Load Balancing: The Long Road from Theory to Practice*

Sebastian Berndt[1], Max A. Deppert[2], Klaus Jansen[3], Lars Rohwedder[4]

[1] University of LÃ¼beck, Germany
s.berndt@uni-luebeck.de
[2] TU Hamburg-Harburg, Germany
max.deppert@tuhh.de
[3] University of Kiel, Germany
kj@informatik.uni-kiel.de
[4] Maastricht University, Netherlands
l.rohwedder@maastrichtuniversity.nl

Keywords : *Combinatorial Optimization, Integer Programming, Approximation Algorithms.*

1 Introduction

Makespan minimization on identical parallel machines (often denoted by $P||C_{\max}$) asks for a distribution of a set J of $n = |J|$ jobs to $m \leq n$ machines. Each job $j \in J$ has a processing time p_j and the objective is to minimize the makespan, i.e., the maximum sum of processing times of jobs assigned to a single machine. More formally, a *schedule* $\sigma\colon J \to \{1, \dots, m\}$ assigns jobs to machines. The load $\ell_{\sigma,i}$ of machine i in schedule σ is defined as $\sum_{j \in \sigma^{-1}(i)} p_j$ and the *makespan* $\mu(\sigma) = \max_i\{\ell_{\sigma,i}\}$ is the maximal load. The goal is to find a schedule σ minimizing $\mu(\sigma)$. This is a widely studied problem both in operations research and in combinatorial optimization and has led to many new algorithmic techniques. For example, it has led to one of the earliest examples of an approximation scheme and the use of the dual approximation technique [5].

2 Known Results

The problem is known to be strongly NP-hard and thus we cannot expect to find an exact solution in polynomial time. Many approximation algorithms that run in polynomial time and give a non-optimal solution have been proposed for this problem. From a theory point of view, the strongest approximation result is a polynomial time approximation scheme (PTAS) which gives a $(1 + \varepsilon)$-approximation, where the

*Published on ALENEX 2022 [3] and GitHub [2]. Supported by German Research Foundation (DFG) project JA 612/20-1

precision $\varepsilon > 0$ can be chosen arbitrarily small and is given to the algorithm as input. This goes back to a seminal work by Hochbaum and Shmoys [5]. The running time of such schemes for $P||C_{\max}$ were drastically improved over time [1, 6, 9] and the best known running time is $2^{O(1/\varepsilon \log^2(1/\varepsilon))} \log(n) + O(n)$ due to Jansen and Rohwedder [8] (extension of [7] by the theory of discrepancy), which is subsequently called the JR-algorithm. The JR-algorithm is in fact an algorithm for integer programming, but gives this running time when applied to a natural formulation of $P||C_{\max}$. A PTAS with a running time of $f(1/\varepsilon) \cdot n^{O(1)}$ like in the JR-algorithm is called an efficient polynomial time approximation scheme (EPTAS). It follows from the strong NP-hardness that no fully polynomial time approximation scheme (FPTAS), an approximation scheme polynomial in both n and $1/\epsilon$, exists unless P = NP. Furthermore, for any $\delta > 0$, there is no PTAS for $P||C_{\max}$ in time $2^{O((1/\varepsilon)^{1-\delta})} + n^{O(1)}$, unless the exponential time hypothesis (ETH) fails [4].

PTAS's are often believed to be impractical. They tend to yield extremely high (though polynomial) running time bounds even for moderate precisions ϵ, see Marx [10]. By some, the research on PTAS's has even been considered damaging for the large gap between theory and practice that it creates [12]. Although EPTAS's (when FPTAS's are not available) are sometimes proposed as a potential solution for this situation [10], we are not aware of a practical implementation of an EPTAS. For example, an approximation scheme for euclidean TSP was implemented by Rodeker et al., but the algorithm was merely inspired by an EPTAS and it does not retain the theoretical guarantee [11]. Although this is an interesting research direction as well, it remains an intriguing question whether one can obtain a practically relevant EPTAS implementation with actual theoretical guarantees. On the one hand, we believe that this is an important question to ask concerning the relevance of such a major field of research. On the other hand, such a PTAS implementation has great advantages in itself, since it exhibits a clean and generic design that is not specific to any concrete precision, as well as a (theoretically) unlimited potential of the precision.

