maintain a ρ-ordering of the jobs. At any moment t apply the following rule. If k is the number of machines that are not blocked at time t, then process on those machines the k highest priority jobs among the available, but not blocked, jobs.

The single machine proof extends almost directly to identical parallel machines: Delayed-Preemptive-WSPT is 1.56γ-competitive on identical parallel machines, where γ is the competitive ratio of Preemptive-WSPT on identical parallel machines. By Proposition 3, the best we can hope for is to prove

$1.56 \cdot 5/4 = 1.95$-competitiveness. This is not very tempting since SWRPT is 2-competitive and its best known lower bound is 1.16.

3.4 Remarks and open problems

The main challenge in on-line scheduling to minimize total (weighted) completion time, with jobs arriving over time, is to find provably good algorithms for identical parallel machines. Recently, Megow and Schulz [63] showed that Delayed-WSPT is 3.28-competitive for the non-preemptive problem. Unfortunately, their algorithm cannot be better than 2.61-competitive. A large gap with the best known general lower bound of 1.309 given by Vestjens [86] still remains. We conjecture that the competitive ratio of the following algorithm is strictly smaller than 2.

λ-Delayed-WSPT:
Schedule the highest priority job J_j at the earliest moment t at which a machine is idle and the total remaining processing time is at most $\lambda mt - p_j$, provided that no new job is released before time t. If a new job is released at a time $t' \leq t$, then repeat the rule at time t' with the new set of jobs.

This algorithm corresponds to the 2-competitive Delayed-WSPT algorithm on a single machine if we choose $\lambda = 1$. A good choice for multiple machines seems $\lambda = 0.5$. A lower bound of $1+\lambda$ follows from a construction similar to the proof of Lemma 2. An alternative algorithm is obtained if at any moment the k highest priority jobs are considered, where k is the number of idle machines at that moment.

For preemptive scheduling on identical parallel machines Vestjens [86] proved that no deterministic algorithm can be better than 1.047-competitve. No relevant lower bound is know for randomized algorithms and the best upper bounnd is 2 for any of the parallel-machine problems: preemptive or non-preemptive and weighted or unweighted [76].

We conjecture that the competitive ratio for any of the problems mentioned above is close to the best known lower bounds on these ratios that we mentioned. A simple deterministic 1.5-competitive algorithm for the non-preemptive problem on parallel machines, and algorithms with much smaller ratios for the other variants are plausible to exist. The existing lower bounds on the optimal value do not seem to be strong enough to prove such small ratios. We presented three properties of parallel machine schedules that we conjecture to ensure (near)-optimality and we expressed the competitive ratios of the considered algorithms in terms of these numbers α,β and γ. It is surprisingly difficult to prove good upper bounds on these numbers. A good understanding of these problems seems

essential to prove small competitive ratios.

The *restart* model was not considered in this chapter. In this setting a job can be preempted at any moment, but if preempted its processing has to start all over again. Van Stee and La Poutré [83] claimed a simple 1.5-competitive algorithm for the unweighted single-machine problem. A lower bound of 1.211 for the single machine is given by Epstein and Van Stee [30]. Despite its simplicity there seems to be no easy generalization of the algorithm to the weighted single-machine problem. The difficulty is that, in the weighted problem, the decision of whether or not to preempt a job in favor of another job, cannot be based on the parameters of these two jobs only. One has to consider the value of a set of jobs. Clearly, any c-competitive algorithm for the non-preemptive problem is c-competitive for the restart problem as well. Hence, on identical parallel machines the non-preemptive problem provides an upper bound of 4. This is the best known so far and no lower bound is known.

Minimizing the total mean busy date on identical parallel machines seems an interesting problem on its own. We showed a lower bound of 5/4 for the PREEMPTIVE-WSPT algorithm but the competitive ratio of the problem itself might be much smaller.

Chapter 4

Complexity of the traveling repairman problem

4.1 Introduction

Given n points $v_1, \ldots, v_n$ and an integer distance between any pair of points, the *traveling repairman problem* is to find a tour π, starting at the *origin* v_1 and visiting all points, for which the sum of the arrival times $d_\pi(v_1, v_i)$ is minimum, where the arrival time is the traveled distance from v_1 to v_i on tour π.

We can think of a repairman that needs to repair machines located at the given points. We assume that the repairman travels at unit speed and that the repair times are negligible in comparison with the travel time, i.e. all repair times are zero. The repairman's objective is to minimize the sum of the arrival time, or equivalently, the sum of completion times of the repairs. The traveling repairman problem has been well-studied in operations research, where it is also known as the *delivery man problem* and in computer science, where it is often called the *minimum latency problem*.

Unlike the traveling salesman problem, where the objective is minimizing maximum arrival time and therefore is server oriented, TRP is client oriented, with objective minimizing the average time that a machine waits for having been repaired. An equivalent objective is minimizing *average* completion time: total completion time divided by the number of points. We can also interpret this objective as minimizing the average number of machines that waits for service, or equivalently, as maximizing the average number of machines that are operating.

A problem almost equivalent to TRP is the *graph searching problem* (GSP) (introduced in [52]). We assume that the points are vertices of a graph and the metric is induced by the shortest paths in the graph. There is an object

hidden in one of the vertices of the graph. A distribution specifies for each of the vertices the probability that the object is hidden there. The objective is to find a tour that minimizes the expected search time for the object. Clearly, GSP is polynomially equivalent to TRP under mild conditions on the numbers in the instances.

The traveling repairman problem was proven to be NP-hard for general metric spaces (METRIC-TRP) by Sahni and Gonzalez [72]. In fact both the traveling salesman problem and TRP are $Max\text{-}SNP$-hard for general metric spaces, but the traveling repairman problem has a reputation for being much harder than the traveling salesman problem. A 7.18-approximation algorithm follows from a result of Goemans and Kleinberg [41] combined with the approximation algorithm for the k-MST problem by Arora and Karakostas [6]. We refer to the papers of Goemans and Kleinberg [41], Arora and Karakostas [7], Ausiello et al.[10], Archer and Williamson [4], and de Paepe et al. [66] for an overview of TRP and related problems.

In this chapter we prove NP-hardness for two special cases of the traveling repairman problem: LINE-TRP with release dates and TREE-TRP. The latter result appeared before in [80]. We define an instance of LINE-TRP as an edge-weighted path, where one of the vertices is labelled as the origin. An instance of TREE-TRP is a tree with root r and weights on the edges. The origin is the root of the tree. For both problems the set of request points is the set of vertices. The distance between two vertices is the length of the path between them. We also consider problems with weights w_j on the vertices. The objective in that case is to minimize $\sum_j w_j C_j$. Notice that TRP on the path and TRP on the tree, with weights on the edges and vertices, polynomially reduce to respectively LINE-TRP and TREE-TRP if all vertex weights are polynomially bounded in the number of vertices. We will use this in our NP-hardness proofs. In the sequel we will sometimes write *edge length* for the weight of an edge and *total completion time* for the sum of the weighted completion times.

4.2 An exact algorithm

We show a simple $O(n^2 2^n)$ *dynamic program* (D.P.) that solves TRP for any metric space. It is a generalization of the algorithm for the LINE-TRP and is similar to the exact algorithm for TSP given in [47]. Each pair (S, s), with S a subset of the request points and $s \in S$, defines a state of the tour. The set S represents all requests served sofar and the last request served is s. Let cost f of a state (S, s) be the minimum over all tours Π of $\sum_{v_i \in S} d_\Pi(v_1, v_i) + (n - |S|) d_\Pi(v_1, s)$. In other words, $f(S, s)$ is the minimum possible total delay accumulated by all requests during the first part of the tour, i.e. until serving

s. Then $f(S, s)$ satisfies the following equation, where $|S| \geq 2$:

$$f(S, s) = \min_{t \in S \setminus \{s\}} f(S \setminus \{s\}, t) + (n - |S| + 1)d(s, t) \tag{4.1}$$

Since there are $O(n2^n)$ states and each value of a state is a function of at most $n-1$ preceding state values, the dynamic programming can be done in $O(n^2 2^n)$ steps.

4.3 The traveling repairman problem on the line with release dates

Afrati et al. [2] show that Line-TRP can be solved in $O(n^2)$ time by dynamic programming. The key observation here is that, if a vertex v is served, then also all vertices on the path from the origin to v are served. Hence, we have to consider only $O(n^2)$ possibilities for the set S in the dynamic program (4.1). Notice also that the last point served in any such set must be one of the two extreme vertices. Implementation of these two observations in the dynamic program reduces the running time to $O(n^2)$.

Suppose now that each vertex has a given release date and we impose the additional condition that a vertex cannot be served before its release date. Neither of the observations above holds true in the presence of release dates. In fact, for any path we can choose the release dates in such a way that the optimal tour will serve the vertices in an arbitrarily chosen order. Tsitsiklis [84] mentioned that the complexity of this problem was unknown and de Paepe et al. [66] consider it as one of the most challenging open problems in their classification of dial-a-ride problems.

By a reduction from the Partition problem we show that the Line-TRP with release dates is ordinarily NP-hard. NP-hardness of Partition was proven by Karp [50].

Partition

Instance: A multiset of natural numbers $\{p_1, p_2 \ldots, p_n\}$, with $p \leq p_i \leq 2p$ for some number p and every $i \in \{1, \ldots, n\}$.
Question: Is there a perfect partition of $\{p_1, p_2 \ldots, p_n\}$, i.e., is there a set $A \subset \{1, \ldots, n\}$ such that $\sum_{i \in A} p_i = \frac{1}{2} \sum_{i=1}^{n} p_i$?

Theorem 8 Line-TRP *with release dates is ordinarily NP-hard.*

Proof. For every instance $I1$ of the Partition problem we define an instance $I2$ of the traveling repairman problem on a path with weights on the edges and

vertices. We claim that there exists a perfect partition for $I1$ if and only if there exists a solution with cost at most $C - P/2$ for $I2$, where $P = \sum_{i=1}^{n} p_i$ and C is a function of $I2$ which we be specify below.

$I2$ consists of the path $v_{11}, v_{21}, \ldots, v_{n1}, O, v_{12}, v_{13}, v_{22}, v_{23}, \ldots, v_{n2}, v_{n3}, z$, where O is the origin. We define the numbers $M = 10n^2$ and $l_i = (n - i + 1)(2M + 1)p_i + p_i/2$ $(i = 1 \ldots n)$. The lengths of the edges are:

$$
\begin{array}{ll}
d(v_{i1}, v_{i+1,1}) = l_i - l_{i+1} + p_i, & (i = 1, \ldots, n-1); \\
d(v_{n1}, O) = l_n - (p_1 + \ldots + p_{n-1}); & \\
d(O, v_{12}) = p_1; & \\
d(v_{i2}, v_{i3}) = 0, & (i = 1, \ldots, n); \\
d(v_{i3}, v_{i+1,2}) = p_{i+1}, & (i = 1, \ldots, n-1); \\
d(v_{n,3}, z) = 0. &
\end{array}
$$

The weights on the vertices are:

$$
\begin{array}{ll}
w(v_{i1}) = w(v_{i2}) = M, & (i = 1, \ldots, n); \\
w(v_{i3}) = w(v_z) = 1, & (i = 1, \ldots, n).
\end{array}
$$

We denote by r_{ij} (r_z) the release date of vertex v_{ij} (z).

$$
\begin{array}{rcll}
r_{13} & = & 0, & \\
r_{i1} & = & r_{i3} + l_i & (i = 1 \ldots n), \\
r_{i2} & = & r_{i3} + 2l_i + p_i & (i = 1 \ldots n), \\
r_{i+1,3} & = & r_{i2} & (i = 1 \ldots n-1), \\
r_z & = & r_{n2} + P. &
\end{array}
$$

Let tour T serve the vertices in the order $O, v_{11}, v_{12}, v_{13}, v_{21}, \ldots, v_{n3}, z$. Notice that $d(O, v_{11}) = l_1$ and $d(v_{i3}, v_{i+1,1}) = l_{i+1}$ for $i = 1, \ldots, n-1$. The vertices v_{i1} and v_{i2} $(i = 1, \ldots, n)$ are served exactly at their release dates. Every vertex v_{i3} is served $2l_i + p_i$ time units after its release date and the tour arrives in z at P time units before its release date. We define C as the total total weighted completion time of this solution T.

For any set $A \subseteq \{1, \ldots, n\}$ we define a solution T_A for $I2$ as follows. We follow tour T defined above, but if $i \in A$ then v_{i3} is served directly after $v_{i-1,3}$ is served and we continu on T. Notice that by this definition $T = T_\emptyset$. We claim that the total weighted completion time C_A of tour T_A with $A \subseteq \{1, \ldots, n\}$ is exactly $C - P/2 + |P/2 - \sum_{i \in A} p_i|$.

Let $i \in A$. The completion time of v_{i3} in T_A is $2l_i$ time units smaller than its completion time in $T_{A \setminus \{i\}}$. On the other hand the completion in T_A of any vertex served after v_{i3} is $2p_i$ time units larger than its completion time in $T_{A \setminus \{i\}}$. An exception is the vertex z, which is not delayed if the total delay $\sum_{i \in A} 2p_i$ is

less than P. Summarizing we have

$$\begin{aligned} C_A &= C + \sum_{i \in A}(2p_i(n-i+1)(2M+1) - 2l_i) + \max\{0, \sum_{i \in A} 2p_i - P\} \\ &= C - \sum_{i \in A} p_i + \max\{0, \sum_{i \in A} 2p_i - P\} \qquad (4.2) \\ &= C - P/2 + |P/2 - \sum_{i \in A} p_i| \ . \end{aligned}$$

If there exists a perfect partition A, then $C_A = C - P/2$.

If there is no perfect partition for $I1$, then any tour T_A has total weighted completion time at least $C_A = C - P/2 + 1$, assuming P is even. It remains to show that in this case any tour has value at least $C - P/2 + 1$.

