Introduction

In combinatorial optimization one wants to find the best solution among many possible choices. The set of candidates is usually given implicitly by the input data and often has exponential size in comparison to the input. The aim is to construct an efficient algorithm which computes the best solution or a solution of provably good quality. A very important topic in combinatorial optimization is scheduling. It treats the assignment of limited resources to activities. In the actual applications, the resources can be very diverse, e. g., machines, money, gasoline, teachers, cars, watchmen, etc. Examples for the many imaginable activities are producing industrial goods, guarding a building, delivering parcels, executing a computer program, working on a project, or the classes of a university. The goal is to assign the resources to the activities in order to optimize some performance measure. Such a measure could be the time when the last activity finishes or the total delay of all activities.

The problems studied in this thesis can be understood as *machine scheduling problems*. In machine scheduling, one is usually given a set of jobs with certain processing times which need to be assigned to some given machines. One has to compute a schedule such that each job is assigned and each machine processes at most one job at a time. Often, additional constraints are present. For instance, jobs could have a deadline by which they have to be finished, some jobs might not be able to start before some others have finished (precedence constraints), or some jobs might not be available before a certain given time (release dates).

A broad range of applications can be understood as machine scheduling problems. A simple example is the manufacturing process of industrial goods in a factory. The jobs model the manufacturing steps which need to be done on the different processing units. The latter are modeled by the machines. However, also the organization of a garden party fits into the machine scheduling framework. The preparation steps (putting on the barbecue, prepare salads, invite friends, etc.) are modeled by jobs whereas the organizers are modeled by the machines. Finally, finding a timetable for the classes of a university and their assignment to lecture theaters can also be formalized as a machine scheduling problem.

For evaluating a computed schedule we need a suitable quality measure. This measure can vary significantly depending on the actual application. For example, two common objective functions in machine scheduling are the *sum of weighted completion times* and the *makespan*. For applications like the pro-

duction of cars in a factory it is desirable to finish as many units as early as possible. To this end, a good objective function is the sum of the completion times of all jobs. If some jobs are more important than others one can additionally give each job a weight which reflects its significance. The resulting objective function is then the sum of the weighted completion times. When scheduling the work of a project or for the mentioned garden party one might rather want that the overall schedule finishes as early as possible. The time when the last job finishes is defined as the *makespan* of a schedule which is a good objective function in this case. In other settings such as time-tabling there might be no objective function necessary and one only wants to compute *some* feasible schedule. However, in such settings it is likely that there are also some not strictly necessary but still desirable "soft" constraints which can be modeled by a suitable objective function.

One very fundamental scheduling problem is the *packet routing problem* (even though in the literature it is usually referred to as a routing problem, see [5, 65, 92], it can also be understood as a scheduling problem). In computer networks like the internet huge amounts of data need to be transported. In particular, due to applications like video-streaming and Voice over IP (VoIP) the internet traffic has grown significantly in the last few years. In computer networks, the transported data is split into packets. Those form the atoms of the network communication. Each packet needs to be transported from its origin to its destination. Since the bandwidth of the communication links is limited, the network might need to delay packets. This results in a scheduling problem. The packets form jobs which need to be "processed" by the links on its path. The limited bandwidth is captured by the fact that each link/machine can process only one packet (or maybe some constant number of packets) at a time.

In some scheduling applications jobs are created repeatedly. For instance, consider an on-board computer of a modern airplane. In a repeated fashion, the computer executes jobs like checking the altitude, operating the auto-pilot, etc. For flight safety it is crucial that each operation finishes before its deadline. Already a small delay in a critical operation might endanger the aircraft and its passengers. Therefore, one needs a (mathematical) proof that a system operates according to its specifications at all times. The area of *real-time scheduling* provides the mathematical foundation for such proofs. The operations are modeled by *tasks* which continuously generate new jobs. Since the schedule is assumed to run infinitely long one cannot compute it explicitly. Instead, one is interested in *scheduling policies* which decide at every point in time which jobs are executed on the machines. Given such a scheduling policy one needs to prove that every job which is ever created meets its deadline.