3 Our Results

As a major milestone we obtain a generic PTAS implementation that achieves in reasonable time a precision which beats the best known guarantee of a polynomial time non-PTAS algorithm. This precision to the best of our knowledge is $2/11 \approx 18.2\%$, which is guaranteed by the MULTIFIT algorithm. The claim might appear vague, since the running time depends not only on ϵ, but also on the instance. We believe that it is plausible nevertheless: The algorithm we use, which is based on the JR-algorithm, reduces the problem to performing $O(\log(n))$ many fast Fourier transformations (FFTs), where the size of the FFT input depends only on ϵ and not the instance itself. Hence, the running time for all instances (using the same precision) is very stable and predictable. This is in the spirit of an EPTAS running time. We successfully run experiments of our implementation for a precision of $\epsilon < 2/11$ and thus make the claim that this precision is practically feasible in general. This is also the main message of our paper. For completeness, we provide comparisons

of the solution quality obtained empirically. While the theoretical guarantee of the PTAS is better, the difference to non-PTAS algorithms is marginal at this state and it is not yet evident in the experiments. The execution of the PTAS is computationally expensive and the considered precision is on the edge of what is realistic for our implementation. However, we believe that further optimization or more computational resources can lead to also empirically superior results. Nevertheless, the successful execution with a low precision value forms a proof of concept for practical PTAS's.

Towards obtaining such an implementation we need to fine-tune the JR-algorithm significantly. In particular, it requires non-trivial theoretical work and novel algorithmic ideas. In fact, our variant has a slightly better dependence on the precision, namely $2^{O(1/\epsilon \log(1/\epsilon) \log\log(1/\epsilon))}$, giving the best known running time for this problem. Our approach also greatly reduces the constants hidden by the O-notation. We first construct an integer program (IP) — the well-known *configuration IP* — that implies a $(1+\epsilon)$-approximation by rounding the processing times. This IP has properties that allow sophisticated algorithms to solve it efficiently. We present several reduction steps to simplify and compress the IP massively. As extensions of this configuration IP are widely used, we believe this to be of interest in itself. For the makespan minimization problem, we obtain an IP where the columns of the constraint matrix have ℓ_∞-norms bounded by 2 and ℓ_1-norms bounded by $O(\log(1/\varepsilon))$. In contrast, in the classical configuration integer program used in many of the previous PTAS's both of these norms are bounded by $O(1/\varepsilon)$. This allows us to greatly reduce the size of the FFT instances in the JR-algorithm without losing the theoretical guarantee. For example, for $\epsilon \approx 17.29\%$, our reduced IP lowers the instance sizes for FFT from 49^{12} words for the configuration IP to 5^{12} words. Another important aspect in the algorithm is the rounding of the processing times. In general, one needs to consider only $O(1/\epsilon \log(1/\epsilon))$ different rounded processing times (to guarantee a precision of ϵ). This number has great impact on the size of the FFT instances. For concrete ϵ the general rounding scheme might not give the optimal number of rounded processing times. We present a mixed integer linear program that can be used to generically optimize the rounding scheme for guaranteeing a fixed precision ϵ (or equivalently, for a fixed number of rounded processing times).

4 References

[1] Noga Alon, Yossi Azar, Gerhard J. Woeginger, and Tal Yadid. Approximation schemes for scheduling on parallel machines. *Journal of Scheduling*, 1:55–66, 1998.

[2] Sebastian Berndt, Max A. Deppert, Klaus Jansen, and Lars Rohwedder. Implementation of the BDJR algorithm. `https://github.com/made4this/BDJR`, 2021.

[3] Sebastian Berndt, Max A. Deppert, Klaus Jansen, and Lars Rohwedder. Load balancing: The long road from theory to practice. In *Proc. ALENEX 2022*, 2022.

[4] Lin Chen, Klaus Jansen, and Guochuan Zhang. On the optimality of exact and approximation algorithms for scheduling problems. *J. Comput. Syst. Sci.*, 96:1–32, 2018.

[5] Dorit S. Hochbaum and David B. Shmoys. Using dual approximation algorithms for scheduling problems theoretical and practical results. *J. ACM*, 34(1):144–162, 1987.

[6] Klaus Jansen, Kim-Manuel Klein, and José Verschae. Closing the gap for makespan scheduling via sparsification techniques. *Math. Oper. Res.*, 45(4):1371–1392, 2020. doi:10.1287/moor.2019.1036.

[7] Klaus Jansen and Lars Rohwedder. On integer programming and convolution. In *Proc. ITCS 2019*, pages 43:1–43:17, 2019.

[8] Klaus Jansen and Lars Rohwedder. On integer programming, discrepancy, and convolution. *Math. Oper. Res.*, 2022. doi:10.1287/moor.2022.1308.

[9] Joseph Y.-T. Leung. Bin packing with restricted piece sizes. *Inf. Process. Lett.*, 31(3):145–149, 1989. doi:10.1016/0020-0190(89)90223-8.

[10] Dániel Marx. Parameterized complexity and approximation algorithms. *Comput. J.*, 51(1):60–78, 2008. doi:10.1093/comjnl/bxm048.

[11] Bárbara Rodeker, M. Virginia Cifuentes, and Liliana Favre. An empirical analysis of approximation algorithms for euclidean TSP. In *Proc. CSC 2009*, pages 190–196. CSREA Press, 2009.

[12] Frances Rosamond. Parameterized complexity-news. *The Newsletter of the Parameterized Complexity Community Volume*, 2006.