Teaching Complex Scheduling Algorithms

Sascha Hunold
TU Wien, Faculty of Informatics
Vienna, Austria
hunold@par.tuwien.ac.at

Bartłomiej Przybylski
Adam Mickiewicz University
Faculty of Mathematics and Computer Science
Poznań, Poland
bap@amu.edu.pl

***Abstract*—We introduce `Scheduling.jl` and show how it can be used for teaching the basics of scheduling theory to Computer Science students. In particular, our course focuses on scheduling algorithms for parallel, identical machines. For these problems, approximation algorithms and approximation schemes exist. However, we believe that students better understand advantages as well as disadvantages of these approximation algorithms when they investigate their implementations and examine how the algorithms work in practice. For that purpose, we have implemented a set of heuristics and approximation algorithms on top of `Scheduling.jl`. In the present article, we go through some of the implemented algorithms and explain why we believe these algorithms are particularly helpful for students to understand the basic concepts of approximation algorithms. In our experience, students remember algorithmic details much better if we show them examples using `Scheduling.jl`.**

***Index Terms*—Scheduling, Julia, Approximation Algorithms, Education, Gantt Charts, PTAS, FPTAS, Dynamic Programming**

I. Introduction

Scheduling is one of the fundamental research subjects, which is central to virtually all scientific domains that require any kind of resource sharing. Therefore, a large body of literature exists that introduces basic scheduling algorithms for various scheduling problems [1], [2], [3], [4], [5]. Most theoretical works in scheduling research use the three-field notation $\alpha|\beta|\gamma$ of Graham et al. [6] for classifying scheduling problems. By using this notation, each scheduling problem can be described by the machine environment α, the job characteristics β, and the optimality criterion γ. For example, $\mathrm{P}\,||\,C_{\max}$ defines the problem of scheduling jobs on identical parallel machines, where the maximum completion time ($C_{\max}$) should be minimized, while no further job characteristics are given. One example of such job characteristics could be the *moldable* job model (denoted as any), i.e., in the problem $\mathrm{P}\,|\,\mathrm{any}\,|\,C_{\max}$ [3], [7]. In the moldable model, each job can not only be executed on a single machine, but it may be allotted to several (between 1 and m) machines. The number of machines is selected by the scheduler, and this number of machines stays constant until a job has been completed. Another example of job characteristics are job processing times that are variable and depend on environmental factors such as the position r of a job in a schedule. For example, in the $\mathrm{P}\,|\,\text{in-tree}, p_{j,r} = \varphi(r)\,|\,C_{\max}$ problem, the processing times of jobs with in-tree precedence constraints are described by the φ function [8], [9].

In the present work, we focus on teaching scheduling algorithms to Master's students, which are enrolled in Computer Science (CS) degree programs. The problem of scheduling is fundamental in Computer Science, since we often need to make automatic decisions of which entity or process can use a certain resource. Thus, in a typical Bachelor program, CS students are taught basic scheduling algorithms, e.g., process scheduling algorithms in operating systems or loop scheduling algorithms in OpenMP. Most often, these classes (and respective literature) emphasize less on theoretical guarantees, as such analyses require the necessary tools from theoretical Computer Science. In a more advanced class on algorithms, and in particular on scheduling algorithms, i.e., in Master's courses, teachers can go into far more depth than in undergraduate courses. Scheduling algorithms in these courses are often motivated by the need to obtain a certain guarantee, which (many times) makes them non-intuitive to students. In order to help us teach these algorithms, we implemented several approximation algorithms on top of `Scheduling.jl`. In this context, we note that it was not our main goal to implement yet another novel algorithm for a specific research problem. The unique feature of our package is to explain algorithmic concepts to students. In the remainder, we describe how `Scheduling.jl` can help university teachers to introduce scheduling algorithms in classes. Although we explain `Scheduling.jl` in the context of teaching approximation algorithms in Master's courses, our software can also be used when teaching scheduling algorithms to undergraduate students, e.g., to introduce Graham's LIST algorithm [10].

We make the following contributions:

- We introduce the package `Scheduling.jl` that contains several approximation algorithms for scheduling problems, e.g., PTASes (polynomial-time approximation schemes) and FPTASes (fully polynomial-time approximation schemes).
- These implementations collect interesting properties of various scheduling algorithms, e.g., we can trace how instances are simplified or how solution sets are trimmed.
- Our scheduling algorithms can be visually analyzed, as `Scheduling.jl` can output the schedules as Gantt charts, either as static images (PNG, PDF) or as dynamic animations (GIF). These animations help students to comprehend the fundamental ideas behind these algorithms.

In the remainder of the article, we first set the context of this work by introducing our course on scheduling algorithms in Section II. Afterwards, we shortly introduce the framework

`Scheduling.jl` in Section III. In Section IV, we show how our package can help teaching approximation algorithms for scheduling problems. We discuss related work in Section V and conclude in Section VI.