Almost all problems which we consider in this thesis are NP-hard. Since it is widely believed that $P \neq NP$, there is not much hope for efficient (polynomial time) algorithms which solve our problems exactly. Therefore, we relax the aim of always finding an optimal solution and search for *approximation algorithms* instead. For this thesis, an α-approximation algorithm is a polynomial time algorithm which computes solutions whose objective values differ by at most a factor of α from the respective optimum. We refer to α as the *approximation factor* or *performance ratio* of an algorithm.

At first glance, approximation algorithms might not seem to be relevant for practical applications. For instance, a solution whose value differs by a factor of two from the optimum is by far not satisfying if it has an impact on a budget of thousands of euros. However, the performance ratio of an algorithm is always based on a worst-case analysis. Hence, there could be (and usually there are) many instance of the respective problem where the algorithm performs much better than in the worst-case scenario. Also, for exact methods like IP-solvers it is often useful to have primal solutions of good quality which help pruning the branch-and-bound tree. Apart from being heuristics that one could use directly, approximation algorithms usually yield important structural properties of the problem. These properties can be exploited in approaches to solve the problem exactly (with more computational effort). A good example is our work on the periodic maintenance problem which we present in Chapter 6. We designed approximation algorithms for the several settings of the problem, especially for the practical relevant harmonic case with pairwise dividing period lengths. The gained structural insights then allowed us to develop an IP-formulation for the problem which was able to solve all instances of our industrial partner optimally [28]. Straightforward approaches without the additional insights failed to solve instances of real-world size.

Outline of the Thesis

In Part I of this thesis we study the packet routing problem. As described above, it is a very fundamental question in computer networks. However, it is by far not fully understood theoretically. The best known approximation algorithm produces a schedule whose length is bounded by $O(C + D)$ [66]. The congestion C and the dilation D form the two natural lower bounds on the length of each schedule. One can argue that with the constructive proof for the Lovász Local Lemma (LLL) [79] the proof by Scheideler [92] yields an algorithm computing schedules which finish after at most $39(C + D)$ steps. The above results almost completely rely on the LLL which – due to its generality – cannot make use of the entire structure of the problem. In this work, we make a step towards a better theoretical understanding of the important properties of the problem. We present approximation algorithms, bounds on the makespan of optimal schedules, and complexity results. In particular, we show how structural insights help improving the above LLL-based results.

Chapter 1: The general packet routing problem is very complex. However, when the underlying graph topology is well-structured – like a tree – one can design better and simpler algorithms. In this chapter, we study the packet routing problem on instances where the given graph is a tree. The gained insights will be very useful later in Chapter 2 where we study general graphs. We show that the straightforward farthest-destination-first (FDF) algorithm has an arbitrarily large approximation factor, even on directed trees. Hence, more sophisticated algorithmic methods are needed. For undirected trees we present

a 2-approximation algorithm. For directed trees we derive structural properties which allow us to design an algorithm computing direct schedules (which delay packets only in their respective origins) which finish within $C + D - 1$ steps. Hence, this algorithm is also a 2-approximation algorithm, but it yields a much better performance guarantee if $C \gg D$ or $D \gg C$. Finally, we derive a general condition for the existence of a direct schedule with a certain makespan, even for general directed graphs. We published the results of this chapter in [84].

Chapter 2: We broaden our scope to general graphs. It is known that there is always a schedule which finishes within $39(C+D)$ steps [92]. No improvement on this bound has been made in more than 10 years, even though it is a very natural and important question. We improve and generalize the previous results for the problem. First, we improve the above bound to $23.4(C + D)$. Moreover, we generalize the problem by allowing arbitrary bandwidths and arbitrary transit times for the edges. If every link in the network has at least a certain bandwidth and/or transit time we prove even better bounds. For example, if every link has a transit time of at least 63 we obtain a bound of $4.32(C + D)$ for the makespan of the optimal schedule. We derive these results with a novel framework. Once one has established a good bound for the makespan of optimal schedules for instances with small dilation (i. e., each packet travels only along a small number of edges) our framework implies a bound for *all* instances. For deriving a bound for instances with small dilation we use the insights gained in Chapter 1. Moreover, our framework has the potential to give even better bounds using further insights for such instances. See [88] for our conference paper which contains these results. Due to the mentioned recent result of Moser and Tardos [79] we even obtain an algorithm which computes a schedules with the above bounds.

