A monotone approximation algorithm for scheduling with precedence constraints

Sven O. Krumke* Anne Schwahn* Rob van Stee† Stephan Westphal*

Abstract

We provide a monotone $O(m^{2/3})$-approximation algorithm for scheduling related machines with precedence constraints.

Keywords: scheduling, algorithmic mechanism design, precedence constraints

1 Introduction

Internet users and service providers act selfishly and spontaneously, without an authority that monitors and regulates network operation in order to achieve some social optimum such as minimum total delay. An interesting and topical question is how much performance is lost because of this. This generates new algorithmic problems, in which we investigate the cost of the lack of coordination, as opposed to the lack of information (online algorithms) or the lack of unbounded computational resources (approximation algorithms).

There has been a large amount of previous research into approximation and online algorithms for a wide variety of computational problems, but most of this research has focused on developing good algorithms for problems under the implicit assumption that the algorithm can make definitive decisions which are always carried out. On the internet, this assumption is no longer valid, since there is no central controlling agency. To solve problems which occur, e.g., to utilize bandwidth efficiently (according to some measure), we now not only need to deal with an allocation problem which might be hard enough to solve in itself, but also with the fact that the entities that we are dealing with (e.g. agents that wish to move traffic from one point to the other) do not necessarily follow our orders but instead are much more likely to act selfishly in an attempt to optimize their private return (e.g. minimize their latency).

Mechanism design is a classical area of research with many results. Typically, the fundamental idea of mechanism design is to design a game in such a way that truth telling is a dominant strategy for the agents: it maximizes the profit for each agent individually. That is, each agent has some private data that we have no way of finding out, but by designing our game properly we can induce them to tell us what that is (out of well-understood self-interest), thus allowing us to optimize some objective while relying on the truthfulness of the data that we have. This is done by introducing *side payments* for the agents. In a way, we reward them (at some cost to us) for telling

*University of Kaiserslautern, Department of Mathematics, P.O.Box 3049, Paul-Ehrlich-Str. 14, 67653 Kaiserslautern, Germany. {krumke,schwahn,westphal}@mathematik.uni-kl.de

†Corresponding author. University of Karlsruhe, Department of Computer Science, 76128 Karlsruhe, Germany. vanstee@ira.uka.de.

us the truth. The role of the mechanism is to collect the claimed private data (bids), and based on these bids to provide a solution that optimizes the desired objective, and hand out payments to the agents. The agents know the mechanism and are computationally unbounded in maximizing their utility.

A seminal paper [4] considered the general problem of one-parameter agents. The class of one-parameter agents contain problems where any agent i has a private value t_i and his valuation function has the form $w_i \cdot t_i$, where w_i is the work assigned to agent i. Each agent makes a bid depending on its private value and the mechanism, and each agent wants to maximize its own profit. The paper [4] shows that in order to achieve a truthful mechanism for such problems, it is necessary and sufficient to design a *monotone* approximation algorithm, and use a payment function given by [4]. An algorithm is monotone if for every agent, the amount of work assigned to it does not increase if its bid increases. More formally, an algorithm is monotone if given two vectors of length m, b, b' which represent a set of m bids, which differ only in one component i, i.e., $b_i > b'_i$, and for $j \neq i$, $b_j = b'_j$, then the total size of the jobs (the work) that machine i gets from the algorithm if the bid vector is b is never higher than if the bid vector is b'.

Using this result, monotone (and therefore truthful) approximation algorithms were designed for several classical problems, like scheduling on related machines to minimize the makespan, where the bid of a machine is the inverse of its speed [4, 2, 6, 1, 14], shortest path [5, 10], set cover and facility location games [9], and combinatorial auctions [15, 16, 3].

Problem definition In this paper, we consider the problem of scheduling jobs in a multiprocessor setting where there are precedence constraints between tasks, and where the performance measure is the makespan, the time when the last task finishes. We denote the number of processors by m and the number of jobs by n. We consider the version of this problem where the machines are related: each machine has a speed at which it runs, which does not depend on the job being run.

Denote the size of job j by p_j ($j = 1, \ldots, n$). Denote the speed of machine i by s_i ($i = 1, \ldots, m$). In our model, each machine belongs to a selfish user. The private value (t_i) of user i is equal to $1/s_i$, that is, the cost of doing one unit of work. The load on machine i, L_i, is the total size of the jobs assigned to machine i divided by s_i. The profit of user i is $P_i - L_i$, where P_i is the payment to user i by the payment scheme defined by [4].