II. A Class on Scheduling for Parallel Processing

We discuss a course on parallel algorithms with a specific focus on scheduling algorithms in parallel computing, which we regularly hold at the Vienna University of Technology. The overall goal of this class is to introduce Master's students in Computer Science degree programs to approximation algorithms in the context of parallel processing.

The class covers several topics. First, we discuss the concepts of NP-hardness and problem reductions (cf. Karp reduction). Second, we introduce basic scheduling notation (cf. Graham's notation) and fundamental algorithms, such as Graham's algorithms (e.g., LIST) for scheduling jobs on identical parallel machines [10]. Note that we use the words *job* and *task* interchangeably, which is also true for *machine* and *processor*. For example, Graham [10] talked about processing units and tasks, whereas most literature on scheduling theory (cf. [5]) refers to machines and jobs. Third, we explain the concepts of approximation algorithms with a special focus on scheduling. For introducing approximation algorithms and for related topics such as linear programming, we use the book by Williamson and Shmoys [11]. When the students have all the required tools, we talk about approximation algorithms and approximation schemes for scheduling parallel machines. These lectures are largely based on the Handbook of Scheduling [4] and the book on "Scheduling for Parallel Processing" by Drozdowski [3]. This is exactly the point where `Scheduling.jl` comes in, which will be discussed in Section IV. The course ends with lectures on further scheduling topics such as batch scheduling (e.g., SLURM [12]) or OpenMP scheduling strategies (cf. Ciorba et al. [13]).

In the context of the present work, we highlight how `Scheduling.jl` can be used to support the lectures when teaching approximation algorithms for scheduling parallel machines. Before we go into detail about individual algorithms, we summarize basic features of `Scheduling.jl`.

III. Scheduling.jl

`Scheduling.jl` [14] is a Julia package that provides building blocks for implementing scheduling algorithms in their most generic form, which are `Job`, `Machine`, `JobAssignment`, and `Schedule`. A classical `Job` J_j is defined by its processing time p_j, but can also be characterized by a weight w_j, a release date r_j, a due date d_j, or a deadline $\bar{d}_j$. A `Machine` M_i is defined by its speed. Of course, the set of parameters used can be extended if required. The goal of a scheduling algorithm is to find an assignment of jobs to machines, such that a given criterion is optimized. An assignment of jobs to machines defines the starting and the completion time of a job J_j on a machine M_i. We note that `Scheduling.jl` is designed to operate on exact values (rational numbers) rather than inexact ones (floating point numbers).

Listing 1: Example of using the basic functionality of `Scheduling.jl`; applying the LPT algorithm and reporting the $C_{\max}$ metric.

```
using Scheduling
using Scheduling.Algorithms
using Scheduling.Objectives

# Generate a set of jobs with processing times
J = Jobs([2, 12, 11, 4, 10, 3, 5])
# Generate a set of 4 identical machines
M = Machines(4)
# Generate a schedule using
# LPT (Largest Processing Time) rule
LPT = Algorithms.lpt(J, M)
println("Cmax     = $(Int(cmax(LPT)))")
Scheduling.TeX(LPT, "lpt_example.tex")
```

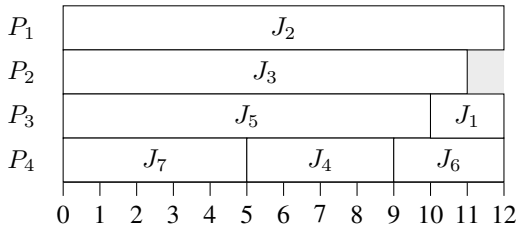


Fig. 1: LPT schedule produced by Listing 1.

Once a schedule has been created by an algorithm, the package `Scheduling.jl` provides different optimization criteria that can be computed for a schedule, e.g., the makespan $C_{\max}$, the average completion time $\sum_j C_j$, or the number of tardy jobs $\sum_j U_j$.

Listing 1 presents an example of how to use the basic functions of `Scheduling.jl`. We create a set of jobs J and a set of machines M and then apply the LPT[1] algorithm to obtain the corresponding LPT schedule. On this schedule, we compute the resulting makespan ($C_{\max}$). Finally, we export the schedule to LaTeX, which is shown in Figure 1.

IV. Understanding Approximation Algorithms for Parallel Machines

We now explain how `Scheduling.jl` and the provided algorithms can help students to understand concepts in scheduling theory. Notice that all algorithms discussed in this section are indeed part of `Scheduling.jl`.