First, we roughly bound C from above. Let R be the sum of weighted release dates.

$$\begin{aligned} C &= R + \sum_{i=1}^{n}(2l_i + p_i) \\ &= R + \sum_{i=1}^{n} 2(n-i+1)(2M+1)p_i + p_i \\ &\leq 2n(n+1)(2M+1)2u + 2p \\ &= (4n(n+1)(20n^2+1) + 2)p. \end{aligned}$$

For $n \geq 1$ we can roughly bound this by $C < 200n^4p$. Consider the order in which the vertices with weight M are served in solution T. If this order is different in a solution T', then at least one of these is served not earlier than $\min\{l_i | i = 1...n\} > 2Mp$ after its release date. Hence, the total weighted completion time of such solution T' is more than $R + 2M^2p = 200n^4p > C$. Similarly, if z is served before v_{n1}, then the total weighted completion time also exceeds C. Therefore, in the optimal solution the vertices with weight M are visited in the order $v_{11}, v_{12}, v_{21}, \ldots, v_{n2}, z$, and z is served after v_{n3}. Under this condition, it is impossible that a vertex v_{i3} is served before $v_{i-1,2}$, since otherwise there must be a vertex with weight M that is released before $v_{i-1,2}$ but served after $v_{i-1,2}$. Now there remain two possible ways to serve a vertex v_{i3}. Either it is served between $v_{i-1,2}$ and v_{i1}, or it is served together with v_{i2}. This leaves exactly the tours of the form T_A for which we already proved that the value is at least $C - P/2 + 1$ if no perfect partition exists. □

It remains an open problem whether LINE-TRP with release dates is pseudo-polynomially solvable or NP-hard in the strong sense. LINE-TRP with deadline constraints is ordinarily NP-hard and pseudo-polynomially solvable by dynamic programming [2]. If we follow an optimal tour, then, at any moment, the visited vertices are neighboring vertices. In addition to the dynamic programming for

the unrestricted LINE-TRP, we need a time index. This makes the running time of the dynamic programming pseudo-polynomial.

The complexity of the problem where, instead of release dates or deadlines, there is a given *repair time* p_j for each request s_j, is an open problem for the line metric [2, 84]. In this problem the server must stay in the request point for a time p_j to serve request s_j. We can formulate this problem as a special case of TREE-TRP by adding a vertex at a distance of half the repair time at any request point on the path. Such trees are called *caterpillars*. Koutsoupias et al. [52] conjecture that TRP is NP-hard on caterpillars.

Tsitsiklis [84] observed that strong NP-hardness of LINE-TRP with both release dates and repair times follows directly from strong NP-hardness of the single machine scheduling problem with releases dates and minimizing total completion time. In fact, TRP with these two constraints is even strongly NP-hard if the metric space is a single point.

LINE-TRP with repair times becomes easy if we add the restriction that a request must be served at the moment that the server passes the request point. We can use the same dynamic program as for the unrestricted LINE-TRP to obtain an $O(n^2)$ algorithm.

Recently García et al. [34] gave a linear time algorithm for LINE-TRP. They use the concepts of Monge matrices to improve the efficiency of the dynamic program. An $m \times n$ matrix A is called Monge if $a_{ij} + a_{i+1,j+1} \leq a_{i+1,j} + a_{i,j+1}$, $\forall 1 \leq i \leq m, 1 \leq j \leq n$. This Monge property has been used before in several optimization problems to improve the running time. García et al. use the same technique to solve the problem of finding a feasible solution for LINE-TRP with deadlines in $O(n)$ time, for which an $O(n^2)$ algorithm was given in [2].

4.4 The traveling repairman problem on a tree

Classifying the complexity of TRP for edge-weighted trees has been mentioned as an open problem in many papers [6, 10, 11, 12, 41, 52, 65, 87, 88]. For example Goemans and Kleinberg [41] write that 'the TRP is not known to be NP-hard on weighted trees, so it is worth considering whether it could be solved optimally'. Notice that the traveling salesman problem is trivial on weighted trees.

For some special cases an exact polynomial time algorithm is known. If the tree is unweighted, then a tour is optimal if and only if it is a depth-first search. Proofs have been given by several authors [65, 12]. The dynamic programming solution for LINE-TRP can be extended to an $O(n^k)$ dynamic program for TREE-TRP [52], where k is the number of leaves. This was also observed by Minieka [65] who formulated the TREE-TRP as a shortest path problem in a network with $O(n^{k+1})$ nodes. Blum et al. [12] show that dynamic programming

gives an $O(n^2)$ algorithm for TREE-TRP on trees with combinatorial diameter at most 3. These trees consist of a central edge with leaves on its two end points. The observation here is that the leaves at either end are visited in increasing order of their length. Wu [88] generalizes this to an exact algorithm for graphs with a bounded number of internal nodes. We prove that no exact polynomial time algorithm exists for the TREE-TRP, unless $P = NP$.

We give a preliminary lemma. Given an instance of TRP on a tree, let T be an optimal tour and let $t_0 = 0, t_1, \ldots, t_k$ be the moments at which the server is in the origin. Without loss of generality we assume that the tour ends at time t_k in the origin. Let T_i be the subtour between time t_{i-1} and t_i, $1 \leq i \leq k$, and let $|T_i|$ and W_i be respectively the length of this subtour and the total weight of the vertices that are served on T_i. Then we have the following lemma.

Lemma 5 *For any optimal tour T the following holds.*

(i) $\frac{W_1}{|T_1|} \geq \frac{W_2}{|T_2|} \geq \ldots \geq \frac{W_k}{|T_k|}$.

(ii) If $\frac{W_i}{|T_i|} = \frac{W_j}{|T_j|}$ *for some* $i, j \in \{1, \ldots, k\}$*, then the total completion time remains the same if the subtours T_i and T_j swap their position on T.*

PROOF. (i) We use a simple interchange argument. Assume that for some i we have $\frac{W_i}{|T_i|} < \frac{W_{i+1}}{|T_{i+1}|}$. If we change the order of the subtours T_i and T_{i+1}, then the increase in the total completion time is $W_i|T_{i+1}| - W_{i+1}|T_i| < 0$. (ii) follows directly from the proof of (i). □

To prove NP-hardness for TREE-TRP we give a reduction from the 3-PARTITION problem, which was proven to be NP-hard in the strong sense by Garey and Johnson [35].

3-PARTITION

Instance: A multiset of natural numbers $\{p_1, p_2 \ldots, p_{3n}\}$, with $P/4 < p_i < P/2$ for all $i \in \{1, \ldots, 3n\}$, and a number P such that $p_1 + \cdots + p_{3n} = nP$.
Question: Is it possible to partition the index set $\{1, \ldots, 3n\}$ in n sets $A_1, \ldots, A_n$ such that $\sum_{i \in A_j} p_i = P$ for all $j \in \{1, \ldots, n\}$?

Theorem 9 *The traveling repairman problem is strongly NP-hard on edge-weighted trees.*

PROOF. Given an instance of 3-PARTITION with the notation as described above, we define $a_i = Kp_i$ for all $i \in \{1, \ldots, 3n\}$ and $Q = KP$, where $K = 2Pn^4$. We define a tree on $3n(n+2)+1$ vertices as follows.

For each $i \in \{1, \ldots, 3n\}$ we construct a path $(r, v_{i1}, v_{i2}, \ldots, v_{in}, z_i)$. All these paths start in the root of the tree, which is appointed as the origin of the TRP-instance. To each of these paths an extra vertex u_i is attached through the edge (v_{i1}, u_i) (see Figure 4.4). For the definition of the lengths of the edges in this tree we introduce the numbers m and l_i, $(i = \{1, \ldots, 3n\})$, and choose their value appropriately later. The lengths of the edges are:

$$\begin{array}{ll} d(r, v_{i1}) = l_i, & (i = 1, \ldots, 3n); \\ d(v_{ij}, v_{i,j+1}) = a_i, & (i = 1, \ldots, 3n, j = 1, \ldots, n-1); \\ d(v_{i1}, u_i) = 2Qa_i, & (i = 1, \ldots, 3n); \\ d(v_{in}, z_i) = m, & (i = 1, \ldots, 3n). \end{array}$$

The weights on the vertices are:

$$\begin{array}{ll} w(v_{i1}) = a_i, & (i = 1, \ldots, 3n); \\ w(v_{ij}) = n - j + 1, & (i = 1, \ldots, 3n, j = 2, \ldots, n); \\ w(z_i) = a_i, & (i = 1, \ldots, 3n); \\ w(u_i) = 1, & (i = 1, \ldots, 3n). \end{array}$$

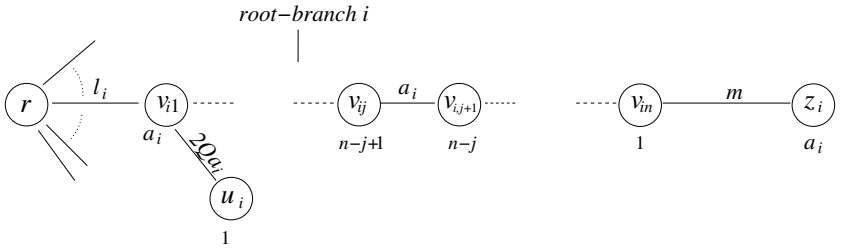

Figure 4.1: Sketch of the TRP instance.

We call the subtree rooted at r and constituted by the path $(r, v_{i1}, \ldots, v_{in}, z_i)$ and the edge (v_{i1}, u_i) the *root-branch* i, $(i = 1, \ldots, 3n)$.
It is easy to see that the values m and l_i can be chosen appropriately such that an optimal tour satisfies the following properties (a) and (b) for all $i \in \{1, \ldots, 3n\}$:

(a) each edge (r, v_{i1}) is traversed exactly once in each direction;

(b) vertex u_i is visited before vertex z_i.

Moreover, we choose l_i, $(i = 1, \ldots, 3n)$, such that for all i and h in $\{1, \ldots, 3n\}$

(c) $\frac{W_i}{L_i} = \frac{W_h}{L_h}$, where W_i and L_i denote, respectively, the sum of all vertex-weights and the sum of all edge-lengths of root-branch i.

With respect to (c) we note that choosing $l_i = M(2a_i + n(n-1)/2 + 1) - (2Q + n - 1)a_i - m$ yields $\frac{W_i}{L_i} = M$, for all $i \in \{1, \ldots, 3n\}$, where M is a large number.

We prove later that the numbers m and M can be bounded by a polynomial in the size of the 3-PARTITION instance.

Consider an optimal tour T. By (a) and (b) there are at most n different ways for this tour to traverse root-branch i: for $k = 1, \ldots, n$, we call the subtour that visits vertex u_i directly after vertex v_{ik} the k-tour. Lemma 5(i) implies that in T, all k-tours precede all $k+1$-tours ($k = 1, \ldots, n-1$). Renumber the root-branches such that root-branches $1, \ldots, i_1$ are traversed by a 1-tour, $i_1+1, \ldots, i_2$ by a 2-tour, etc. until $i_{n-1}+1, \ldots, i_n = 3n$ being the root-branches traversed by an n-tour.

We compare the optimal tour T with the tour in which all root-branches are traversed by a 1-tour. Applying Lemma 5(ii) and (c) we may assume that the order in which the root-branches are served is the same for both tours.

Let us investigate the gains and losses in objective value of serving root-branch i by a k-tour instead of by a 1-tour. The gains come from a reduction of $4Qa_i$ in the completion time of each of the vertices $v_{i2}, \ldots, v_{ik}$. The losses come from a delay of $2(k-1)a_i$ in visiting the vertices $v_{i,k+1}, \ldots . v_{in}, u_i, z_i$ and all the vertices on the root-branches $h \geq i+1$.

Thus the total gain in objective value by serving according to T instead of the all 1-tour is

$$\sum_{k=2}^{n} \sum_{i=i_{k-1}+1}^{3n} 4(n-k+1)Qa_i = \sum_{k=2}^{n} \left(4(n-k+1)Q \sum_{i=i_{k-1}+1}^{3n} a_i \right). \qquad (4.3)$$

Expressing the total loss we first give a crude bound on R^T the loss due to delay of all vertices except the vertices z_i and v_{i1}, $i = 1, \ldots, 3n$. There are $3n^2$ of them and their total weight is $3n(\frac{1}{2}n(n-1)+1) < 2n^3$. The delay for each of them is at most $2(n-1)Q$. Therefore, $R^T < 4Qn^4$.

To express the total loss due to delay of the vertices v_{i1} and z_i, $i = 1, \ldots, 3n$, consider a root-branch i served by a k-tour, i.e., $i_k \leq i < i_{k+1}$. Compared to serving it by a 1-tour, the vertex v_{i1} is not delayed, but, as noted before, the vertex z_i is delayed by an amount $2(k-1)a_i$, which multiplied by its weight gives a loss in objective value of $2(k-1)a_i^2$. Moreover, for all $h > i$ the vertices v_{h1} and z_h are delayed by $2(k-1)a_i$. Thus, the total loss by serving root-branch i by the k-tour i.o. the 1-tour due to a delay of vertices v_{h1} and z_h, $h = i, \ldots, 3n$ amounts to

$$2(k-1)a_i(a_i + \sum_{h=i+1}^{3n} 2a_h).$$

The total loss in objective value by serving according to tour T instead of the all 1-tour is therefore given by

$$R^T + \sum_{k=2}^{n} \sum_{i=i_{k-1}+1}^{i_k} 2(k-1)a_i(a_i + \sum_{h=i+1}^{3n} 2a_h) = R^T + \sum_{k=2}^{n} 2(\sum_{i=i_{k-1}+1}^{3n} a_i)^2. \quad (4.4)$$

Let C^T and C be the total completion times, respectively, for T and the all 1-tour. Combining (4.3) and (4.4) we obtain

$$C^T = C + R^T + \sum_{k=2}^{n} \left(2(\sum_{i=i_{k-1}+1}^{3n} a_i)^2 - 4Q(n-k+1) \sum_{i=i_{k-1}+1}^{3n} a_i \right). \qquad (4.5)$$

Writing $b_k = \sum_{i=i_{k-1}+1}^{3n} a_i$, $k = 2, \ldots, n$, (4.5) becomes

$$C^T = C + R^T + \sum_{k=2}^{n} (2b_k^2 - 4Q(n-k+1)b_k). \qquad (4.6)$$

Each of the terms $2b_k^2 - 4Q(n-k+1)b_k$, $k = 2, \ldots n$, attains its minimum value $-2Q^2(n-k+1)^2$ at $b_k = (n-k+1)Q$.