Chapter 3: After having studied algorithms for the packet routing problem, in this chapter we analyze its complexity. It is known that the problem is NP-hard (implicitly in [25]). We show that it is even NP-hard to approximate the problem with a factor better than $6/5$. In particular, this rules out the existence of a PTAS, assuming that $P \neq NP$. Even more, we prove the latter also for the special case that the underlying graph is a directed tree. For this very restricted graph class we show that the existence of an approximation algorithm with a factor better than $8/7$ implies $P = NP$. This surprising result underlines the difficulty of the packet routing problem since most NP-hard optimization problems (e. g., Coloring, Independent Set, etc.) become polynomial time solvable if one restricts the input graphs to trees. The results of this chapter were published in [84].

Chapter 4: So far we studied the packet routing problem only in the static setting where a finite set of packets needs to be transported to their respective destinations. However, in applications like live video-streams or Voice Over IP (VoIP) usually arbitrarily many packets are created repeatedly. This requires a different model. Therefore, in this chapter we study the *periodic packet routing problem*. The input consists of tasks (rather than single packets) which continuously generate new packets. We assume an infinite time horizon. Hence, we cannot compute the actual schedule explicitly. Instead, we need to design

scheduling policies which define the prioritization of the packets at runtime. We study two paradigms for these policies, *template schedules* and *priority schedules*. Priority schedules are an adaption of rate-monotonic schedules as they are known in classical real-time scheduling. Template schedules are a class of schedules which are especially designed for periodic packet routing. We give a comprehensive characterization of these paradigms. We present algorithms which compute schedules of the respective types and prove (almost) matching lower bounds for the potential of the two paradigms. We refer to [86] for our paper with the results of this chapter.

In Part II of this thesis, we study three other scheduling problems, the flow scheduling problem, the periodic maintenance problem, and finally the problem of scheduling jobs on unrelated machines.

Chapter 5: Dynamic flows and scheduling are two important areas of combinatorial optimization which in practice often arise in a combined manner. However, in theory both problems are mostly treated separately. Better practical solutions need thorough theoretical knowledge. Therefore, in this chapter we make a step towards a better understanding of the interaction of the two areas by studying the flow scheduling problem. In that problem, we want to transport items (jobs) through a network by a dynamic flow from a source to a sink. The objective is to minimize the sum of weighted arrival (completion) times at the sink. We first establish the connection between the routing and the scheduling aspect of this problem. Then we show that the scheduling part of the problem reduces to the problem of scheduling jobs on a single machine whose speed might increase over time (the objective is still to minimize the sum of weighted completion times). We treat this interesting problem in its own right. In contrast to the case where the speed of the machine stays constant, the natural Smith's Rule algorithm is not always optimal in this setting. However, we show that it is exactly a $\big((\sqrt{3}+1)/2\big)$-approximation algorithm. Usually in approximation algorithms, one establishes a lower bound against which one compares the value of the computed solution. Here we pursue a different approach. We constructively characterize properties of worst-case instances. We do this so precisely, that finally computing the worst-case approximation ratio of Smith's Rule on these instances reduces to basic calculus. This procedure gives us the exact approximation ratio of the algorithm and a family of tight examples. In addition, we generalize the PTAS for the unit speed case with release dates [3] to our problem with release dates. Studying the online setting, we show that here Smith's Rule is still a 2-approximation algorithm. We round up the picture by showing certain other helpful properties of the problem which yield some polynomial time solvable special cases. The results of this chapter have been published in [102].