A. Approximation Algorithms for $P2 \,||\, C_{max}$

The scheduling problem $P2 \,||\, C_{\max}$ is a very good choice when introducing the concepts of NP-hardness and approximation algorithms. In this problem, we have two identical machines (processors) and a set of jobs (tasks). The goal is to assign the tasks to the processors in such a way

[1] Largest Processing Time first

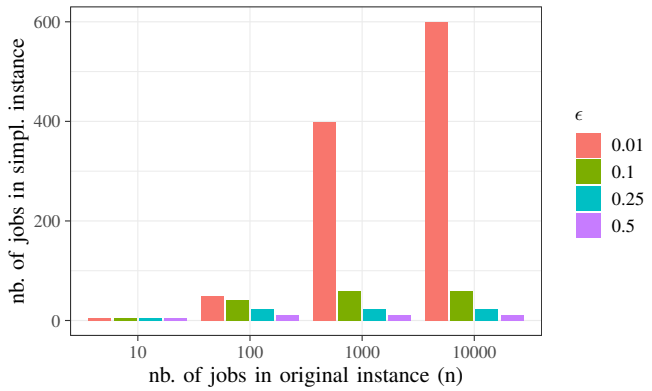


Fig. 2: Trade-off: Precision parameter (ε) vs. the number of jobs in the simplified instance for different sizes of the original instance.

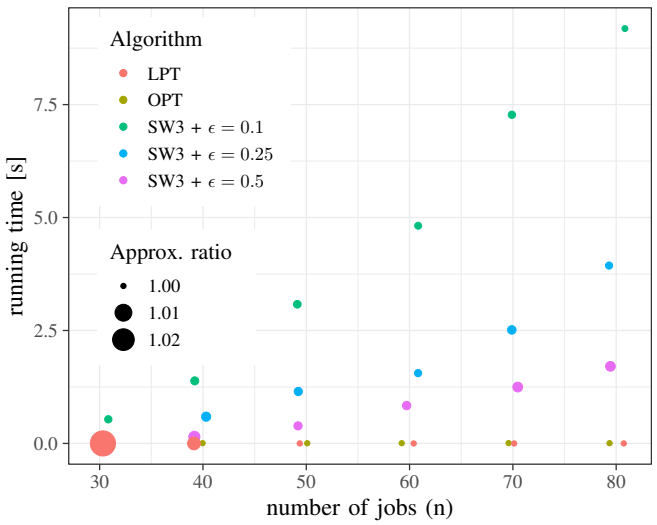


Fig. 3: Running time of SW3 depending on the number of jobs for different algorithms and precision bounds.

that the makespan is minimized. This problem is used as an example scheduling problem throughout the literature. Casanova et al. [15] show how to derive worst-case bounds for LIST and LPT on two processors. Jaehn and Pesch [16] also introduce the problem of scheduling independent jobs by using P2 $||$ $C_{\max}$.

In our class, it is important to understand how approximation schemes work. Therefore, we now present how `Scheduling.jl` supports this objective.

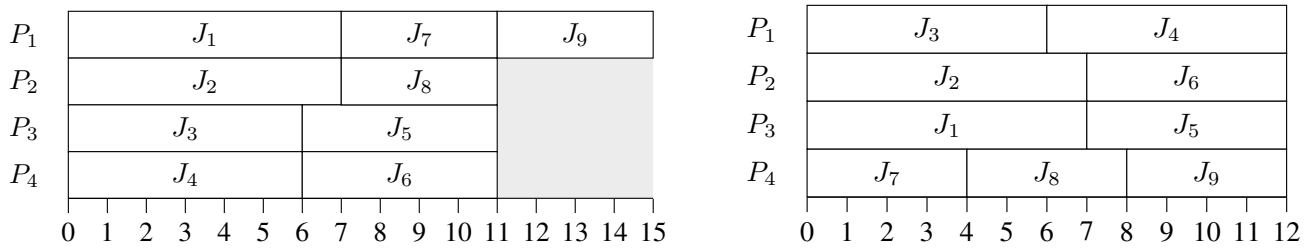
Schuurman and Woeginger [17] summarized different ways of how to obtain approximation schemes, i.e., FPTASes and PTASes. The main idea is to add structure when solving problems, e.g., by structuring the inputs, by structuring the outputs, or by structuring the algorithm itself. For ease of comprehension, Schuurman and Woeginger apply each of these three ways of structuring to the problem of P2 $||$ $C_{\max}$.

1) Goal: Introduce the Concept of Input Structuring: The input structuring for P2 $||$ $C_{\max}$ works as follows. We note that our goal is to obtain a PTAS, i.e., an approximation scheme with a $(1+\varepsilon)$OPT performance guarantee. Depending on the precision bound ε, the input-structuring procedure divides the jobs from the original instance into small and large jobs. All small jobs are virtually glued together and cut into equal-sized pieces. The goal is to reduce the number of jobs in the original instance. Thus, the simplified instance should contain a much smaller number of tasks, and in particular, the number of tasks must not depend on n but on ε, which is a constant.