Assume the 3-PARTITION instance has a yes-answer and let $A_1, \ldots, A_n$ be a perfect partition. Consider the tour $\widetilde{T}$ which traverses the 3 root-branches that have their index in A_k by a k-tour, and all k-tours precede all $k+1$-tours, for every $k \in \{1, \ldots, n\}$. We define $\widetilde{b}_k = \sum_{i \in A_k \cup \ldots \cup A_n} a_i$ implying $\widetilde{b}_k = (n-k+1)Q$ for all $k \in \{2, \ldots, n\}$. Thus, by (4.6):

$$\begin{aligned} C^T \leq C^{\widetilde{T}} &= C + R^{\widetilde{T}} + \sum_{k=2}^{n} (2\widetilde{b}_k^2 - 4Q(n-k+1)\widetilde{b}_k) \\ &= C + R^{\widetilde{T}} - 2Q^2 \sum_{k=2}^{n} (n-k+1)^2 \\ &= C + R^{\widetilde{T}} - \tfrac{1}{3}Q^2 n(n-1)(2n-1) \\ &< C + 4Qn^4 - \tfrac{1}{3}Q^2 n(n-1)(2n-1) \\ &= C + 2K^2 - \tfrac{1}{3}Q^2 n(n-1)(2n-1), \end{aligned}$$

using $Q = KP$ and $K = 2Pn^4$ for the last equality.

If no 3-partition exists then there must be a $j \in \{2, \ldots, n\}$ such that $|b_j - (n-j+1)Q| \geq K$. (Remember that a_k is a multiple of K for all k.) Standard calculus tells us that

$$2b_j^2 - 4Q(n-j+1)b_j \geq -Q^2(n-j+1)^2 + 2K^2,$$

implying

$$\begin{aligned} C^T &= C + R^T + \sum_{k=2}^{n} (2b_k^2 - 4Q(n-k+1)b_k) \\ &\geq C + R^T - \tfrac{1}{3}Q^2 n(n-1)(2n-1) + 2K^2 \\ &\geq C - \tfrac{1}{3}Q^2 n(n-1)(2n-1) + 2K^2. \end{aligned}$$

We conclude that the 3-PARTITION instance has a yes-answer if and only if

$$C^T < C - \frac{1}{3}Q^2 n(n-1)(2n-1) + 2K^2.$$

It remains to show that the numbers m and M can be bounded by a polynomial in the size of the 3-PARTITION instance, and still satisfy the conditions that each edge (r, v_{i1}) is traversed exactly once in each direction, and that vertex u_i is visited before vertex z_i.

Consider a tour T_1 that traverses the edge (r, v_{i1}) more than once in each direction. We adjust this tour to a tour T_2 that passes (r, v_{i1}) only once in each direction. When tour T_1 visits vertex v_{i1} for the first time we continue this tour by visiting all the remaining vertices of root-branch i in the same order as they are visited on T_1. Next, we continue T_1.

We will show that T_2 has smaller total completion time than T_1 if M is chosen large enough. For any vertex, its completion time on T_2 is at most $2((2n-2+2Q)a_i+m)$ larger than its completion time on T_1. On the other hand, for at least one of the vertices of root-branch i, its completion time is at least $2l_i$ lower than its completion time on T_1. Therefore, the total completion time is at least $2l_i - 2((2n-2+2Q)a_i+m)W$ lower than the total completion time of T_1, where $W = \frac{3}{2}n^2(n-1) + 3n + 2nQ$ is the total weight of the tree. Now it easy to see that we can choose M such that $l_i > ((2n-2+2Q)a_i+m)W$ for all $i \in \{1, \ldots, 3n\}$ and to do so M can be kept polynomially bounded in the size of the 3-PARTITION instance and in m.

Now consider a tour T_1 that traverses the edges (r, v_{i1}) $(i = 1 \ldots 3n)$ exactly once in each direction but vertex z_h is visited before vertex u_h for some $h \in \{1, \ldots, 3n\}$. We let tour T_2 be similar to T_1 with the exception that root-branch h is traversed as a 1-tour. The completion time is unchanged for any vertex that is not in root-branch h. The completion time of vertex u_h in T' is exactly $2(m + (n-1)a_h)$ lower than its completion time in T_1. The completion time in T_2 of the vertices $v_{i2}, \ldots, v_{in}, z_i$ is exactly $4Qa_h$ larger than their completion time in T_1. Therefore, the total completion time of T' is $2(m + (n-1)a_i) - 4Qa_i(n(n-1)/2 + a_i)$ smaller than the total completion time of T_1. Now it easy to see that we can choose m such that the total completion of T_2 is smaller than the total completion time of T_1 and keep m polynomially bounded in the size of the 3-PARTITION instance. □

In the proof of Theorem 9 we used weights on the vertices and interpreted a vertex with weight w as w unit-weight vertices at a distance zero from one another. This might seem a deceptive way to get rid of the vertex weights in the NP-hardness statement. If we replace a vertex with weight w by w unit-weight vertices at a distance ϵ from one another and choose ϵ small enough, then the proof works as well. Hence, Theorem 9 applies as well if we disallow edge lengths to be zero.

Corollary 3 *TRP is strongly NP-hard for edge-weighted trees where all weights are integers larger than or equal to 1.*

If we do allow zero-length edges, then we can strengthen Theorem 9 in the sense that NP-hardness holds even if all edge-lengths are either zero or one. This is equivalent to the next theorem.

Theorem 10 *TRP is strongly NP-hard for vertex-weighted trees.*

PROOF. We reduce from edge-weighted trees. Let I be an arbitrary instance of TREE-TRP with vertices $v_1, \ldots, v_n$ and edges $e_1, \ldots, e_m$ and with integer l_j being the length of edge e_j. From I we define an instance I' of TREE-TRP with all vertex weights 1 and edge-lengths 0 and 1.

We give each vertex v_i $(i = 1, \ldots, n)$ a large weight K, which we choose appropriately later. For any edge e_j with length $l_j \geq 1$ incident to vertices u and v we insert $l_j - 1$ extra vertices $v_{j1}, \ldots, v_{j,l_j-1}$ and the path $u, v_{j1}, \ldots, v_{j,l_j-1}, v$ containing l_j edges of length 1. The inserted vertices receive weight 1. Choose $K > 2L(n + L - m)^2$, with L the sum of the lengths of all edges of I. We claim that, for any positive integer C, there exists a solution for I with total completion time $C(I) < C$ if and only if there exists a solution for I' with total completion time $C(I') < KC$. This claim proves the theorem.

If there exists a tour T' for I' with $C(I') < KC$, then the tour for I that follows from T' in the obvious way has total completion time less than C. On the other hand, if there exists a tour T for I with $C(I) \leq C - 1$, then the tour T' for I' that follows from T in the obvious way has objective $K(C - 1) + D$, where D is the total completion time of the inserted vertices. It suffices to show that $D < K$.

The number of vertices of I' is $n + L - m$. Notice that $2L(n + L - m)$ is an upper bound on the length of the tour that returns to the origin every time a new vertex has been visited, which in its turn is clearly an upper bound on the length of T'. Therefore, $D < 2L(n + L - m)^2 < K$. □

4.5 Approximation algorithms

Max-SNP-hardness of METRIC-TRP follows directly from Max-SNP-hardness of TSP for general metric spaces [68]. In the reduction from METRIC-TSP we simply add a lot of requests to the requests in the TSP instance, and make sure that the optimal TRP-tour serves the added requests last. This simple reduction can be made for almost any metric space, which supports the general idea that TRP is more difficult than TSP.

It is unlikely that TREE-TRP is Max-SNP-hard since Arora and Karakostas gave a so called quasi-polynomial time approximation scheme with running time $n^{O(\log n/\epsilon)}$ [7]. Hence, for any fixed ϵ there is an ϵ-approximation with $n^{O(\log n)}$

running time. $Max\text{-}SNP$-hardness of TREE-TRP implies that there exists a number $\epsilon > 1$ such that approximating TREE-TRP within a factor ϵ is an NP-hard problem. It is generally believed that NP-hard problems cannot be solved in $n^{O(\log n)}$ time.

As mentioned before, TREE-TRP is easily solved if the edges are unweighted. For general unweighted graphs the problem is $Max\text{-}SNP$-hard, which follows directly from $Max\text{-}SNP$-hardness of the Hamiltonian path problem. Koutsoupias et al. [52] give a 1.66-approximation algorithm, based on Christofides' algorithm for TSP. In addition, a 2-approximate solution is easily obtained by traversing an arbitrary spanning tree in depth-first order.

The first constant-factor approximation algorithm for METRIC-TRP was a 144-approximation algorithm given by Blum et al. [12]. An improved version was given by Goemans and Kleinberg [41]. We briefly describe their algorithm.

Let T_k $(k = 1, 2, \ldots, n)$ be a minimum k-TSP, i.e. a shortest tour, from origin to origin, that contains at least k request points. Now consider the problem of minimizing total completion time under the restriction that the tour must be a concatenation of minimum k-TSPs. Goemans and Kleinberg prove that the optimal value of this problem is at most 3.59 times the optimal value. Additionally they show how an optimal solution for this problem can be find efficiently if the k-TSPs are given. The k-TSP problem is NP-hard on a general metric space but can be solved efficiently for the tree metric [12], yielding a 3.59-approximation algorithm for TREE-TRP. For Euclidean spaces with bounded dimension Arora's PTAS [5] yields a $3.59 + \epsilon$-approximation algorithm. For an arbitrary metric space Garg [38] gave a 3-approximation for k-TSP, and a $2 + \epsilon$-approximation algorithm was given by Arora and Karakostas [6], which leads to a $7.18 + \epsilon$-approximation for METRIC-TRP. Instead of a k-TSP, we could also consider k-MST, i.e., a tree rooted in the origin, with at least k vertices and with a minimum sum of edge weights. The approximation guarantee that follows from the algorithm of Goemans and Kleinberg remains the same.

Recently, Archer and Williamson [4] gave a 9.28-approximation algorithm for METRIC-TRP with a much smaller running time than the 7.18-approximation algorithm. Their basic idea is that they do not use k-MST as a black box. Instead, they show that their k-MST algorithm, produces relatively small solutions for some (uncontrolled) values of k. Subsequently, they exploit this property in the choice for the trees that are concatenated.

The current best approximation ratio for METRIC-TRP is 3.59. This algorithm, given by Chaudhuri et al [20], elaborates on the algorithm of Goemans and Kleinberg.

Fakcharoenphol et al. [31] generalized the algorithm of Goemans and Kleinberg to the problem with multiple repairmen.

All polynomial time approximation algorithms mentioned for METRIC-TRP

use k-MST as a subroutine. This in itself does not exclude finding good approximation algorithms. Arora and Karakostas [7] showed that the optimal TRP-tour can be partitioned in $O(\log(n)/\epsilon)$ paths such that the overall loss for the TRP objective is at most a factor ϵ if any path is replaced by an optimal traveling salesman path. Any of the algorithms use at most n TSP tours (or MST's). One might obtain better approximation algorithms by defining this set in a more sophisticated way. On one hand this set must be small enough to maintain polynomial running time, and on the other hand it must be rich enough to construct a small TRP-tour.

4.6 Related problems

A variant of TRP is the *search ratio problem*, introduced by Koutsoupias et al. [52]. One has to find an object hidden in a vertex not known to the server. Unlike the graph searching problem, where a distribution on the vertices is given, one is facing an adversary who chooses at which vertex it hides the object. The search ratio is then defined as the optimal competitive ratio, which is the distance traveled by the server, divided by the length of the shortest path from the origin to the object. Koutsoupias et al. [52] show that both the randomized and the deterministic version are $Max\text{-}SNP$-hard for general graphs. They note that computing the (randomized) search ratio for trees is a 'surprisingly tough problem' but show that, if one can solve the TRP for a class of graphs in polynomial time, then by duality one can solve the randomized search ratio problem for that same class of graphs. Our result excludes this tool for finding a polynomial time algorithm for the randomized search ratio problem for trees. However, polynomial time approximation algorithms for TRP may be transferable to the randomized search ratio problem [52].

The value of the optimal TRP-tour clearly depends on the given starting vertex. To find the best starting vertex we could repeatedly run an algorithm that solves TRP, each time using a different starting vertex. However, Sichmi-Levi and Berman [78] notice that for general metric spaces one run suffices: add a point at the same large distance to any of the other points. Wu [88] gives an $O(n^2)$ algorithm for this problem on the line, and Minieka [65] notices that for unweighted trees, any vertex at the end of a longest path can be taken as the best starting vertex. It is unknown to us whether, for weighted trees and for arbitrary metric spaces, there is a more subtle way than solving a TRP instance to find the best starting vertex.

4.7 Open problems

We summarize some of the open problems mentioned in this chapter and raise some new ones.

The tree used in the proof of Theorem 9 is not a caterpillar. So the complexity of the LINE-TRP with repair times remains open. But our proof does show that the repair time problem is strongly NP-hard for another kind of trees: spider graphs. Such graphs consist of a set of paths connected in a single point.

It remains open whether LINE-TRP with release dates is strongly NP-hard or pseudo-polynomially solvable.

It is interesting to see if we can push NP-hardness for trees even further than we did in Corollary 3 and Theorem 10. In this context we consider the problem in which all edge-lengths are strictly positive integers and bounded by some constant k. It is easy to prove that in any optimal tour, the tour through the first n/k points must have minimum length. This implies for the tree metric that the first n/k points are visited in depth-first order. More generally, if at some moment still m points are unvisited in the tree, then the next m/k points are visited in depth-first order. It seems that to prove NP-hardness we need an essentially different reduction than we used in Section 4.4. On the other hand, a polynomial time algorithm follows if we can prove a polynomial upper bound on the number of interesting states in the dynamic program 4.1. The complexity status of this problem remains open.

We restrict the problem even more if we bound all distances of the metric space by some constant k. Blum et al. [12] gave a polynomial time algorithm for $k \leq 3$ but there does not seem to be an easy away to extend this to larger diameter trees.

A major challenge in this research is to find good approximation algorithms for the traveling repairman problem. For example, the smallest approximation ratios for TREE-TRP and METRIC-TRP are respectively 3.59 and 7.18. Compare this to the ratios 1 and 1.5 for the corresponding TSP problems.

Chapter 5

The general two-server problem

5.1 Introduction

In the *general k-server problem* we are given k servers each of which is moving in some metric space $\mathbb{M}_i$, $(i = 1, \ldots, k)$, starting in some prefixed point $O_i \in \mathbb{M}_i$. They are to serve requests $r \in \mathbb{M}_1 \times \mathbb{M}_2 \times \ldots \times \mathbb{M}_k$ which arrive one by one. A request $r = (z_1, z_2, \ldots, z_k)$ is served by moving, for at least one i, the server in space $\mathbb{M}_i$ to the point z_i. The decision which server to move to the next request is irrevocable and has to be taken without any knowledge about future requests. The cost of moving the ith server to z_i is equal to the distance travelled by this server from his current location to z_i. The objective is to minimize the total cost to serve all given requests.