Our goal is to minimize the makespan. This problem is NP-complete in the strong sense [11] even on identical machines and without precedence constraints. As is generally the case in algorithmic mechanism design, we are not interested in maximizing the total profit of the users. In this case, our objective function is not even directly related to the loads (which determine the profits of the agents).

In order to analyze our approximation algorithms we use the approximation ratio. For an algorithm $\mathcal{A}$ and input σ, we denote the cost of $\mathcal{A}$ on input σ by $\mathcal{A}(\sigma)$. An optimal algorithm is denoted by OPT. The approximation ratio of $\mathcal{A}$ is the infimum $\mathcal{R}$ such that for any input σ, $\mathcal{A}(\sigma) \leq \mathcal{R} \cdot \text{OPT}(\sigma)$. If the approximation ratio of an offline algorithm is at most ρ we say that it is a ρ-approximation.

Previous results (non-selfish machines) The classic problem where all machines are identical was considered by Graham in his seminal paper [12], where he showed that list scheduling produces a $(2 - \frac{1}{m})$-approximate solution. Jaffe [13] presented an algorithm for the problem with related machines with approximation ratio of $O(\sqrt{m})$. This was later improved to $O(\log m)$, first using a

linear programming relaxation [8] and then with a combinatorial algorithm [7], using a new and more involved lower bound for the optimal makespan.

Our result We present a monotone approximation algorithm based on Jaffe [13] which achieves an approximation ratio of $O(m^{2/3})$. Throughout the paper, we assume that the machines are sorted in a fixed order of non-decreasing bids (i.e. non-increasing speeds, assuming the machine agents are truthful, $s_1 \geq s_2 \geq \ldots \geq s_m$).

2 Algorithm

Our algorithm works as follows. For simplicity of presentation, we assume that $m^{2/3}$ is an integer. We define a fixed ordering of the machines which does not depend on the speeds. We use this ordering in Step 4 below.

1. For a given bid vector $b = (b_1, \ldots, b_m)$, normalize such that the largest speed (smallest bid) is 1.
2. Ignore all machines with speed less than α, where $\alpha < 1$ is a parameter to be fixed later.
3. If at most $m^{2/3}$ machines remain, assign all jobs to the fastest machine. Extend the partial ordering given by the precedence constraints to a complete ordering and run the jobs in this order.
4. If $i > m^{2/3}$ machines remain, consider all the schedules produced by List Scheduling on $m^{2/3}+1, \ldots, i$ *identical* machines and use the lexicographically smallest schedule that minimizes the **maximum load** (i.e. **not** necessarily the makespan!). For this schedule, reorder the job loads such that the ith largest load ends up on the ith fastest machine according to the bids.
5. During the execution, each machine repeatedly selects a job that is assigned to it and for which all predecessors have finished running. This job is run until it completes. If there is no such job, the machine is idle.

3 Analysis

Theorem 1 *This algorithm is monotone.*

Proof If a machine that receives no load becomes slower (increases its bid), it will still receive zero load. If it becomes faster, it might get some load, whereas previously it did not get any load.

If a machine which is not the fastest but which receives some load changes its bid, it (possibly) changes its place in the speed ranking of the machines, and thus can only get more load if it gets faster and less when it gets slower. Note that the schedule that our algorithm produces does not change so far, because in Step 4 our algorithm produces the same job sets each time (since List Scheduling is applied on identical machines, and the lexicographically smallest schedule with minimal maximum load is used).

If the machine which is already fastest becomes even faster, this might lead to some other machines being dropped from consideration. If *only* the fastest machine remains (that is, there are

at most $m^{2/3}$ machines with speed at least α times the maximum speed), it clearly gets more load than before, because it now gets all the load.

Otherwise, the *largest load* (which is the load on the fastest machine, that we are considering) does not decrease, because our algorithm considers all options of using $m^{2/3}+1, \ldots, i$ machines where i is the number of machines that are not ignored with the old speeds. Thus if the largest load is smaller with the new amount of machines, we would have used this amount of machines earlier even though we had more machines available.

Conversely, if the fastest machine becomes slower, we have the same process as above in reverse. In all cases, our algorithm is monotone. □

Theorem 2 *For $\alpha = 1/(\phi m^{1/3})$, this algorithm has an approximation ratio of $\phi m^{2/3}(1+o(1))$.*

Proof Scale the job sizes such that the optimal makespan is 1. If our algorithm uses only one machine, then the last $m - m^{2/3}$ machines all have speed at most α. Thus the optimal load on the first $m^{2/3}$ machines is at most 1, and the optimal load on the remaining machines is at most α. Furthermore, there are no gaps in the schedule produced by our algorithm. Thus it has a makespan of at most

$$x(m) = m^{2/3} + (m - m^{2/3})\alpha,$$

compared to an optimal makespan of 1. Since $\alpha = 1/(\phi m^{1/3})$, we have $\lim_{m\to\infty} x(m)/m^{2/3} = \phi$.