This trade-off, precision vs. number of tasks in the simplified instance, is the main teaching objective in this context. Therefore, we show the students examples like the one presented in Fig. 2. In this figure, we compare the number of jobs in the original instance (x-axis) to the number of jobs in the simplified instance (y-axis), all depending on the selected precision parameter ε. We can clearly observe that the smaller the value of ε the larger the number of jobs in the simplified instance. We do stress in class that although the number of jobs for $\varepsilon = 0.01$ is much larger than for the value of $\varepsilon = 0.1$, it is still much smaller than in the original instance, i.e., 600 jobs in case of $n = 10\,000$ original jobs.

2) Goal: Introduce the Concept of Algorithmic Structuring: In our class on parallel algorithms, we also present how we can structure the algorithm. This technique is used by Schuurman and Woeginger [17], which we call SW3 in the remainder. The same technique is explained by Casanova et al. [15]. The essence of this method is to build schedules incrementally, but making sure that the number of partial schedules does not grow exponentially. In contrast to the method before, which led to the construction of a PTAS, the algorithmic structuring allows us to obtain an FPTAS. The students need to understand that the running time depends on the precision parameter and, of course, the algorithm. In Fig. 3, we compare the running time of the SW3 algorithm (an FPTAS) to the running time of the LPT heuristic and to the running time for obtaining the optimal solution. For solving the exact problem (OPT), we use the integer programming formulation given by Drozdowski [3] for P $||$ $C_{\max}$.

Not surprisingly, Fig. 3 clearly shows that the running time of all algorithms depends on the number of jobs in an instance. However, the students should recognize that the running time also depends on the precision parameter ε. If ε is small, e.g., $\varepsilon = 0.1$, the running time is significantly larger than for bigger values of ε. The circle size represents the obtained approximation ratio for a specific instance. The students should also notice that the smaller the epsilon, the better the approximation ratio is. In addition, we can observe that LPT produces the largest circles, i.e., it leads to the largest ratios. We need to add that the approximation ratios for these instances are very small; less than 1.02 (2%) in all cases. This analysis also demonstrates the unpredictability of the running time for integer programs. In Fig. 3, the running time of OPT is mostly similar to the running time of LPT and smaller than the FPTASes. However, in class, we show that this is only true



(a) LPT schedule.

(b) OPT schedule.

Fig. 4: Example of a worst case instance of LPT, which matches the bound of $\frac{4}{3} - \frac{1}{3m}$, i.e., $\frac{4}{3} - \frac{1}{12} = \frac{15}{12}$.

for some of the instances and some of the solvers. In the cases presented in the figure, Gurobi 9 solves the instances so quickly that one may get the impression that integer programming is always efficient. Thus, we demonstrate how much longer it takes other solvers, like GLPK, to solve these instances. This shows the students that they must be careful when relying on integer programming.

B. Approximation Algorithms for $P \,||\, C_{max}$

When we talk about basic scheduling algorithms in class, we surely need to introduce the problem of $\mathrm{P}\,||\,C_{\max}$. This is such a fundamental problem that it has to be included in the curriculum on scheduling for parallel processing. Graham [10] has proved that the LIST algorithm (schedule next task on least loaded machine) and the LPT algorithm (schedule the largest task on the least loaded machine) have approximation ratios of about 2 and $4/3$. These algorithms (LIST and LPT) can be easily understood by the students. In general, $\mathrm{P}\,||\,C_{\max}$ is a central problem in most textbooks on scheduling. For example, Benoit et al. [18] show that the problem is NP-hard by presenting a reduction from 3-PARTITION. Schwiegelshohn introduces concepts of scheduling jobs on identical parallel machines in a dedicated chapter on Job Scheduling [19].

Although the problem has been discussed in different ways in various textbooks, $\mathrm{P}\,||\,C_{\max}$ still has a few properties where `Scheduling.jl` can help the students to comprehend the main idea.

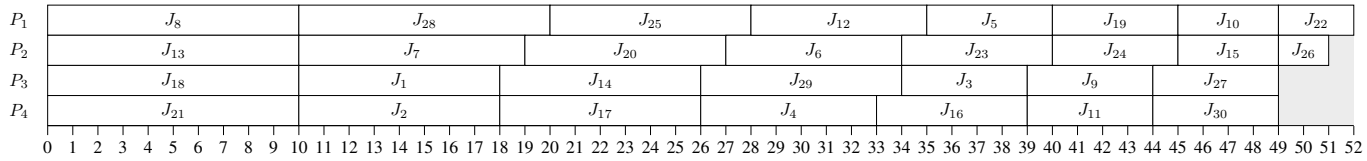
1) Goal: Visualizing the Worst Case of LPT: We first look at the worst case of LPT. Often students ask whether the provided bound of $\frac{4}{3}-\frac{1}{3m}$ is tight (cf. [19], [5]). Schwiegelshohn presents one example that matches this bound [19], which is depicted in Fig. 4. With `Scheduling.jl`, the students do not only see this worst case instance in action, they can also experiment with other instances and can try to find other badly behaving instances. This example gives them a very good understanding of what these bounds mean.