Chapter 6: As part of our collaboration with our industrial partner, a major avionics company, we study the periodic maintenance problem (PMP). As described above, in a modern aircraft the flight control is to a great extent automatically done by the on-board computer. Therefore, it is crucial that the computer works according to its specifications at all times. The computer programs are modeled by tasks which periodically need processing time on the

processors that they are assigned to. The scheduling rule is very conservative: Once a task has released a new job, this job has to be processed immediately without any delay or preemption. The only degrees of freedom are the assignment of the tasks to processors and the initial time-offsets for the tasks. Even though real-time scheduling is a very active field of research, this particular problem has not been studied much. It arises at our industrial partner and is likely to appear in similar settings where very conservative scheduling rules are necessary. So far, it lacked the theoretical understanding of the properties which are needed for real-world solutions. In this chapter, we study the problem from a theoretical perspective and give a comprehensive characterization of its approximation landscape in the various settings. In particular, we study the practical relevant case of harmonic periods where the period lengths of the tasks divide each other pairwise. This setting can be understood as a generalization of the well-known Bin-Packing problem. Therefore, studying the problem is not only interesting from a practical point of view but also yields interesting theoretical insights. It turns out that important algorithmic properties of Bin-Packing do not generalize to the PMP. Among other results, using involved analytic tools we present a 2-approximation algorithm for the harmonic case and a tightly matching non-approximability result. As mentioned above, the gained insights for the harmonic case helped us to design a sophisticated IP-formulation for the problem which was very successful in practice [28]. See [27] for our publication on the theoretical aspects of the problem presented in this chapter.

Chapter 7: One of the most prominent open problems in scheduling is to determine the best possible approximation factor for scheduling jobs on unrelated machines to minimize the makespan. In contrast, for parallel and related machines approximation schemes and matching NP-hardness results have been known for a long time [52, 53]. The best possible approximation factor for unrelated machines is still not clear. Since they capture a very general setting, settling this question is a very important problem. In a seminal work, Lenstra et al. [69] developed an LP-based 2-approximation algorithm for the problem. On the other hand, they proved that the problem is NP-hard to approximate with a factor of $3/2 - \varepsilon$ for any $\varepsilon > 0$. This gap between 2 and 3/2 has not been diminished in more than 20 years[1] even though the problem is considered to be very important, see e. g. [95]. One natural way to approach the problem is to strengthen the linear program by adding suitable inequalities. A linear program which already implicitly contains a huge class of such inequalities is the configuration-LP. Recently, Svennson [104] showed that the configuration-LP has an integrality gap of $33/17 < 2$ for the restricted assignment case of the problem. However, we show that for the general case this is not true. We prove that in general the configuration-LP has an integrality gap of 2. Even more, we show that this is still true if we require that each job can be assigned to at most two machines (unrelated graph balancing). This is an indicator that the core difficulty of the problem lies in the unrelated graph bal-

[1] The only improvement is a slightly better rounding procedure for the LP by Lenstra et al. that yields a $2 - 1/m$ approximation algorithm [96].

ancing case rather than in the restricted assignment case. Then we study the closely related MaxMin-allocation problem where one wants to maximize the minimum machine load (rather than minimizing the maximum machine load as before). This objective can be understood as establishing a fairness condition for some given agents/machines. For that problem, it is known that in the unrelated graph balancing case the configuration-LP has an integrality gap of 2. However, solving the configuration-LP is very difficult and results only in a computationally very costly $(2+\varepsilon)$-approximation algorithm. In contrast, we give a purely combinatorial 2-approximation algorithm with a running time of $O(n^2)$. Our approximation factor is best possible, unless $P = NP$.

Most results of this thesis have already been published in conference proceedings [27, 84, 86, 88, 102, 106]. Moreover, as part of our research on the packet routing problem we published three further conference papers [60, 85, 87]. However, those results are beyond the scope of this thesis. Likewise, the computational aspects of the periodic maintenance problem were published in [28] but are not covered here.

Throughout this thesis, we assume knowledge of basic concepts of combinatorial optimization such as graphs, algorithms, NP-completeness, etc. For an introduction see e. g. [61]. Most algorithms presented in this thesis are approximation algorithm. For introductions to approximation algorithms see [51, 105, 109].