We can see the general k-server problem as a single server problem, moving in the metric space $\mathbb{M} = \mathbb{M}_1 \times \ldots \times \mathbb{M}_k$, and interpret the server positions and the requests in the description above as the 'coordinates' of the single server and the requests. Interpreted in this way, it is a special case of a *metrical service system*, in which each request can be any subset of the metric space, and is served by moving the server to one of the points in the request. Metrical service systems were introduced in [24] to provide a formalism for investigating a wide variety of on-line optimization problems. A precise definition is given in Section 5.2. In the same section we derive the basic theorem of this chapter. It provides a sufficient condition for the existence of constant competitive algorithms for general metrical service systems. The result on the general two-server problem then becomes a matter of verifying this condition.

The general k-server problem is a natural generalization of the well-known *k-server problem* for which $\mathbb{M}_1 = \mathbb{M}_2 = \ldots = \mathbb{M}_k$ and $z_1 = z_2 = \ldots = z_k$ at

each time step. The k-server problem was introduced by Manasse, McGeoch and Sleator [61], who proved a lower bound of k on the competitive ratio of any deterministic algorithm for any metric space with at least $k+1$ points and posed the well-known k-server conjecture saying that there exists a k-competitive algorithm for any metric space. The conjecture has been proved for $k = 2$ [61] and some special metric spaces [22][23]. For $k \geq 3$ the current best upper bound of $2k - 1$ is given by Koutsoupias and Papadimitriou [51].

The *weighted k-server problem* turns out to be much harder. In this problem a weight is assigned to each server and the total cost is the sum of the weighted distances. Fiat and Ricklin [33] prove that for any metric space there exists a set of weights such that the competitive ratio of any deterministic algorithm is at least $k^{\Omega(k)}$. For a uniform metric space, on which the problem is called the weighted paging problem, Feuerstein et al. [32] give a 6.275-competitive algorithm. For $k = 2$ Chrobak and Sgall [25] provided a 5-competitive algorithm and proved that no better competitive ratio is possible.

A weighted k-server algorithm is called competitive if the competitive ratio is independent of the weights. For a general metric space no competitive algorithm is known yet even for $k = 2$. It is easy to see that the general k-server problem is a generalization of the weighted k-server problem as well.

The general two-server problem in which both servers move on the real line has become well-known as the CNN-problem. Koutsoupias and Taylor [53] emphasize the importance of the CNN-problem as one of the simplest problems in a rich class of so-called sum-problems [13]. In the sum-problem each of a set of systems gets a request and only one system has to serve this request.

Koutsoupias and Taylor [53] prove a lower bound of $6 + \sqrt{17}$ on the competitive ratio of any deterministic on-line algorithm for the general two-server problem, through an instance of the weighted two-server problem on the real line. They also conjecture that the *generalized work function algorithm* has constant competitive ratio for the general two-server problem. It seems to be a bad tradition of multiple-server problems to keep unsettled conjectures. For the general two-server problem the situation was even worse than for the k-server problem: the question if any algorithm exists with constant competitive ratio remained unanswered.

In Section 5.3 we answer this question affirmatively, by designing an algorithm and prove a rough upper boud of $44,800$ on its competitive ratio. Our algorithm is a combination of the well-known *balance algorithm* and the *generalized work function algorithm*. The result is merely checking the condition of the general theorem for metrical service systems in Section 5.2, announced above.

Optimal off-line solutions of metrical service systems can easily be found by dynamic programming (see [13]), which yields an $O(n^k)$ time algorithm for the

general k-server problem. For the classical k-server problem this running time can be reduced to $O(kn^2)$ by formulating it as a min-cost flow problem [22]. No such improvement should be expected for the general k-sever problem since the problem is NP-hard as we will show in Section 5.4.

5.2 Competitiveness of metrical service systems

A metrical service system $\mathcal{S} = (\mathbb{M}, \mathcal{R})$ is specified by a metric space $\mathbb{M}$ with distance function $d : \mathbb{M}^2 \to \mathbb{R}^+$ and a set of requests $\mathcal{R}$. Each request $r \in \mathcal{R}$ is a subset of $\mathbb{M}$. An instance of the system consists of an initial server position $O \in \mathbb{M}$ and a sequence of requests $\sigma = r_1, r_2, \ldots, r_n$. Every request r_t must be served immediately and irrevocably by moving the server to a point $x_t \in r_t$, before the future requests, $r_{t+1}, r_{t+2}, \ldots$, are given. The cost of the solution is the length of the path in $\mathbb{M}$ followed by the server.

For any metrical service system we will define functions $\Psi_m(\sigma) \geq 0$ depending on the input sequence σ and any integer $m \geq 1$. We show that if an algorithm A exists for a metrical service system $\mathcal{S}$ with the property that for some $m \geq 1$ and any sequence σ the length of its path is at most c times the length of the optimal path plus $\Psi_m(\sigma)$, then there exists an on-line algorithm A' for this metrical service system producing a path of length at most c' times the optimal path length, for some constant c'. Since $\Psi_m(\sigma) \geq 0$ the reverse of this statement is obviously always true, i.e. if A' is c' competitive then the length of any solution is certainly no more than c' times the optimal path plus $\Psi_m(\sigma)$. If we would choose $\Psi_m(\sigma) \equiv 0$ then the statement is clearly a tautology. The statement is formalized in Theorem 11 and the usefulness of our choice of $\Psi_m(\sigma)$ follows from Section 5.3 in which we show how a competitive algorithm for the general two-server problem follows quite easily from Theorem 11. Roughly speaking we define $\Psi_m(\sigma)$ as the minimum sum of lengths of the paths of m different adversaries, each serving the request sequence σ. What is meant by *different* adversaries will be formalized later. The theorem then says that if there exists an algorithm that is competitive against m different adversaries then there exists an algorithm that is competitive against one adversary.

The competitive algorithm A', which is called ONLINE from now on, is a combination of an algorithm A with the property mentioned above and the *generalized work function algorithm*. The latter algorithm was introduced independently by several people and has shown to be competitive for several on-line problems. In fact we go along with Koutsoupias and Taylor [53] in conjecturing that it is competitive for the general two-server problem as well. The generalized work function algorithm bases its moves on the position of the on-line server and the values of the *work function*.

Definition 8 *Given a metrical service system $\mathcal{S} = (\mathbb{M}, \mathcal{R})$ with origin $O \in \mathbb{M}$, and request sequence σ we define the* work function $W_{O;\sigma} : \mathbb{M} \to \mathbb{R}^+$. *For any point $s \in \mathbb{M}$, $W_{O;\sigma}(s)$ is the length of the shortest path that starts in O, ends in s and serves σ. Omitting the restriction on the initial point we obtain an alternative work function $\widehat{W}_\sigma$, i.e., $\widehat{W}_\sigma(s)$ is the length of the shortest path that ends in s and serves σ.*

We assume here that the work function is always well-defined, i.e. for any $\sigma = r_1, \ldots, r_n$ and point $s \in \mathbb{M}$ there are points $s_i \in r_i$ $(i = 1 \ldots n)$ such that $d(O, s_1) + d(s_1, s_2) + \ldots + d(s_{n-1}, s_n) + d(s_n, s) \leq d(O, t_1) + d(t_1, t_2) + \ldots + d(t_{n-1}, t_n) + d(t_n, s)$ for any set of points $t_i \in r_i$ $(i = 1 \ldots n)$. Notice that this implies $W_{O;r_1,\ldots,r_i}(s_i) = d(O, s_1) + d(s_1, s_2) + \ldots + d(s_{i-1}, s_i)$ for all $i \in \{1, \ldots, n\}$.

We will use the following properties, which are rather obvious. The work function on the empty string of requests is $W_{O;\emptyset}(s) = d(O, s)$, for all $s \in \mathbb{M}$. The work function is Lipschitz continuous: for any two points s and s' in $\mathbb{M}$, $|W_{O;\sigma}(s) - W_{O;\sigma}(s')| \leq d(s, s')$. It exhibits monotonicity with respect to the request sequence for every $s \in \mathbb{M}$: given any request sequence σ and any new request r, $W_{O;\sigma,r}(s) \geq W_{O;\sigma}(s)$, for all $s \in \mathbb{M}$. Equality holds for all $s \in r$.

Given a work function $W_{O;\sigma}$ we say that point s is *dominated* by point t if $W_{O;\sigma}(s) = W_{O;\sigma}(t) + d(s, t)$. We define the *support* of $W_{O;\sigma}$ as

$$\mathrm{Supp}(W_{O;\sigma}) = \{t \in \mathbb{M} : t \text{ is not dominated by any other point}\}.$$

If $W_{O;\sigma,r}$ is a well-defined work function then $\mathrm{Supp}(W_{O;\sigma,r}) \subseteq r$, since for any point s there is a point $t \in r$ such that $W_{O;\sigma,r}(s) = W_{O;\sigma,r}(t) + d(t, s)$, which implies that if $s \in \mathrm{Supp}(W_{O;\sigma,r})$ then $s = t$. For more properties and a deeper analysis of work functions we refer to [18],[24].

The generalized work function algorithm is a work function based algorithm parameterized by some constant $\lambda < 1$. We call it λ-WFA. For any request sequence σ and any new request r, λ-WFA-moves the server from the position s it had after serving σ to point

$$s' = \mathrm{argmin}_{t \in \mathbb{M}}\{W_{O;\sigma,r}(t) + \lambda d(s, t)\}.$$

Note that this minimum may not be well-defined if the request contains infinitely many points of the metric space. In Theorem 11 we assume that the minimum is always attained for some $s' \in \mathbb{M}$. As a result of any such a λ-WFA-move, we have for any point $t \in \mathbb{M}$ that

$$W_{O;\sigma,r}(s') + \lambda d(s, s') \leq W_{O;\sigma,r}(t) + \lambda d(s, t).$$

Therefore, in particular,

$$W_{O;\sigma,r}(s') \leq W_{O;\sigma,r}(s) - \lambda d(s, s'), \tag{5.1}$$

and for any $t \in \mathbb{M}$

$$W_{O;\sigma,r}(s') \leq W_{O;\sigma,r}(t) + \lambda d(s', t). \tag{5.2}$$

The choice $\lambda < 1$ together with (5.2) implies that s' is not dominated by any other point, whence $s' \in \mathrm{Supp}(W_{O;\sigma,r}) \subseteq r$. We see that if the moves of λ-WFA are well-defined then the choice of $\lambda < 1$ ensures that the point s' always serves the last request.

As announced before, we define an algorithm ONLINE as a combination of λ-WFA and some algorithm A. ONLINE works in phases. The final point of a phase is the starting point of the next phase. This is the only information taken from one phase to the next phase. The phases of the algorithm induce a partition of the request sequence σ. Let σ_j be the subsequence of requests served in the j-th phase of ONLINE; i.e., if J is the total number of phases then $\sigma = \sigma_1, \ldots, \sigma_J$. We denote by $A(s, \sigma)$ and $\mathrm{OPT}(s, \sigma)$ the cost of the algorithm A and the cost of the optimal path starting in s and serving request sequence σ. Thus, $\mathrm{OPT}(O, \sigma) = \min_{t \in \mathbb{M}} W_{O;\sigma}(t)$. In the description of a generic phase j of the algorithm we use σ_j^h for the subsequence consisting of the first h requests of σ_j. The starting point in phase j is denoted by O_j. The algorithm is parameterized by γ ($\gamma \geq 1$) and λ ($0 < \lambda < 1$).

> *Phase j of* ONLINE(A, γ, λ):
> Given σ_j^h, if $A(O_j, \sigma_j^h) \leq \gamma \mathrm{OPT}(O_j, \sigma_j^h)$, move the server according to A and continue the phase. Otherwise, $\sigma_j = \sigma_j^h$; move the server according to λ-WFA, based on the work function $W_{O;\sigma_1,\ldots,\sigma_j}$, and start a new phase.

Thus, each phase contains only one λ-WFA-move and this moves concludes the phase. We emphasize that the work function employed in the λ-WFA-move is defined over the complete input sequence, released sofar. In the sequel we refer to the moves made according to A as A-moves.

We are now ready to define our main theorem. The idea behind it is best explained by using adversaries serving the same request sequence, but knowing the sequence in advance. Whereas our algorithm A has to start from O, these adversaries are allowed to choose their starting point. On the other hand we restrict the paths of the adversaries by requiring that they are sufficiently different. We do so by requiring that the points where the m paths end are pairwise σ-independent.

Definition 9 *Given a sequence of requests σ we say that the points s_1 and s_2 are* σ-independent *if $d(s_1, s_2) \geq \widehat{W}_\sigma(s_1) + \widehat{W}_\sigma(s_2)$. Otherwise, the points are called σ-dependent.*

The theorem says that if A is competitive against the optimal path plus m such adversaries, then algorithm ONLINE is competitive against one adversary starting also in O, i.e., against the length of the optimal path.

Theorem 11 *Let A be an algorithm for some metrical service system $\mathcal{S}$, with origin O, on which the work function and λ-WFA are well-defined. If there exist constants c and $m \geq 2$ such that for any sequence σ and any set $\{s_1, \ldots, s_m\}$ of m pairwise σ-independent points*

$$A(O, \sigma) \leq c\,\mathrm{OPT}(O, \sigma) + c \sum_{h=1}^{m} \widehat{W}_\sigma(s_h), \tag{5.3}$$

then $\mathrm{ONLINE}(A, \gamma, \lambda)$ with $\gamma = c(1-\lambda)(1-2\lambda)^{-1}$ is $(2+5/4c)\lambda^{-2}(1/2-5\lambda)^{1-m}$-competitive for $\mathcal{S}$, for every $\lambda < 1/10$.

If algorithm A satisfies (5.3) for $m = 1$ then A itself is $2c$ competitive for S. Restricting m to be at least 2 in the theorem allowed us to prove a slightly better ratio of ONLINE. In Section 5.3 we show that for the general two-server problem the balance algorithm is competitive against the optimal path plus 3 such adversaries. A competitive algorithm follows then directly from the theorem.

As a preliminary to the proof we define functions and prove some of their properties. Given any function $f : \mathbb{X} \to \mathbb{R}$, defined on some metric space $\mathbb{X}$, define, for $i \geq 1$ and $0 < \beta < 1$, the functions $G_f^i : \mathbb{X}^i \to \mathbb{R}$ as

$$G_f^i(x_1, \ldots, x_i) = f(x_i) - \min_{h<i}\{\ f(x_h) + \beta d(x_h, x_i)\ \}, \tag{5.4}$$

and for $n \geq 1$ and $0 < \alpha < 1/2$ the functions $F_f^n : \mathbb{X}^n \to \mathbb{R}$ as

$$F_f^n(x_1, \ldots, x_n) = \sum_{i=1}^{n} \alpha^{i-1} G_f^i(x_1, \ldots, x_i). \tag{5.5}$$

We denote $\mathcal{F}_f^n = \min_{x_1,\ldots,x_n \in \mathbb{X}}\{F_f^n(x_1, \ldots, x_n)\}$.