2) Goal: Comprehending the Concept of Dual Approximation: `Scheduling.jl` contains an implementation of the PTAS for the $\mathrm{P}\,||\,C_{\max}$ problem, which was proposed by Hochbaum and Shmoys (HS) [20]. They introduced the concept of dual approximation, where they find an approximate solution to $\mathrm{P}\,||\,C_{\max}$ by solving a dual problem approximately, which is the bin-packing problem in this case. We found that the basic idea of dual approximation seems clear to the students, but running code examples in `Scheduling.jl` has answered many open questions. We let students compare the algorithms LPT, HS (with different precision bounds), and the integer programming formulation OPT in terms of running times and computed schedules. Fig. 5 presents one such example, where we used `Scheduling.jl`'s feature to produce LaTeX Gantt charts to analyze the HS algorithm. The students usually observe that the HS algorithm does not seem to produce a certain pattern, as seen for LPT. More importantly, the students quickly understand that PTASes are often only of theoretical value, as they may require a long time for producing a solution. Together with the students, we examine the running time of the HS algorithm when adjusting either the precision bound (ε) or the number of tasks (n).

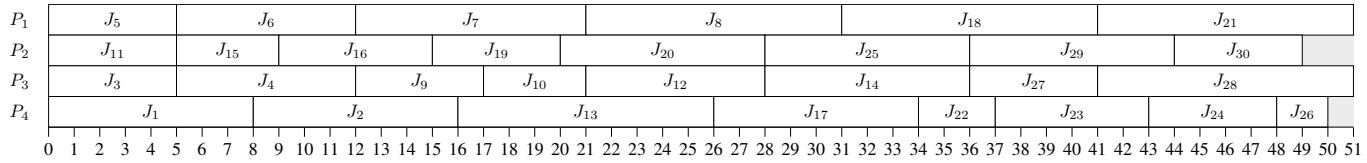
3) Goal: Understanding Bounds in the Online Case: Until now, we have only introduced the offline case to students, i.e., the case in which we have knowledge about all jobs and their processing times. In the online case, we are presented with a sequence of jobs, and we need to schedule them immediately, i.e., assign them to the machines. Albers [21] has shown a novel strategy how to solve this problem, and her concept was later improved by Fleischer and Wahl (MR algorithm). The algorithm of Albers is 1.923-competitive and the MR algorithm is 1.9201-competitive. When we introduce these algorithms in class, the students understand the general idea, which is to keep a few machines with high and a few with low load at all times. However, presenting animations of how such an algorithm works in practice helped students immensely. Figure 6 shows different stages of the MR algorithm when the algorithm is presented a list of 20 jobs. The figure presents the actual screenshots from the animated images. With such animations, it becomes much clearer which machines are lightly loaded and which machines heavily. All this is built-in into `Scheduling.jl`.

C. Approximation Algorithms for $P|any|\,C_{max}$

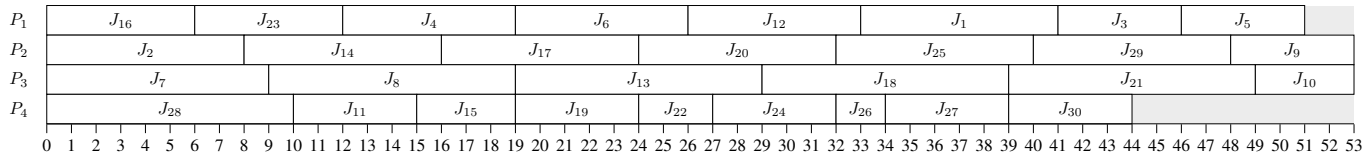
A class on scheduling in parallel computing surely needs to cover the topic of scheduling parallel tasks. There are several task models when scheduling parallel tasks, but the moldable model is often applied, as it has many applications in practice (e.g., scheduling MPI applications). In the moldable model, a task can be executed by $1, 2, \ldots,$ or m machines (processors) simultaneously. Depending on the number of processors allotted to a task, the running time of this very task changes. Typically,



(a) LPT

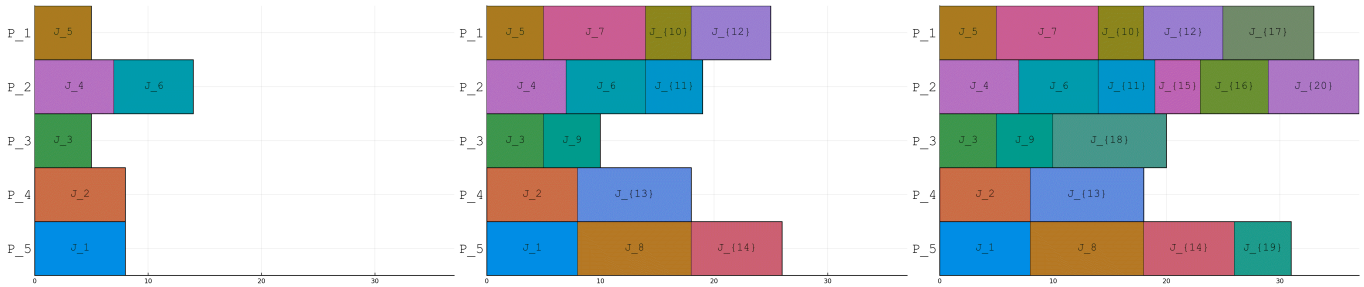


(b) OPT



(c) HS

Fig. 5: Schedules produced by LPT and HS compared to the optimal solution (OPT); $n = 30$, $m = 4$, $\varepsilon = \frac{1}{10}$ (HS).