If our algorithm uses i machines, we have that the makespan on *identical* machines is at most $1+(m-1)/i$ times optimal according to Graham [12]. This expression is maximized for $i = m^{2/3}+1$, which is the smallest value of i that our algorithm uses (besides 1).

Since the algorithm uses machines of speeds at least $\alpha < 1$ instead of machines of speed 1, the actual makespan is at most

$$y(m) = \frac{1}{\alpha}\left(1 + \frac{m-1}{m^{2/3}+1}\right)$$

times the optimal makespan on identical machines *of speed 1*, and therefore certainly at most $y(m)$ times the optimal makespan on the actual (slower) machines. Note that in this argument, we only use that all machines that are used have speed at least α. Thus the fact that we reorder the loads in Step 4 does not affect our argument. We have $\lim_{m\to\infty} y(m)/m^{2/3} = \phi$, where $\phi = 1.618\ldots$ □

4 Open questions

An obvious open question is to improve this approximation ratio. However, finding a better approximation ratio in the context of selfish machines does not seem easy. In particular, the approach of Chudak and Shmoys [8] does not seem suitable because we do not know how the output of the linear programming relaxation changes when the speeds of the machines change. On the other hand, the lower bound introduced by Chekuri and Bender [7] is a complicated formula of the speeds, for which it is also not easy to analyse the change when one of these speeds changes.

5 Acknowledgment

The research of the third author was supported by the Alexander von Humboldt Foundation.

References

[1] Nir Andelman, Yossi Azar, and Motti Sorani. Truthful approximation mechanisms for scheduling selfish related machines. In *Proc. of 22nd International Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 69–82, 2005.

[2] Aaron Archer. *Mechanisms for Discrete Optimization with Rational Agents.* PhD thesis, Cornell University, 2004.

[3] Aaron Archer, Christos Papadimitriou, Kunal Talwar, and Eva Tardos. An approximate truthful mechanism for combinatorial auctions with single parameter agents. In *Proc. of 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 205–214, 2003.

[4] Aaron Archer and Eva Tardos. Truthful mechanisms for one-parameter agents. In *Proc. 42nd Annual Symposium on Foundations of Computer Science*, pages 482–491, 2001.

[5] Aaron Archer and Eva Tardos. Frugal path mechanisms. In *Proc. of 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 991–999, 2002.

[6] Vincenzo Auletta, Roberto De Prisco, Paolo Penna, and Giuseppe Persiano. Deterministic truthful approximation mechanisms for scheduling related machines. In *Proc. of 21st International Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 608–619, 2004.

[7] Chandra Chekuri and Michael A. Bender. An efficient approximation algorithm for minimizing makespan on uniformly relate d machines. *Journal of Algorithms*, 41:212–224, 2001.

[8] Fabián A. Chudak and David B. Shmoys. Approximation algorithms for precedence-constrained scheduling problems o n parallel machines that run at different speeds. *Journal of Algorithms*, 30(2):323–343, 1999.

[9] Nikhil R. Devanur, Milena Mihail, and Vijay V. Vazirani. Strategyproof cost-sharing mechanisms for set cover and facility location games. In *ACM Conference on E-commerce*, pages 108–114, 2003.

[10] Edith Elkind, Amit Sahai, and Ken Steiglitz. Frugality in path auctions. In *Proc. of 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 701–709, 2004.

[11] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the theory of of NP-Completeness.* Freeman and Company, San Francisco, 1979.

[12] Ronald L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical J.*, 45:1563–1581, 1966.

[13] Jeffrey M. Jaffe. Efficient scheduling of tasks without full use of processor resources. *Theoretical Computer Science*, 12:1–17, 1980.

[14] Annemaria Kovacs. Fast monotone 3-approximation algorithm for scheduling related machines. In *Proc. of 13th Annual European Symposium on Algorithms (ESA)*, pages 616–627, 2005.

[15] Daniel J. Lehmann, Liadan O'Callaghan, and Yoav Shoham. Truth revelation in rapid, approximately efficient combinatorial auctions. In *ACM Conference on Electronic Commerce*, pages 96–102, 1999.

[16] Ahuva Mu'alem and Noam Nisan. Truthful approximation mechanisms for restricted combinatorial auctions. In *Proc. of the 18th National Conference on Artificial Intelligence and 14th Conference on Innovative Applications of Artificial Intelligence (AAAI/IAAI)*, pages 379–384, 2002.