Lemma 6 *For any two functions, f and g, and any two sequences, $(x_1, \ldots, x_n)$ and $(y_1, \ldots, y_n)$ such that $f(x_i) \geq g(y_i)$ for all $i \in \{1, \ldots, n\}$ we have*

$$F_f^n(x_1, \ldots, x_n) - F_g^n(y_1, \ldots, y_n) \geq$$
$$\sum_{i=1}^{n} \left((\ \alpha^{i-1} - \sum_{h=i+1}^{n} \alpha^{h-1}\ )(\ f(x_i) - g(y_i)\ ) - \beta d(x_i, y_i) \sum_{h=i}^{n} \alpha^{h-1} \right) .$$

PROOF. Suppose that $h_i < i$ is such that

$$G_g^i(y_1, \ldots, y_i) = g(y_i) - (g(y_{h_i}) + \beta d(y_i, y_{h_i})).$$

Clearly,

$$\begin{aligned} G_f^i(x_1,\ldots,x_i) &= f(x_i) - \min_{h<i}\{f(x_h) + \beta d(x_i,x_h)\} \\ &\geq f(x_i) - (f(x_{h_i}) + \beta d(x_i,x_{h_i})). \end{aligned}$$

Hence, using the triangle inequality to bound $-\beta d(x_i,x_{h_i})$,

$$\begin{aligned} & G_f^i(x_1,\ldots,x_i) - G_g^i(y_1,\ldots,y_i) \\ \geq\ & f(x_i) - g(y_i) - (f(x_{h_i}) - g(y_{h_i})) - \beta d(x_i,x_{h_i}) + \beta d(y_i,y_{h_i}) \\ \geq\ & f(x_i) - g(y_i) - (f(x_{h_i}) - g(y_{h_i})) - \beta d(x_i,y_i) - \beta d(x_{h_i},y_{h_i}). \end{aligned}$$

Thus,

$$\begin{aligned} & F_f^n(x_1,\ldots,x_n) - F_g^n(y_1,\ldots,y_n) \\ =\ & \textstyle\sum_{i=1}^n \alpha^{i-1}(G_f^i(x_1,\ldots,x_i) - G_g^i(y_1,\ldots,y_i)) \\ \geq\ & \textstyle\sum_{i=1}^n \alpha^{i-1}((\ f(x_i) - g(y_i)\ - \beta d(x_i,y_i)) - (f(x_{h_i}) - g(y_{h_i})) - \beta d(x_{h_i},y_{h_i})) \\ \geq\ & \textstyle\sum_{i=1}^n (\ (\ \alpha^{i-1} - \sum_{h=i+1}^n \alpha^{h-1}\ )(\ f(x_i) - g(y_i)\ ) - \beta d(x_i,y_i) \sum_{h=i}^n \alpha^{h-1}\ ). \end{aligned}$$

□

Lemma 7 *Assume that $\mathcal{F}_f^n = F_f^n(x_1,\ldots,x_n)$. Any pair $p,q \in \{1,\ldots,n\}$ satisfies*

$$|f(x_p) - f(x_q)| \leq (1-2\alpha)^{-1}\beta d(x_p,x_q).$$

Proof. Assume without loss of generality that $f(x_p) \geq f(x_q)$. Since the global minimum of F_f^n is attained in $(x_1,\ldots,x_n)$ we must have that $F_f^n(x_1,\ldots,x_n) - F_f^n(x_1,\ldots,x_{p-1},x_q,x_{p+1},\ldots,x_n) \leq 0$. Now we apply Lemma 6 with $g \equiv f$, $y_p = x_q$ and $y_i = x_i$ for $i \neq p$.

$$\begin{aligned} f(x_p) - f(x_q) &\leq (\sum_{h=p}^n \alpha^{h-1})(\ \alpha^{p-1} - \sum_{h=p+1}^n \alpha^{h-1}\ )^{-1}\beta d(x_p,x_q) \\ &\leq (\sum_{h=p}^\infty \alpha^{h-1})(\ \alpha^{p-1} - \sum_{h=p+1}^\infty \alpha^{h-1}\ )^{-1}\beta d(x_p,x_q) \\ &= (\alpha^{p-1}/(1-\alpha))(\alpha^{p-1} - \alpha^p/(1-\alpha))^{-1}\beta d(x_p,x_q) \\ &= (\ 1-2\alpha\ )^{-1}\beta d(x_p,x_q). \end{aligned}$$

□

Preliminary to the proof of Theorem 11 we derive an inequality which will be used several times. In the proof we choose $\beta = (1-2\alpha)/5$ implying that for any $\alpha < 1/2$ we have $\alpha(1+\beta) \leq (1-\alpha)(1-\beta)$ and for any $n \geq 1$ and any $1 \leq i \leq n$:

$$\begin{aligned} & \alpha^{i-1} - \sum_{h=i+1}^{n} \alpha^{h-1} - \beta \sum_{h=i}^{n} \alpha^{h-1} \\ = \;& (1-\beta)\alpha^{i-1} - (1+\beta) \sum_{h=i+1}^{n} \alpha^{h-1} \\ = \;& (1-\beta)\alpha^{i-1} - (1+\beta)(\alpha^i - \alpha^n)/(1-\alpha) \\ \geq \;& (1-\beta)\alpha^{i-1} - (1-\beta)(1-\alpha)/\alpha \cdot (\alpha^i - \alpha^n)/(1-\alpha) \\ = \;& (1-\beta)\alpha^{n-1}. \end{aligned} \tag{5.6}$$

Proof of Theorem 11. For each $j \geq 1$ we define for $n \geq 1$ the functions G_j^i, for $i = 1 \ldots, n$, and F_j^n, as in (5.4) and (5.5) by choosing $f(s) = W_{O;\sigma_1,\ldots,\sigma_{j-1}}(s)$. We define $\sigma_0 = \emptyset$.

From now on we let $\beta = (1-2\alpha)/5$ and $\alpha = 1/2 - 5\lambda$, whence $\beta = 2\lambda$. We also assume $0 < \lambda < 1/10$ implying $0 < \alpha < 1/2$.

First we show that $\mathcal{F}_j^n$ is a lower bound on the length of the optimal path serving $\sigma_1, \ldots, \sigma_{j-1}$ for every $j \in \{1, \ldots, J+1\}$ and any $n \geq 1$. From the definition, $G_j^1(s) = W_{\sigma_1,\ldots,\sigma_{j-1}}(s)$ and $G_j^i(s, \ldots, s) = 0$, for all $i > 1$, whence $F_j^n(s, \ldots, s) = W_{O;\sigma_1,\ldots,\sigma_{j-1}}(s)$ for any j and $n \geq 1$. In particular this implies that

$$\mathcal{F}_j^n \leq \min_{s \in \mathbb{M}} F_j^n(s, \ldots, s) = \min_{s \in \mathbb{M}} W_{O;\sigma_1,\ldots,\sigma_{j-1}}(s) = \textsc{Opt}(O, \sigma_1, \ldots, \sigma_{j-1}). \tag{5.7}$$

Also, $F_1^n(O, O, \ldots, O) = W_{O;\emptyset}(O) = 0$, for any $n \geq 1$. In fact $\mathcal{F}_1^n = 0$, for every $n \geq 1$, which follows from Lemma 6 applied to any series $(s_1, \ldots, s_n)$:

$$\begin{aligned} & F_1^n(s_1, \ldots, s_n) - F_1^n(O, \ldots, O) \\ \geq \;& \sum_{i=1}^{n} ((\alpha^{i-1} - \sum_{h=i+1}^{n} \alpha^{h-1})(W_{O;\emptyset}(s_i) - W_{O;\emptyset}(O)) - \beta d(s_i, O) \sum_{h=i}^{n} \alpha^{h-1}) \\ = \;& \sum_{i=1}^{n} (\alpha^{i-1} - \sum_{h=i+1}^{n} \alpha^{h-1} - \beta \sum_{h=i}^{n} \alpha^{h-1}) d(s_i, O) \\ \geq^{(5.6)} \;& \sum_{i=1}^{n} (1-\beta)\alpha^{n-1} d(s_i, O) \geq 0. \end{aligned}$$

Hence,

$$\mathcal{F}_1^n = F_1^n(O, \ldots, O) = 0. \tag{5.8}$$

First we show that Theorem 11 follows from the next claim. Allowing σ_J to be empty, we assume, without loss of generality, that phase J ends with an A-move.

Claim. *For each phase $j \le J-1$ we have* $\mathcal{F}^m_{j+1} - \mathcal{F}^m_j \ge \lambda\alpha^{m-1}\textsc{Opt}(O_j, \sigma_j)$.

Let C_j be the cost of the A-moves in phase j. By definition of algorithm Online, $C_j \le \gamma\textsc{Opt}(O_j, \sigma_j)$. Let O'_j denote the position of the server after the last A-move in phase j. Clearly,

$$\begin{aligned} W_{O;\sigma_1,\ldots,\sigma_j}(O'_j) - W_{O;\sigma_1,\ldots,\sigma_{j-1}}(O_j) &\le 2\textsc{Opt}(O_j,\sigma_j) + d(O_j, O'_j) \\ &\le (2+\gamma)\textsc{Opt}(O_j,\sigma_j). \end{aligned}$$

Since the last move in phase j, the move from O'_j to O_{j+1}, is a λ-Wfa-move, we use (5.1) to obtain

$$W_{O;\sigma_1\ldots,\sigma_j}(O_{j+1}) \le W_{O;\sigma_1,\ldots,\sigma_j}(O'_j) - \lambda d(O'_j, O_{j+1}).$$

Therefore,

$$\begin{aligned} & \textstyle\sum_{j=1}^{J-1} d(O'_j, O_{j+1}) \\ \le\ & 1/\lambda \textstyle\sum_{j=1}^{J-1}(W_{O;\sigma_1,\ldots,\sigma_j}(O'_j) - W_{O;\sigma_1\ldots,\sigma_j}(O_{j+1})) \\ \le\ & 1/\lambda(2+\gamma)\textstyle\sum_{j=1}^{J-1}\textsc{Opt}(O_j,\sigma_j) + \\ & \qquad 1/\lambda\textstyle\sum_{j=1}^{J-1}(W_{O;\sigma_1,\ldots,\sigma_{j-1}}(O_j) - W_{O;\sigma_1\ldots,\sigma_j}(O_{j+1})) \\ =\ & 1/\lambda(2+\gamma)\textstyle\sum_{j=1}^{J-1}\textsc{Opt}(O_j,\sigma_j) + 1/\lambda(W_{O;\emptyset}(O) - W_{O;\sigma_1\ldots,\sigma_{J-1}}(O_J)) \\ \le\ & 1/\lambda(2+\gamma)\textstyle\sum_{j=1}^{J-1}\textsc{Opt}(O_j,\sigma_j). \end{aligned}$$

Hence,

$$\textsc{Online}_\sigma \le (2/\lambda + \gamma/\lambda + \gamma)\sum_{j=1}^{J-1}\textsc{Opt}(O_j,\sigma_j) + C_j.$$

Applying the claim together with (5.7) and (5.8) yields

$$\begin{aligned} \textsc{Online}_\sigma &\le (2/\lambda+\gamma/\lambda+\gamma)\lambda^{-1}\alpha^{1-m}\sum_{j=1}^{J-1}(\mathcal{F}^m_{j+1} - \mathcal{F}^m_j) + C_j \\ &= (2/\lambda+\gamma/\lambda+\gamma)\lambda^{-1}\alpha^{1-m}(\mathcal{F}^m_J - \mathcal{F}^m_1) + C_j \qquad (5.9) \\ &\le (2/\lambda+\gamma/\lambda+\gamma)\lambda^{-1}\alpha^{1-m}\textsc{Opt}(O,\sigma) + C_j. \end{aligned}$$

Now we bound C_j. Let s be the endpoint of an optimal path starting in O and serving σ.

$$\begin{aligned} (1-\lambda)d(O,O_J) &\le W_{O;\sigma_1,\ldots,\sigma_{J-1}}(O_J) - \lambda d(O,O_J) \\ &\le^{(5.2)} W_{O;\sigma_1,\ldots,\sigma_{J-1}}(s) + \lambda d(O_J,s) - \lambda d(O,O_J) \\ &\le W_{O;\sigma_1,\ldots,\sigma_{J-1}}(s) + \lambda d(O,s) \\ &\le \textsc{Opt}(O,\sigma) + \lambda\textsc{Opt}(O,\sigma). \end{aligned}$$

Hence,

$$\begin{aligned} C_j &\leq \gamma \textsc{Opt}(O_J, \sigma_J) \\ &\leq \gamma(d(O, O_J) + \textsc{Opt}(O, \sigma)) \\ &\leq \gamma((1+\lambda)/(1-\lambda) + 1)\textsc{Opt}(O, \sigma). \end{aligned}$$

Together with the choices $\gamma = c(1-\lambda)/(1-2\lambda)$ and $0 < \lambda < 1/10$ and $m \geq 2$ this yields

$$\begin{aligned} & \textsc{Online}_\sigma \\ \leq\ & ((2+\gamma+\gamma\lambda)\lambda^{-2}(1/2-5\lambda)^{1-m} + \gamma((1+\lambda)/(1-\lambda)+1))\textsc{Opt}(O,\sigma) \\ \leq\ & ((2+99/80c)\lambda^{-2}(1/2-5\lambda)^{1-m} + 5/2c)\textsc{Opt}(O,\sigma) \\ <\ & ((2+5/4c)\lambda^{-2}(1/2-5\lambda)^{1-m}\textsc{Opt}(O,\sigma). \end{aligned}$$

Proof of Claim. Consider a specific phase j, $1 \leq j \leq J-1$. Given a point $s \in \mathbb{M}$ we define s^- as the starting point at the beginning of phase j of the part of the optimal path, serving $\sigma_1, \ldots, \sigma_j$ and ending in s; i.e., $W_{s^-;\sigma_j}(s)$ is the length of the part of the optimal path ending in s which is traversed in phase j. By these definitions $W_{O;\sigma_1,\ldots,\sigma_j}(s) = W_{O;\sigma_1,\ldots,\sigma_{j-1}}(s^-) + W_{s^-;\sigma_j}(s)$. Clearly, $W_{s^-;\sigma_j}(s) \geq d(s, s^-)$. Moreover, remembering the definition of $\widehat{W}$, we have $\widehat{W}_{\sigma_j}(s) \leq W_{s^-;\sigma_j}(s)$. In the remainder of the proof we simplify notation by abbreviating $W_{O;\sigma_1,\ldots,\sigma_j}(s)$ to $W_{\leq j}(s)$ and $W_{s^-;\sigma_j}(s)$ and $\widehat{W}_{\sigma_j}(s)$ to, respectively, $W_j(s)$ and $\widehat{W}_j(s)$. Assume $\mathcal{F}^m_{j+1} = F^m_{j+1}(t_1, \ldots, t_m)$, i.e. the minimum at the end of phase j is attained in the points $(t_1, \ldots, t_m)$. We distinguish two cases.