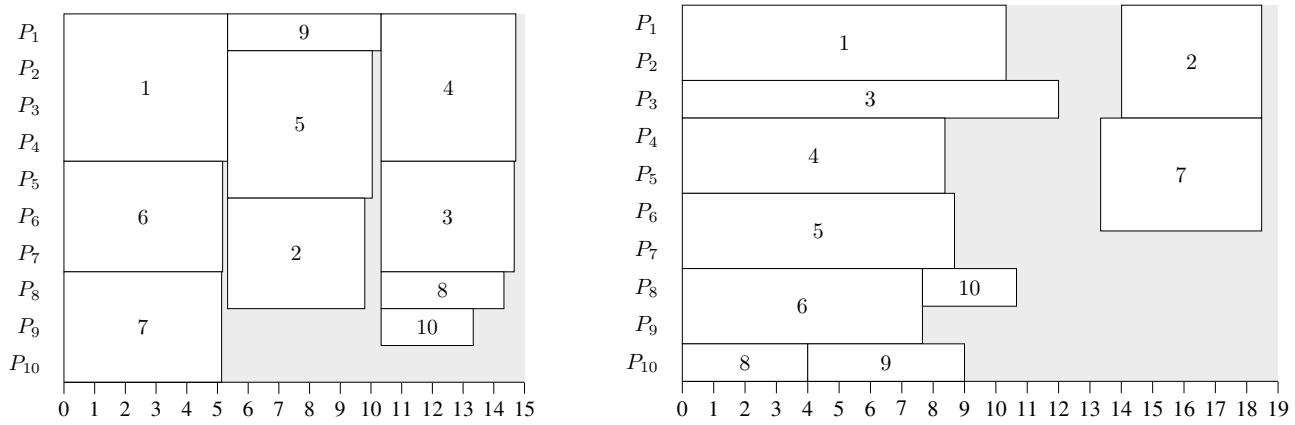


(a) After 6 Jobs. (b) After 14 Jobs. (c) After 20 Jobs.

Fig. 6: Step-wise schedule building for the MR algorithm using `Scheduling.jl`; $n = 20$, $m = 5$, uniform processing times.

we use running time functions that decrease when the number of allotted processors increases. In other words, if we use more resources we expect a shorter running time. An OpenMP program is one example of such a moldable task. Before we start an OpenMP program, we define the number of threads that this program should start. The number of threads then stays constant during the execution of the program. (We oversimplify the processing using threads in OpenMP.) The moldable model for parallel tasks is often denoted as *any* (cf. Drozdowski [3]), and thus, in our class, we investigate the scheduling problem of $\mathrm{P} \,|\, \mathrm{any} \,|\, C_{\max}$.

Solving this problem efficiently is not easy. Turek et al. [22] presented a novel strategy (we call it TWY) to schedule such moldable tasks with a performance guarantee. The interesting point for the students to comprehend is that this algorithm 1) does not have any limitation on the running time function of the moldable tasks and 2) uses strip packing algorithms to obtain its bound. The overall structure of the TWY algorithm can be observed in Fig. 7a. First, the tasks 1, 6, and 7 are allotted to a set of processors and scheduled LPT-style in the first shelf. The next shelf starts where the highest task of the previous shelf has ended, which is the completion time of task 1. Then again, the biggest remaining tasks are assigned to the available processors until this shelf is also filled, after which the last shelf is packed with tasks. The TWY algorithm obtains a bound of 2.5 with respect to the optimal solution. The best-known bound for scheduling independent moldable tasks was proposed by Mounié et al. [23] and provides a performance guarantee of $\frac{3}{2} + \varepsilon$. This algorithm, denoted as MRT, leverages the concept of dual approximation, but instead of bin packing, as done by Hochbaum and Shmoys, a knapsack problem is solved as the dual problem. In contrast to Turek et al. [22], the algorithm of Mounié et al. [23] assumes that moldable tasks are monotonic, i.e., the running time is non-increasing in the number of processors. If this is the case, the problem can be reformulated as a knapsack problem. Here, the students should see how we can apply all previously introduced concepts such as approximation, dual-approximation, shelf scheduling, and solving the knapsack problem efficiently (whose running time, depending on the definition, is mostly pseudo-polynomial).



(a) TWY. (b) MRT.

Fig. 7: Schedules produced by the TWY and the MRT algorithm for one instance of $\mathrm{P} \,|\, \mathrm{any} \,|\, C_{\max}$, $n = 10$, $m = 10$.

We compare the schedules produced by the TWY and the MRT algorithm in Fig. 7. As before, the students should inspect the overall structure of the MRT algorithm. The MRT schedule consists of two shelves, one of size d and one of size $d/2$, where d is a guess of the optimal makespan. In this example, we used the running time model for the parallel tasks by Prasanna and Musicus [24], which guarantees us a monotonic task behavior. In class, we usually experiment with both algorithms and different inputs. It is often interesting to observe that moldability is only important (can only be effectively used) if the number of tasks is relatively small compared to the number of processors. In these cases, tasks are indeed assigned to more than one processor. However, if the number of tasks exceeds a certain limit (e.g., about two to three times the number of processors), then tasks are exclusively scheduled as single-processor tasks.

V. Related Work

In 1995, Baeza-Yates [25] shared his view on how algorithms should be taught to students. He names approximation algorithms as one of the important advanced topics. This is what we do in our work, with a focus on scheduling problems.

Fundamental scheduling algorithms are part of many Computer Science curricula and are taught in most universities. We can imagine that many classes on these subjects are supported by example codes and applications presenting Gantt charts. What we do know is that lecturers at universities use the book by Pinedo [5] as a textbook in the reading list. Pinedo uses a system called LEKIN[2] to visualize the results of different scheduling algorithms. A similar project is called LISA, which is, like `Scheduling.jl`, a framework for implementing scheduling algorithms, mostly for shop problems. LISA was used in several papers, e.g., by Bräsel et al. [26]. Jedule [27] is another Gantt chart viewer for scheduling problems, which lets the user to interactively explore a schedule and which can also be used for teaching the basics of scheduling theory.

[2]http://web-static.stern.nyu.edu/om/software/lekin/

VI. Conclusions

Teaching scheduling algorithms is an important topic in Computer Science programs. We have introduced `Scheduling.jl`, which is a collaborative platform for education and research. The choice of Julia is not a coincidence, as speed and simplicity are our main objectives, which are greatly supported by the Julia programming language. We have implemented several approximation algorithms for various scheduling problems, such as $\mathrm{P2} \,||\, C_{\max}$ and $\mathrm{P} \,||\, C_{\max}$. Our main objective is teaching the essence of approximation algorithms in scheduling. With `Scheduling.jl`, students can easily study how the running time depends on various algorithmic and instance parameters, such as the precision bound for approximation schemes or the input size. Currently, `Scheduling.jl` gets extended to provide reference implementations for research, such as algorithms for $\mathrm{Q} \,||\, C_{\max}$ [28] and $(\mathrm{P}m,\mathrm{P}k) \,|\, \mathrm{any} \,|\, C_{\max}$ [29].

We would like to emphasize that `Scheduling.jl` can also be used in other teaching scenarios, e.g., in undergraduate classes on operating systems. The simplicity of `Scheduling.jl` makes it easily extensible. Therefore, students can implement their own scheduling algorithms on top of `Scheduling.jl` in less than five minutes. For example, it would be very easy to use `Scheduling.jl` in a class on operating systems to introduce basic process scheduling algorithms, such as Shortest Job First (SJF) or Shortest Remaining Processing Time first (SRPT).

References

[1] J. Blazewicz, K. Ecker, E. Pesch, G. Schmidt, M. Sterna, and J. Weglarz, *Handbook on Scheduling: From Theory to Practice*. Springer, 2019.

[2] P. Brucker, *Scheduling algorithms*, 4th ed. Springer, 2004.

[3] M. Drozdowski, *Scheduling for Parallel Processing*, ser. Computer Communications and Networks. Springer, 2009.

[4] J. Y. Leung, Ed., *Handbook of Scheduling - Algorithms, Models, and Performance Analysis*. Chapman and Hall/CRC, 2004.

[5] M. L. Pinedo, *Scheduling: Theory, Algorithms, and Systems*, 5th ed. Springer, 2016.

[6] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. R. Kan, "Optimization and approximation in deterministic sequencing and scheduling: a survey," in *Annals of discrete mathematics*. Elsevier, 1979, vol. 5, pp. 287–326.

[7] S. Hunold, "One step toward bridging the gap between theory and practice in moldable task scheduling with precedence constraints," *Concurr. Comput. Pract. Exp.*, vol. 27, no. 4, pp. 1010–1026, 2015.

[8] B. Przybylski, "Precedence constrained parallel-machine scheduling of position-dependent jobs," *Optimization Letters*, vol. 11, no. 7, pp. 1273–1281, 2017.

[9] ——, "A new model of parallel-machine scheduling with integral-based learning effect," *Computers & Industrial Engineering*, vol. 121, pp. 189–194, 2018.