Case 1. The points $t_1, \ldots, t_m$ are pairwise σ_j-independent. Condition (5.3) together with the definition of Online implies that for any $j \leq J-1$ we have $\gamma\textsc{Opt}(O_j, \sigma_j) \leq C_j \leq c\textsc{Opt}(O_j, \sigma_j) + c\sum_{i=1}^m \widehat{W}_j(t_i)$, whence $\sum_{i=1}^m W_j(t_i) \geq \sum_{i=1}^m \widehat{W}_j(t_i) \geq (\gamma/c - 1)\textsc{Opt}(O_j, \sigma_j)$. We apply Lemma 6 with $f \equiv W_{\leq j}$ and with $g \equiv W_{\leq j-1}$.

$$\begin{aligned} & \mathcal{F}^m_{j+1} - \mathcal{F}^m_j \\ \geq\ & F^m_{j+1}(t_1, \ldots, t_m) - F^m_j(t_1^-, \ldots, t_m^-) \\ \geq\ & \sum_{i=1}^m ((\alpha^{i-1} - \sum_{h=i+1}^m \alpha^{h-1})(W_{\leq j}(t_i) - W_{\leq j-1}(t_i^-)) - \beta d(t_i, t_i^-) \sum_{h=i}^m \alpha^{h-1}) \\ \geq\ & \sum_{i=1}^m (\alpha^{i-1} - \sum_{h=i+1}^m \alpha^{h-1} - \beta \sum_{h=i}^m \alpha^{h-1}) W_j(t_i) \end{aligned}$$

$$\begin{aligned} &\geq^{(5.6)} && \alpha^{m-1}(1-\beta)\sum_{i=1}^{m} W_j(t_i) \\ &\geq && \alpha^{m-1}(1-\beta)(\gamma/c-1)\textsc{Opt}(O_j,\sigma_j) \\ &= && \alpha^{m-1}\lambda\textsc{Opt}(O_j,\sigma_j). \end{aligned}$$

Case 2. There are points t_p and t_q ($p \neq q \in \{1,\ldots,m\}$) that are σ_j-dependent, i.e. $d(t_p,t_q) < \widehat{W}_j(t_p) + \widehat{W}_j(t_q) \leq W_j(t_p) + W_j(t_q)$. Assume without loss of generality that $W_j(t_p) \geq W_j(t_q)$. Define the series $(u_1,\ldots,u_m)$ by $u_i = t_i^-$ for $i \neq q$ and $u_q = t_p^-$. Lemma 6 implies

$$F_{j+1}^m(t_1,\ldots,t_m) - F_j^m(u_1,\ldots,u_m) \geq \sum_{i=1}^{m} H_i, \tag{5.10}$$

with for any $i \neq q$,

$$\begin{aligned} H_i &= && (\alpha^{i-1} - \sum_{h=i+1}^{m} \alpha^{h-1} - \beta\sum_{h=i}^{m}\alpha^{h-1})W_j(t_i) \\ &\geq^{(5.6)} && \alpha^{m-1}(1-\beta)W_j(t_i) \geq 0 \end{aligned}$$

and

$$H_q = (\ \alpha^{q-1} - \sum_{h=q+1}^{m} \alpha^{h-1}\ )(\ W_{\leq j}(t_q) - W_{\leq j-1}(t_p^-)\ ) - \beta d(t_q,t_p^-)\sum_{h=q}^{m}\alpha^{h-1}.$$

To see that $H_q \geq 0$, notice that by Lemma 7

$$\begin{aligned} W_{\leq j}(t_q) - W_{\leq j-1}(t_p^-) &= (W_{\leq j}(t_p) - W_{\leq j-1}(t_p^-)) + (W_{\leq j}(t_q) - W_{\leq j}(t_p)) \\ &\geq W_j(t_p) - (1-2\alpha)^{-1}\beta d(t_p,t_q) \\ &> W_j(t_p) - 2(1-2\alpha)^{-1}\beta W_j(t_p). \end{aligned}$$

Also notice that

$$d(t_q,t_p^-) \leq d(t_q,t_p) + d(t_p,t_p^-) \leq 2W_j(t_p) + W_j(t_p) = 3W_j(t_p).$$

Hence, since we have chosen $\beta = (1-2\alpha)/5$,

$$\begin{aligned} H_q\ /\ W_j(t_p) &\geq (\ \alpha^{q-1} - \sum_{h=q+1}^{m} \alpha^{h-1}\ )(1-2(1-2\alpha)^{-1}\beta)\ -\ 3\beta\sum_{h=q}^{m}\alpha^{h-1} \\ &= (\alpha^{q-1} - \sum_{h=q+1}^{m} \alpha^{h-1}\ )3/5 - 3(1-2\alpha)/5(\alpha^{q-1} + \sum_{h=q+1}^{m}\alpha^{h-1}) \\ &= 6/5(\alpha-1)\sum_{h=q+1}^{m}\alpha^{h-1}\ + 6/5\alpha^q\ =\ 6/5\alpha^m \geq 0. \end{aligned}$$

In the series $(u_1, \ldots, u_m)$ the point t_p^- appears at least twice, say $u_a = t_p^-$ and $u_b = t_p^-$, with $a < b$. By definition

$$G_j^b(u_1, \ldots, u_b) \geq W_{\leq j-1}(u_b) - \{W_{\leq j-1}(u_a) + \beta d(u_a, u_b)\} = 0. \tag{5.11}$$

Next we define series $(v_1, \ldots, v_{m-1})$ with $v_i = u_i$ for $i = 1, \ldots, b-1$. We distinguish two cases in defining $v_b, \ldots, v_{m-1}$. If $\sum_{i=b+1}^m G_j^i(u_1, \ldots, u_i) \leq 0$ then $v_i = u_{i+1}$ for $i = b, \ldots, m-1$. Thus, $G_j^i(v_1, \ldots, v_i) = G_j^{i+1}(u_1, \ldots, u_i, u_{i+1})$ for $i = b, \ldots, m-1$, and, using (5.11), we have

$$F_j^m(u_1, \ldots, u_m) - F_j^{m-1}(v_1, \ldots, v_{m-1}) \geq (\alpha - 1) \sum_{i=b+1}^{m} \alpha^{i-1} G_j^i(u_1, \ldots, u_i) \geq 0.$$

If $\sum_{i=b+1}^m G_j^i(u_1, \ldots, u_i) \geq 0$ then define $v_i = \text{argmin}\{W_{\leq j-1}(u_h) | 1 \leq h \leq b-1\}$ for $i = b, \ldots, m-1$, implying that $G_j^i(v_1, \ldots, v_i) = 0$ for $i = b, \ldots, m-1$. Again

$$\begin{aligned} & F_j^m(u_1, \ldots, u_m) - F_j^{m-1}(v_1, \ldots, v_{m-1}) \\ = & \sum_{i=b}^{m} \alpha^{i-1} G_j^i(u_1, \ldots, u_i) - \sum_{i=b}^{m-1} \alpha^{i-1} G_j^i(v_1, \ldots, v_i) \ \geq \ 0. \end{aligned}$$

Now choose $v_m = O_j$. Then

$$G_j^m(v_1, \ldots, v_m) \ = \ W_{\leq j-1}(O_j) - \min_{i \leq m-1} \{ \ W_{\leq j-1}(v_i) + \beta d(v_i, O_j)\}.$$

Assume that the minimum in this expression is attained for the point $v_i = t_\ell^-$. (Notice that the multiset $\{v_1, \ldots, v_{m-1}\}$ is a subset of the multiset $\{t_1^-, \ldots, t_m^-\}$.) Again we apply (5.2) to obtain, $W_{\leq j-1}(t_\ell^-) \geq W_{\leq j-1}(O_j) - \lambda d(O_j, t_\ell^-)$. Another observation is $\textsc{Opt}(O_j, \sigma_j) \leq d(O_j, t_\ell^-) + W_j(t_\ell)$. Together they yield,

$$\begin{aligned} & F_j^{m-1}(v_1, \ldots, v_{m-1}) - F_j^m(v_1, \ldots, v_m) \\ = & -\alpha^{m-1} G_j^m(v_1, \ldots, v_m) \\ = & -\alpha^{m-1} \left(W_{\leq j-1}(O_j) - W_{\leq j-1}(t_\ell^-) - \beta d(t_\ell^-, O_j) \right) \\ \geq & \ \alpha^{m-1}(\beta - \lambda) d(t_\ell^-, O_j) \\ \geq & \ \alpha^{m-1}(\beta - \lambda)(\textsc{Opt}(O_j, \sigma_j) - W_j(t_\ell)). \end{aligned} \tag{5.12}$$

Hence,

$$\begin{aligned}
\mathcal{F}^m_{j+1} &= F^m_{j+1}(t_1,\ldots,t_m) \\
&\geq^{(5.10)} F^m_j(u_1,\ldots,u_m) + \sum_{i=1}^{m} H_i \\
&\geq F^{m-1}_j(v_1,\ldots,v_{m-1}) + H_\ell \\
&\geq^{(5.12)} F^m_j(v_1,\ldots,v_m) + \alpha^{m-1}(\beta-\lambda)(\textsc{Opt}(O_j,\sigma_j) - W_j(t_\ell)) + H_\ell \\
&\geq \mathcal{F}^m_j + \alpha^{m-1}\lambda(\textsc{Opt}(O_j,\sigma_j) - W_j(t_\ell)) + H_\ell \\
&\geq^{(5.2)} \mathcal{F}^m_j + \alpha^{m-1}\lambda(\textsc{Opt}(O_j,\sigma_j) - W_j(t_\ell)) + \alpha^{m-1}(1-\beta)W_j(t_\ell) \\
&= \mathcal{F}^m_j + \alpha^{m-1}(1-\beta-\lambda)W_j(t_\ell) + \alpha^{m-1}\lambda\textsc{Opt}(O_j,\sigma_j) \\
&\geq \mathcal{F}^m_j + \alpha^{m-1}\lambda\textsc{Opt}(O_j,\sigma_j).
\end{aligned}$$

5.3 The general two-server problem

In the general two-server problem we are given a server, to whom we will address as the x-server, moving in a metric space $\mathbb{X}$, starting from point $x_0 \in \mathbb{X}$ and a server, the y-server, moving in a metric space $\mathbb{Y}$, starting in $y_0 \in \mathbb{Y}$. Requests $(x,y) \in \mathbb{X}\times\mathbb{Y}$ are presented on-line one by one and are served by moving one of the servers to the corresponding point in its metric space. The objective is to minimize the sum of the distances travelled by the two servers. This problem can easily be modelled as a metrical service system: There is one server moving in the product space $\mathbb{X}\times\mathbb{Y}$ and any pair $(x,y) \in \mathbb{X}\times\mathbb{Y}$ defines a request $r = \{\{x\}\times\mathbb{Y}\} \cup \{\mathbb{X}\times\{y\}\}$. For any two points (x_1,y_1) and (x_2,y_2) in $\mathbb{X}\times\mathbb{Y}$ we define $d((x_1,y_1),(x_2,y_2)) = d_x(x_1,x_2) + d_y(y_1,y_2)$, where d_x and d_y are the distance function of the metric spaces $\mathbb{X}$ and $\mathbb{Y}$.

Lemma 8 *The work function and λ-WFA are well-defined for the general two-server problem.*

Proof. Let $\sigma = (x_0,y_0),\ldots,(x_n,y_n)$ be a request sequence for some general two-server problem, where we assume, without loss of generality, that the first request is given in the origin $O = (x_0,y_0)$. Consider an arbitrary path serving σ and ending in some point (x,y). If at some request both servers move on this path then the move of the server that does not serve the request can be postponed to the next request at no extra cost. Hence, there exists a path ending in (x,y) of at most the same length on which only one server is moved at each request, possibly with the exception of the last request. Since the number of paths that move only one server with each request is 2^n there exists a path ending in (x,y) that has minimal length, whence the work function is well-defined. More precisely, the endpoint of any such path is in the set $S =$

$\{x_n\} \times \{y_0, \ldots, y_{n-1}\} \cup \{x_0, \ldots, x_{n-1}\} \times \{y_n\}$. Hence $W_{O,\sigma}(x,y) = W_{O,\sigma}(s) + d(s,(x,y))$ for some $s \in S$ implying $\mathrm{Supp}(W_{O,\sigma}) \subseteq S$. Since S has only a finite number $(2n-2)$ of elements the generalized work function algorithm λ-WFA is well-defined. □

Thus, a part of the conditions of Theorem 11 is satisfied. We are still to define an algorithm A that satisfies condition (5.3). As such an algorithm we have chosen the simple BALANCE algorithm, which keeps track of the costs made by the x- and y-server and tries to balance their costs. The BALANCE algorithm is not competitive for our problem as it is known not to be competitive even for the classical two-server problem. However, we will show that it satisfies condition (5.3) with $c = 4$ and $m = 3$.

We define BALANCE starting in (x_0, y_0) and serving the request sequence $\sigma = (x_1, y_1), (x_2, y_2), \ldots$. Let S_j^x and S_j^y be the total costs made by, respectively, the x- and the y-server after the j-th request is served and let $S_j := S_j^x + S_j^y$. We denote the positions of the servers after serving request (x_j, y_j) by $(\hat{x}_j, \hat{y}_j)$.

> BALANCE
>
> If $S_j^x + d(\hat{x}_j, x_{j+1}) \leq S_j^y + d(\hat{y}_j, y_{j+1})$, then move the x-server to request x_{j+1}. Else move the y-server to request y_{j+1}.

The following lemmas give an upper bound on the cost of BALANCE. We denote by P_{ij}^x, $(0 \leq i < j)$, the length of the path $x_i, x_{i+1}, \ldots, x_j$. We denote P_{ij}^y $(0 \leq i < j)$ in a similar way.