[10] R. L. Graham, "Bounds on multiprocessing timing anomalies," *SIAM Journal of Applied Mathematics*, vol. 17, no. 2, pp. 416–429, 1969.

[11] D. P. Williamson and D. B. Shmoys, *The Design of Approximation Algorithms*. Cambridge University Press, 2011.

[12] A. B. Yoo, M. A. Jette, and M. Grondona, "SLURM: simple linux utility for resource management," in *Proceedings of 9th International Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)*, ser. Lecture Notes in Computer Science, D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn, Eds., vol. 2862. Springer, 2003, pp. 44–60. [Online]. Available: https://doi.org/10.1007/10968987_3

[13] F. M. Ciorba, C. Iwainsky, and P. Buder, "OpenMP loop scheduling revisited: Making a case for more schedules," in *Proceedings of 14th International Workshop on OpenMP (IWOMP)*, ser. Lecture Notes in Computer Science, B. R. de Supinski, P. Valero-Lara, X. Martorell, S. M. Bellido, and J. Labarta, Eds., vol. 11128. Springer, 2018, pp. 21–36. [Online]. Available: https://doi.org/10.1007/978-3-319-98521-3_2

[14] S. Hunold and B. Przybylski, "Scheduling.jl - collaborative and reproducible scheduling research with Julia," *CoRR*, vol. abs/2003.05217, 2020. [Online]. Available: https://arxiv.org/abs/2003.05217

[15] H. Casanova, A. Legrand, and Y. Robert, *Parallel Algorithms*. CRC Press, 2008.

[16] F. Jaehn and E. Pesch, *Ablaufplanung: Einführung in Scheduling*. Springer Berlin Heidelberg, 2019.

[17] P. Schuurman and G. Woeginger, *Lectures on Scheduling*, 2000, ch. Approximation Schemes - A Tutorial, (was supposed to appear).

[18] A. Benoit, Y. Robert, and F. Vivien, *A Guide to Algorithm Design - Paradigms, Methods, and Complexity Analysis*, ser. Chapman and Hall / CRC Applied Algorithms and Data Structures Series. CRC Press, 2013.

[19] Y. Robert and F. Vivien, Eds., *Introduction to Scheduling*, ser. CRC computational science series. CRC Press / Chapman and Hall / Taylor & Francis, 2009.

[20] D. S. Hochbaum and D. B. Shmoys, "Using dual approximation algorithms for scheduling problems theoretical and practical results," *J. ACM*, vol. 34, no. 1, pp. 144–162, 1987.

[21] S. Albers, "Better bounds for online scheduling," in *STOC*, F. T. Leighton and P. W. Shor, Eds. ACM, 1997, pp. 130–139.

[22] J. Turek, J. L. Wolf, and P. S. Yu, "Approximate algorithms scheduling parallelizable tasks," in *Proceedings of the 4th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA '92)*, L. Snyder, Ed. ACM, 1992, pp. 323–332.

[23] G. Mounie, C. Rapine, and D. Trystram, "A 3/2-approximation algorithm for scheduling independent monotonic malleable tasks," *SIAM J. Comput.*, vol. 37, no. 2, pp. 401–412, 2007.

[24] G. N. S. Prasanna and B. R. Musicus, "Generalized multiprocessor scheduling and applications to matrix computations," *IEEE Trans. Parallel Distributed Syst.*, vol. 7, no. 6, pp. 650–664, 1996.

[25] R. A. Baeza-Yates, "Teaching algorithms," *SIGACT News*, vol. 26, no. 4, pp. 51–59, 1995.

[26] H. Bräsel, A. Herms, M. Mörig, T. Tautenhahn, J. Tusch, and F. Werner, "Heuristic constructive algorithms for open shop scheduling to minimize mean flow time," *Eur. J. Oper. Res.*, vol. 189, no. 3, pp. 856–870, 2008.

[27] S. Hunold, R. Hoffmann, and F. Suter, "Jedule: A tool for visualizing schedules of parallel applications," in *Proceedings of the 39th International Conference on Parallel Processing (ICPP Workshops)*. IEEE Computer Society, 2010, pp. 169–178.

[28] K. Jansen, "An EPTAS for scheduling jobs on uniform processors: Using an MILP relaxation with a constant number of integral variables," in *Proceedings of 36th International Colloquium on Automata, Languages and Programming, (ICALP), Part I*, ser. Lecture Notes in Computer Science, S. Albers, A. Marchetti-Spaccamela, Y. Matias, S. E. Nikoletseas, and W. Thomas, Eds., vol. 5555. Springer, 2009, pp. 562–573. [Online]. Available: https://doi.org/10.1007/978-3-642-02927-1_47

[29] R. Bleuse, S. Hunold, S. Kedad-Sidhoum, F. Monna, G. Mounie, and D. Trystram, "Scheduling independent moldable tasks on multi-cores with gpus," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 9, pp. 2689–2702, 2017.