Lemma 9 *If* BALANCE *is applied to the sequence* $(x_1, y_1), \ldots, (x_j, y_j)$ *starting from* (x_0, y_0)*, then* $S_j \leq 2\min\{P_{0j}^x, P_{0j}^y\}$ $\forall j \geq 0$.

PROOF. Clearly, $S_j^x \leq P_{0j}^x$ and $S_j^y \leq P_{0j}^y$. Let request (x_i, y_i), be the last request served by the x-server. Then, by definition, $S_j^x = S_i^x \leq S_{i-1}^y + d(\hat{y}_{i-1}, y_i) \leq P_{0i}^y \leq P_{0j}^y$. Hence, $S_j^x \leq \min\{P_{0j}^x, P_{0j}^y\}$. Similarly it is shown that $S_j^y \leq \min\{P_{0j}^x, P_{0j}^y\}$.

□

Let $\mathrm{OPT}(O, \sigma)$ denote the cost of an optimal path serving the sequence σ and starting from (x_0, y_0).

Lemma 10 *If* BALANCE *is applied to the sequence* $\sigma = (x_1, y_1), \ldots, (x_j, y_j)$ *starting from* (x_0, y_0)*, then* $S_j \leq 2\mathrm{OPT}(O, \sigma) + 4\min\{P_{1j}^x, P_{1j}^y\}$.

PROOF. If the optimal path uses only one server then the lemma follows immediately from Lemma 9 since $\mathrm{OPT}(O, \sigma) = \min\{P_{0j}^x, P_{0j}^y\} \geq S_j/2$. So assume the optimal path uses both servers and assume without loss of generality

$P^x_{1j} \leq P^y_{1j}$. Since the x-server of the optimal path serves at least one request $\textsc{Opt}(O,\sigma) \geq P^x_{01} - P^x_{1j}$. Again using Lemma 9 yields $S_j \leq 2P^x_{01} + 2P^x_{1j} \leq 2\textsc{Opt}(O,\sigma) + 4P^x_{1j} = 2\textsc{Opt}(O,\sigma) + 4\min\{P^x_{1j}, P^y_{1j}\}$. □

Lemma 11 *If s_1, s_2 and s_3 are pairwise σ-independent points with respect to the request sequence $\sigma = (x_1, y_1), \ldots, (x_j, y_j)$, then $\widehat{W}_\sigma(s_1) + \widehat{W}_\sigma(s_2) + \widehat{W}_\sigma(s_3) \geq \min\{P^x_{1j}, P^y_{1j}\}$.*

Proof. Denote by T_i $(i = 1, 2, 3)$ the path of length $\widehat{W}_\sigma(s_i)$ that serves σ and ends in s_i. Since $\widehat{W}_\sigma(s_1) + \widehat{W}_\sigma(s_2) \leq d(s_1, s_2)$ it is impossible that there are two requests such that one request is served by the x-servers of T_1 and T_2 and the other is served by the y-servers of T_1 and T_2. The same holds of course for the other two pairs T_1, T_3 and T_2, T_3. So assume without loss of generality that the x-server of T_1 shares no request with the x-server of T_2 and shares no request with the x-server of T_3. If the y-server of T_1 serves all requests then the lemma obviously holds. So assume request (x_i, y_i) is served by the x-server of T_1. Then T_2 and T_3 must serve point y_i. But this means that the x-servers of T_2 and T_3 do not share a request. Hence, the three x-servers share no request. Thus, each request is served by at least two y-servers and for each two consecutive requests there is a y-server that serves both requests. □

Combining Lemma 10, Lemma 11 and Theorem 11 yields the next result.

Corollary 4 *Algorithm* Online(Balance, γ, λ) *with $\gamma = 4(1-\lambda)/(1-2\lambda)$ is $7/(\lambda(1/2-5\lambda))^2$-competitive for any instance of the general two-server problem, for any $0 < \lambda < 1/10$.*

The ratio is minimized by $\lambda = 1/20$, yielding a competitive ratio of 44,800.

5.4 Postlude

Unfortunately, Theorem 11 does not provide a competitive algorithm for the general k-server problem for $k \geq 3$ just as easily as for $k = 2$. The question whether a competitive algorithm exists for $k \geq 3$ remains unresolved. We believe that the generalized work function is competitive for the general k-server problem for any fixed $k \geq 1$ and $\lambda < 1$.

It is not hard to prove that the generalized work function algorithm for the general two-server problem satisfies Lemma 10 up to a constant factor. This does not imply that the generalized work function algorithm is competitive for the general two-server problem, since the work function in this part of the Online

algorithm is defined for the requests within the phase. This is different from the work function used in the last move of ONLINE in a phase, which is defined for the complete sequence. It remains open to prove competitiveness of the generalized work function algorithm. We notice that our proof does not use any information about the work function that is specific for the general two-server problem. To make this step it seems essential to have a better understanding of the structure of the work function of the general two-server problem.

A problem to which Theorem 11 applies directly is the k-client problem. In this metrical service system the set of possible requests consists of all k-element subsets of the metric space. Burley [18] showed that the generalized work function algorithm is $O(k2^k)$-competitive when λ is chosen appropriately for each k. Our proof has many similarities with the proof of Burley. Notice that if we choose $m = k + 1$ in Theorem 11 the condition of this theorem is satisfied for any algorithm A since for any sequence σ at most k pairwise σ-independent points exist. Therefore we can restrict the phases in ONLINE to the λ-WFA-moves. This direct application of the theorem only shows that λ-WFA is $O(k^2 2^k)$-competitive for appropriate λ. However, adjusting the proof of Theorem 11 to the k-client problem gives exactly Burley's proof.

Condition (5.3) of Theorem 11 is imposed on an algorithm A. It would be more interesting to give a (non-trivial) sufficient condition for a metrical service system to have a finite competitive ratio. Even more interesting would it be to know under what condition on the system λ-WFA is competitive.

We conclude with the NP-hardness proof of the off-line version of the general k-server problem. The proof is straightforward from the EXACT 3-COVER problem. We first remind the reader to the definition of the latter problem. For NP-completeness of this problem we refer to [50].

EXACT 3-COVER:
An instance is given by a set Z with $|Z| = 3q$ and a collection C of 3-element subsets of Z. The question is whether C contains an exact cover for Z, i.e. a subcollection $C' \subseteq C$ such that every element of Z occurs in exactly one member of C'.

Theorem 12 *The general k-server problem is NP-hard.*

PROOF. Take any instance I of the exact 3-cover problem (with the notation as defined above) we define an instance I' of the general k-server problem. For each 3-element subset we define a metric space with its server. The $|C|$ metric spaces are identical and consist of an origin, one point at distance 1 from the origin, and one point at distance $q + 1$ from the origin. For each element of Z we define one request, i.e. a $|C|$-tuple of points, one in each of the $|C|$ metric spaces. In each metric space the three requests that correspond to the 3-element

subset are given at the point at distance 1. The remaining $3q - 3$ requests are given in the point at distance $q + 1$. Instance I' has a solution with value of at most q if and only if I has an exact cover for Z. $\square$

Bibliography

[1] F. Afrati, E. Bampis, C. Chekuri, D. Karger, C. Kenyon, S. Khanna, I. Milis, M. Queyranne, M. Skutella, C. Stein, and M. Sviridenko, *Approximation schemes for minimizinge average weighted completion time with release dates*, Proc. 40th Symp. Foundations of Computer Science, New York, NY, USA, IEEE Computer Society, 1999, pp. 32–44.

[2] F. Afrati, S. Cosmadakis, C.H. Papadimitriou, G. Papageorgiou, and N. Papakostantinou, *The complexity of the travelling repairman problem*, RAIRO Journal on Information Theory and Applications **20** (1986), 79–87.

[3] E.J. Anderson and C.N. Potts, *On-line scheduling of a single machine to minimize total weighted completion time*, Proc. 13th Symp. on Discrete Algorithms, San Francisco, California, 2002, pp. 548–557.

[4] A. Archer and D.P. Williamson, *Faster approximation algoriths for the minimum latency problem*, Proc. 14th Symp. on Discrete Algorithms, Baltimore, Maryland, 2003, pp. 88–96.

[5] S. Arora, *Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems*, Journal of the ACM **45** (1998), 753–782.

[6] S. Arora and G. Karakostas, *A* $2+\epsilon$*-approximation for the k-mst problem*, Proc. 11th Symp. on Discrete Algorithms, San Francisco, California, 2000, pp. 754–759.

[7] ———, *Approximation schemes for minimum latency problems*, SIAM Journal on Computing **32** (2003), 1317–1337.

[8] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy, *Proof verification and hardness of approximation problems*, Journal of the ACM (1998), 501–555.

[9] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi, *Complexity and approximation; combinatorial optimization problems and their approximability properties*, Springer, Berlin, 1999.

[10] G. Ausiello, S. Leonardi, and A. Marchetti-Spaccamela, *On salesmen, repairmen, spiders, and other traveling agents*, Proc. 4th Italian Conference on Algorithms and Complexity, Lecture Notes in Computer Science, vol. 1767, Rome, Italy, Springer, 2000, pp. 1–16.

[11] I. Averbakh and O. Berman, *Sales-delivery man problems on treelike networks*, Networks **25** (1995), 45–58.

[12] A. Blum, P. Chalasani, D. Coppersmith, W. Pulleyblank, P. Raghavan, and M. Sudan, *The minimum latency problem*, Proc. 26th ACM Symposium on Theory of Computing, Montreal, Quebec, Canada, 1994, pp. 163–171.

[13] A. Borodin, N. Linial, and M. Saks, *An optimal online algorithm for metrical task system*, Journal of the ACM **39** (1992), 745–763.

[14] P. Brucker, Personal communication, 2001.

[15] P. Brucker and S.A. Kravchenko, *Preemption can make parallel machine scheduling problems hard*, Reihe P (Preprints) 211, Osnabruecker Schriften zur Mathematik, 1999.

[16] P. Brucker, S.A. Kravchenko, and Y.N. Sotskov, *Preemptive job-shop scheduling problems with a fixed number of jobs*, Mathematical Methods of Operations Research **49** (1999), 41–76.

[17] J. Bruno, Jr. E.G. Coffman, and R. Sethi, *Scheduling independent tasks to reduce mean finishing time*, Communications of the ACM **17** (1974), 382–387.

[18] W.R. Burley, *Traversing layered graphs using the work function algorithm*, Journal of Algorithms **20** (1996), 479–511.

[19] S. Chakrabarti, C.A. Phillips, A.S. Schulz, D.B. Shmoys, C. Stein, and J. Wein, *Improved scheduling algorithms for minsum criteria*, Proc. 23rd International Colloquium on Automata, Languages, and Programming, Lecture Notes in Computer Science, vol. 1099, Springer, 1996, pp. 646–657.

[20] K. Chaudhuri, B. Godfrey, S. Rao, and K. Talwar, *Paths, trees, and minimum latency tours*, Proc. 44nd Symp. Foundations of Computer Science, Cambridge, Massachusettes, 2003, pp. 36–45.

[21] C. Chekuri, R. Motwani, B. Natarajan, and C. Stein, *Approximation techniques for average completion time scheduling*, SIAM Journal on Computing **31** (2001), 146–166.

[22] M. Chrobak, H. Karloff, T.H. Payne, and S. Vishwanathan, *New results on server problems*, SIAM Journal on Discrete Mathematics **4** (1991), 172–181.

[23] M. Chrobak and L.L. Larmore, *An optimal online algorithm for k servers on trees*, SIAM Journal on Computing **20** (1991), 144–148.

[24] _____, *Metrical service systems: Deterministic strategies*, Tech. Report UCR-CS-93-1, Department of Computer Science, University of California at Riverside, 1992.

[25] M. Chrobak and J. Sgall, *The weighted 2-server problem*, Proc. 17thSymp. on Theoretical Aspects of Computer Science, Lecture Notes in Computer Science, vol. 1770, Springer, 2000, pp. 593–604.

[26] R.W. Conway, W.L. Maxwell, and L.W. Miller, *Theory of scheduling*, Addison-Wesley, Reading, Massachusetts, 1967.

[27] S.A. Cook, *The complexity of theorem-proving procedures*, Proc. 3th Symp. Theory of Computing, 1971, pp. 151–158.

[28] J. Du, J.Y.-T. Leung, and G.H. Young, *Minimizing mean flow time with release time constraint*, Theoretical Computer Science **75** (1990), 347–355.

[29] M.E. Dyer and L.A. Wolsey, *Formulating the single machine sequencing problem with release dates as a mixed integer program*, Discrete Applied Mathematics **26** (1990), 255–270.

[30] L. Epstein and R. van Stee, *Lower bounds for on-line single-machine scheduling*, Proc. 26th Symp. on Mathematical Foundations of Computer Science, Lecture Notes in Computer Science, vol. 2136, Springer, 2001, pp. 338–350.

[31] J. Fakcharoenphol, C. Harrelson, and S. Rao, *The k-traveling repairman*, Proc. 14th Symp. on Discrete Algorithms, Baltimore, Maryland, 2003, pp. 646–654.

[32] E. Feuerstein, S. Seiden, and A. Strejlevich de Loma, *The related server problem*, Unpublished manuscript, 1999.

[33] A. Fiat and M. Ricklin, *Competitive algorithms for the weighted server problem*, Theoretical Computer Science **130** (1994), 85–99.

[34] A. García, P. Jodrá, and J. Tejel, *A note on the travelling repairman problem*, Networks **40** (2002), 27–31.

[35] M.R. Garey and D.S. Johnson, *Complexity results for multiprocessor scheduling under resource constraints*, SIAM Journal on Computing **4** (1975), 397–411.

[36] ———, *Strong NP-completeness results: motivation, examples and implications*, Journal of the ACM **25** (1978), 499–508.

[37] ———, *Computers and intractability: A guide to the theory of NP-completeness*, Freeman and Company, San Francisco, 1979.

[38] N. Garg, *A 3-approximation for the minimum tree spanning k vertices*, Proc. 37th Symp. Foundations of Computer Science, Burlington, Vermont, USA, IEEE Computer Society, 1996, pp. 302–309.

[39] M.X. Goemans, *A supermodular relaxation for scheduling with release dates*, Proc. 5th Int. Conf. Integer Programming and Combinatorial Optimization (W.H. Cunningham, S.T. McCormick, and M. Queyranne, eds.), Lecture Notes in Computer Science, vol. 1084, Springer, 1996, pp. 288–300.

[40] ———, *Improved approximation algorithms for scheduling with release dates*, Proc. 8th Symp. on Discrete Algorithms, New Orleans, Louisiana, United States, 1997, pp. 591–598.

[41] M.X. Goemans and J. Kleinberg, *An improved approximation ratio for the minimum latency problem*, Mathematical Programming **82** (1998), 111–124.

[42] M.X. Goemans, M. Queyranne, A.S. Schulz, M. Skutella, and Y. Wang, *Single machine scheduling with release dates*, SIAM Journal on Discrete Mathematics **15** (2002), 165–192.

[43] M.X. Goemans, J.M. Wein, and D.P. Williamson, *A 1.47-approximation algorithm for a preemptive single-machine scheduling problem*, Operations Research Letters **26** (2000), 149–154.

[44] T. Gonzalez, *Optimal mean finish time preemptive schedules*, Technical Report 220, Computer Science Department, Pennsylvania State University, 1977.

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

[46] L.A. Hall, A.S. Schulz, D.B. Shmoys, and J. Wein, *Scheduling to minimize average completion time: Off-line and on-line approximation algorithms*, Mathematics of Operations Research **22** (1997), 513–544.

[47] M. Held and R.M. Karp, *A dynamic programming approach to sequencing problems*, SIAM Journal on Applied Mathematics **10** (1962), 196–210.

[48] J.A. Hoogeveen and A.P.A. Vestjens, *Optimal on-line algorithms for single-machine scheduling*, Proc. 5th Int. Conf. Integer Programming and Combinatorial Optimization (W.H. Cunningham, S.T. McCormick, and M. Queyranne, eds.), Lecture Notes in Computer Science, vol. 1084, Springer, 1996, pp. 404–414.

[49] W.A. Horn, *Minimizing average flow time with parallel machines*, Operations Research **21** (1973), 846–847.

[50] R.M. Karp, *Reducibility among combinatorial problems*, Complexity of computer computations (R.E. Miller and J.W. Thatcher, eds.), Plenum Press, New York, 1972, pp. 85–103.

[51] E. Koutsoupias and C.H. Papadimitriou, *On the k-server conjecture*, Journal of the ACM **42** (1995), 971–983.

[52] E. Koutsoupias, C.H. Papadimitriou, and M. Yannakakis, *Searching a fixed graph*, Proc. 23rd International Colloquium on Automata, Languages, and Programming, Lecture Notes in Computer Science, vol. 1099, Paderborn, Germany, Springer, 1996, pp. 280–289.

[53] E. Koutsoupias and D. Taylor, *The* CNN *problem and other k-server variants*, Proc. 17thSymp. on Theoretical Aspects of Computer Science, Lecture Notes in Computer Science, vol. 1770, Springer, 2000, pp. 581–592.

[54] S.A. Kravchenko and Y.N. Sotskov, *Optimal makespan schedule for three jobs on two machines*, Mathematical Methods of Operations Research **43** (1996), 233–238.

[55] S.O. Krumke, W.E. de Paepe, D. Poensgen, and L. Stougie, *News from the online traveling repairman problem*, Theoretical Computer Science **295** (2003), 279–294.

[56] J. Labetoulle, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan, *Preemptive scheduling of uniform machines subject to release dates*, Progress in Combinatorial Optimization (W. R. Pulleyblank, ed.), Academic Press, 1984, pp. 245–261.

[57] E.L. Lawler, *Recent results in the theory of machine scheduling*, Mathematical Programming: the State of the Art (M. Grötchel A. Bachem and B. Korte, eds.), Springer, Berlin, 1983, pp. 202–234.

[58] E.L. Lawler and J. Labetoulle, *On preemptive scheduling of unrelated parallel processors by linear programming*, Journal of the ACM **25** (1978), 612–619.

[59] J.K. Lenstra, A.H.G. Rinnooy Kan, and P. Brucker, *Complexity of machine scheduling problems*, Annals of Discrete Mathematics **1** (1977), 343–362.

[60] X. Lu, R.A. Sitters, and L. Stougie, *A class of on-line scheduling algorithms to minimize total completion time*, Operations Research Letters **31** (2002), 232–236.

[61] M. Manasse, L.A. McGeoch, and D. Sleator, *Competitive algorithms for server problems*, Journal of Algorithms **11** (1990), 208–230.

[62] R. McNaughton, *Scheduling with deadlines and loss functions*, Management Science **6** (1959), 1–12.

[63] N. Megow, *Performance analysis of on-line algorithms in machine scheduling*, Diplomarbeit, Institut für Mathematik der Technische Universität Berlin, April 2002.

[64] N. Megow, R.A. Sitters, and L. Stougie, Private communication, 2003.

[65] E. Minieka, *The delivery man problem on a tree network*, Annals of Operations Research **18** (1989), 261–266.

[66] W.E. de Paepe, J.K. Lenstra, J. Sgall, R.A. Sitters, and L. Stougie, *Computer-aided complexity classification of dial-a-ride problems*, SPOR-report, Technische Universiteit Eindhoven, 2003, to appear in INFORMS J. on Computing.

[67] C.H. Papadimitriou and M. Yannakakis, *Optimization, approximation, and complexity classes*, Journal of Computer and System Sciences **43** (1991), 425–440.

[68] ———, *The traveling salesman problem with distances one and two*, Mathematics of Operations Research **18** (1993), 1–11.

[69] C. Phillips, C. Stein, and J. Wein, *Scheduling jobs that arrive over time*, Proc. of the 4th. Workshop on Algorithms and Data Structures, Lecture Notes in Computer Science, vol. 955, Springer, 1995, pp. 86–97.

[70] ———, *Minimizing average completion time in the presence of release dates, networks and matroids; sequencing and scheduling*, Mathematical Programming **82** (1998), 199–223.

[71] M. Queyranne, *On the Anderson-Potts single machine on-line scheduling algorithm*, Unpublished manuscript, 2001.

[72] S. Sahni and T. Gonzalez, *P-complete approximation problems*, Journal of the ACM **23** (1976), 555–565.

[73] L. Schrage, *A proof of the shortest remaining processing time processing discipline*, Operations Research **16** (1968), 687–690.

[74] A.S. Schulz, *Scheduling to minimize total weighted completion time: Performance guarantees of LP-based heuristics and lower bounds*, Proc. 5th Int. Conf. Integer Programming and Combinatorial Optimization (W.H. Cunningham, S.T. McCormick, and M. Queyranne, eds.), Lecture Notes in Computer Science, vol. 1084, 1996, pp. 301–315.

[75] A.S. Schulz and M. Skutella, *The power of α-points in single machine scheduling*, Journal of Scheduling **5** (2002), 121–133.

[76] ———, *Scheduling unrelated machines by randomized rounding*, SIAM Journal on Discrete Mathematics **15** (2002), 450–469.

[77] J. Sgall, *On-line scheduling*, Online algorithms: the state of the art (A. Fiat and G.J. Woeginger, eds.), Lecture Notes in Computer Science, vol. 1442, Springer, 1998, pp. 196–231.

[78] D. Simchi-Levi and O. Berman, *Minimizing the total flow time of n jobs on a network*, IIE Transactions **23** (1991), 236–244.

[79] R.A. Sitters, *Two NP-hardness results for preemptive minsum scheduling of unrelated parallel machines*, Proc. 8th Int. Conf. Integer Programming and Combinatorial Optimization (Karen Aardal and Bert Gerards, eds.), Lecture Notes in Computer Science, vol. 2081, Springer, 2001, pp. 396–405.

[80] ———, *The minimum latency problem is NP-hard for weighted trees*, Proc. 9th Int. Conf. Integer Programming and Combinatorial Optimization (William J. Cook and A.S. Schulz, eds.), Lecture Notes in Computer Science, vol. 2337, Springer, 2002, pp. 230–239.

[81] ———, Unpublished manuscript, 2004.

[82] W.E. Smith, *Various optimizers for single-stage production*, Naval Research Logistics Quarterly **3** (1956), 59–66.

[83] R. van Stee and H. La Poutré, *Minimizing total completion time on-line on a single machine, using restarts*, Proc. 10th European Symp. on Algorithms (R. Möhring and R. Raman, eds.), Lecture Notes in Computer Science, vol. 2461, Springer, 2002, pp. 872–883.

[84] J.N. Tsitsiklis, *Special cases of traveling salesman and repairman problems with time windows*, Networks **22** (1992), 263–282.

[85] V.V. Vazirani, *Approximation algorithms*, Springer, Berlin, 2001.

[86] A.P.A. Vestjens, *On-line machine scheduling*, Ph.D. thesis, Department of Mathematics and Computing Science, Technische Universiteit Eindhoven, Eindhoven, the Netherlands, 1997.

[87] I.R. Webb, *Depth-first solutions for the deliveryman problem on tree-like networks: an evaluations using a permutation model*, Transportation Science **30** (1996), 134–147.

[88] B.Y. Wu, *Polynomial time algorithms for some minimum latency problems*, Information Processing Letters **75** (2000), 225–229.

Samenvatting

Een klassiek probleem op het gebied van de combinatorische optimalisering is het *handelsreizigersprobleem*. Stel dat een handelsreiziger een lijst met klanten heeft die hij wil bezoeken om zijn handelswaar te verkopen, alvorens weer naar huis terug te keren. Het probleem dat deze persoon zichzelf stelt is om de volgorde van bezoek zodanig te kiezen dat de totaal afgelegde afstand op zijn reis minimaal is. In theorie modelleren we dit probleem als het vinden van de kortste tour door een gegeven verzameling punten met hun onderlinge afstanden. Optimaliseringsproblemen van deze vorm komen in de praktijk op grote schaal voor, bijvoorbeeld in de transportwereld, maar ook bij de assemblage van produkten. Het is daarom ook niet verwonderlijk dat er binnen de combinatorische optimalisering veel onderzoek is gedaan naar het handelsreizigersprobleem, welk een eenvoudig wiskundig model is voor problemen die in de praktijk vaak veel complexer zijn. Kennis van elementaire wiskundige modellen is van essentieel belang om overweg te kunnen met de veel complexere modellen die nodig zijn voor problemen uit de praktijk.

In dit proefschrift behandelen we een aantal elementaire problemen uit twee gebieden binnen de combinatorische optimalisering: het gebied van *routeringsproblemen* en het gebied van *machine-volgordeproblemen*. Het handelsreizigersprobleem is een voorbeeld van een routeringsprobleem. Bij het andere gebied kunnen we denken aan een machine die een reeks opdrachten één voor één moet uitvoeren. Van elke opdracht veronderstellen we dat het moment waarop deze voor bewerking beschikbaar komt en de bewerkingsduur bekend is. Het moment waarop de bewerking van een opdracht voltooid is noemen we de completeringstijd. Het probleem is om de volgorde van bewerking te bepalen die de som van alle completeringstijden minimaliseert.

Indien alle informatie bekend is bij het zoeken naar een oplossing, zoals bij de twee geschetste problemen, noemen we het probleem *statisch*. Daartegenover is het voor veel problemen reëel dat de oplossing stap voor stap wordt gevormd en dat informatie zoals bedieningsduur of de lijst van klanten, tijdens het vormen van de oplossing geleidelijk bekend wordt. In dat geval noemen we het probleem *dynamisch*.

Voor de meeste statische problemen geldt dat een beetje vernuft en een eindeloos geduld voldoende zijn voor het vinden van een optimale oplossing. Aangezien men dit geduld in de praktijk vaak niet heeft, beperkt men zich in de theorie vaak tot zogenaamde *polynomiale algoritmen*. Dit zijn stap voor stap gedefinieerde oplossingsmethoden waarin het aantal stappen relatief klein is. Met deze beperking is het een stuk moeilijker om een algoritme te vinden welke gegarandeerd altijd een optimale oplossing vindt. Voor dynamische problemen is het vaak zelfs onmogelijk om altijd een optimale oplossing te vinden. In dit proefschrift richten we ons op *benaderingsgaranties*. Een algoritme heeft een benaderingsgarantie α als het voor iedere, willekeurig gekozen, instantie een oplossing geeft waarvan dat waarde niet meer dan een constante α maal de optimale waarde is. We doen zowel positieve als negatieve uitspraken en noemen hier de drie belangrijkste resultaten.

Het *traveling repairman probleem* is een variant van het handelsreizigersprobleem waarbij de doelstelling niet het minimaliseren van de reistijd van de handelsreiziger is, maar het minimaliseren van de gemiddelde aankomsttijd bij de punten. Het is bekend dat het bestaan van een optimaal ($\alpha = 1$) polynomiaal algoritme uiterst onwaarschijnlijk is. We bewijzen hier dat dit ook geldt voor een eenvoudigde versie van het probleem waarin de afstanden tussen de punten een eenvoudige structuur heben in de vorm van een boom-graaf. Een gelijke negatieve stelling geven we ook voor een machine-volgordeprobleem op *ongerelateerde parallelle* machines. In dit model zijn er verscheidene machines beschikbaar voor het bewerken van de opdracht. We mogen nu de bewerking meerdere malen onderbreken en door verscheidene machines na elkaar uitvoeren totdat de opdracht is voltooid. De bewerkingssnelheid hangt hierbij af van de opdracht en de machine. Wij laten dus zien dan het toestaan van preemptie het vinden van een optimale oplossing moeilijker maakt. Een positieve stelling geven we voor het *twee-server probleem*. In dit dynamische probleem krijgen twee servers steeds elk een punt te zien waarvan er slechts één door de betreffende server bezocht moet worden, voordat het volgende paar bekend wordt. Het doel is het minimaliseren van de gezamelijk afgelegde afstand. We geven voor dit probleem een algoritme met constante benaderingsgarantie α. Vooraf-gaand geven we een algemener resultaat, dat voor een grote klasse van on-line routeringsproblemen een voldoende (en noodzakelijke) voorwaarde geeft voor het bestaan van algoritmen met constante-factor benaderingsgaranties.

Biography

René Sitters was born on March 15th. 1972 in Alkmaar, the Netherlands. He studied applied mathematics at the University of Twente and graduated in 1999 on the subject of graph coloring problems related to frequency assignment, under supervision of Hajo Broersma and Cees Hoede. From September 1999 to December 1999 he was teaching at the University of Twente.

In February 2000 he started as a PhD student at the Eindhoven University of Technology under supervision of Leen Stougie and Jan Karel Lenstra. In March and April 2003, and March and April 2004 he visited the university of Rome, La Sapienza. In September 2004 he will take up a postdoctoral fellowship at the Max-Panck-Institut für Informatik in Saarbrücken, Germany.