Title:	Online Preemptive Scheduling on Parallel Machines
Name:	Jiří Sgall
Affil./Addr.	Computer Science Institute of Charles University, Faculty of Mathematics and Physics, Praha, Czech Republic, sgall@iuuk.mff.cuni.cz
Keywords:	online scheduling; makespan; preemption
SumOriWork:	2009; Ebenlendr, Jawor, Sgall 2010; Ebenlendr 2011; Ebenlendr, Sgall

Online Preemptive Scheduling on Parallel Machines

Jiří Sgall

Computer Science Institute of Charles University, Faculty of Mathematics and Physics, Praha, Czech Republic, sgall@iuuk.mff.cuni.cz

Years aud Authors of Summarized Original Work

2009; Ebenlendr, Jawor, Sgall
2010; Ebenlendr
2011; Ebenlendr, Sgall

Keywords

online scheduling; makespan; preemption

Problem Definition

We consider an online version of the classical problem of preemptive scheduling on uniformly related machines.

We are given m machines with *speeds* $s_1 \geq s_2 \geq \ldots \geq s_m$ and a sequence of jobs, each described by its *processing time* (length). The actual time needed to process a job with length p on a machine with speed s is p/s. In the *preemptive version*, each job may be divided into several pieces, which can be assigned to different machines in disjoint time slots. (A job may be scheduled in several time slots on the same machine, and there may be times when a partially processed job is not running at all.) The objective is to find a schedule of all jobs in which the *maximal completion time (makespan)* is minimized.

In the *online problem*, jobs arrive one-by-one and the algorithm needs to assign each incoming job to some time slots on some machines, without any knowledge of the jobs that arrive later. This problem, also known as list scheduling, was first studied by Graham [8] for identical machines (i.e., $s_1 = \ldots = s_m = 1$), without preemption. In the preemptive version, upon arrival of a job its complete assignment at all times must

be given and the algorithm is not allowed to change this assignment later. In other words, the online nature of the problem is in the order of the input sequence and it is not related to possible preemptions and the time in the schedule.

Key Results

The main result is an optimal online algorithm RatioStretch for preemptive scheduling on uniformly related machines [4]. RatioStretch achieves the best possible competitive ratio not only in the general case, but also for any number of machines and any particular combination of machine speeds. Although RatioStretch is deterministic, its competitive ratio matches the best competitive ratio of any randomized algorithm. This proves that randomization does not help for this variant of preemptive scheduling.

For any fixed set of speeds the competitive ratio of the algorithm RatioStretch can be computed by solving a linear program. However, its worst case value over all speed combinations is not known. Nevertheless, using the fact that there exists an e-competitive randomized algorithm [5], it is possible to conclude that RatioStretch also achieves the ratio of at most $e \approx 2.718$. The best lower bound shows that RatioStretch (and thus any algorithm) is not better than 2.112-competitive, by providing an explicit numerical instance on 200 machines [3].

Key Techniques

The idea of the algorithm RatioStretch is fairly natural. Suppose that the algorithm is given a ratio R which we are trying to achieve. For each arriving job, RatioStretch computes the optimal makespan for jobs that have arrived so far and runs the incoming job as slow as possible so that it finishes at R times the computed optimal makespan. There are many ways of creating such a schedule given the flexibility of preemptions. RatioStretch chooses a particular one based on the notion of a *virtual machine* from [5]. Given a schedule, the ith virtual machine at each time corresponds to the ith fastest real machine that is idle. (In particular, before the first job, the virtual machines are the real machines.) This assignment of the real machines to the virtual machines can vary at different times in the schedule. Due to preemption, a virtual machine can be thought of and used as a single machine with changing speed. The key idea of RatioStretch is to schedule each job on two adjacent virtual machines.

If RatioStretch fails on some input for a given R, it is possible to use the lower bound technique from [7] and show that there is no R-competitive algorithm. This implies that if the algorithm knows the optimal competitive ratio R, it never fails and thus it is R-competitive.

It remains to find the optimal competitive ratio R. Since the lower bound technique from [7] results in a linear condition, one can show that R can be computed by a linear program for each combination of speeds.

Semi-Online Scheduling

The algorithm RatioStretch can be extended to *semi-online* scenarios [6]. This term encompasses situations where some partial information about the input is given to the scheduler in advance. Already Graham [9] studied a semi-online variant of scheduling on identical machines: He proved that if the jobs are presented in non-increasing order

of their processing times, the approximation ratio of list scheduling decreases from 2 to 4/3. Since then numerous semi-online models of scheduling have been studied; typical examples include (sequences of) jobs with decreasing processing times, jobs with bounded processing times, sequences with known total processing time of jobs and so on. Most of these models can be viewed as online algorithms on a restricted set of input sequences.

RatioStretch can be generalized so that it is optimal for any chosen semi-online restriction. This means not only the cases listed above—the restriction can be given as an arbitrary set of sequences that are allowed as inputs. Again, for any semi-online restriction, RatioStretch achieves the best possible approximation ratio for any number of machines and any particular combination of machine speeds; it is deterministic, but its approximation ratio matches the best possible approximation ratio of any randomized algorithm. This result also provides a clear separation between the design of the algorithm and the analysis of the optimal approximation ratio. While the algorithm is always the same, the analysis of the optimal ratio depends on the studied restrictions.

For typical semi-online restrictions, the optimal ratio can again be computed by linear programs (with machine speeds as parameters). Then we can study the relations between the optimal approximation ratios for different semi-online restrictions and give some bounds for a large number of machines by analysis of these linear programs. One interesting result is that the overall ratio with known sum of processing times is the same as in the purely online case—even though for a small fixed number of machines knowing the sum provides a significant advantage.

Some basic restrictions form an inclusion chain: The inputs where the first job has the maximal processing time (which is equivalent to known maximal processing time) include the inputs with non-increasing processing times, which in turn include the inputs with all jobs of equal processing time. The restriction to non-increasing processing times gives the same approximation ratio as when all jobs have equal processing times, even for any particular combination of speeds. The overall approximation ratio of these two equivalent problems is at most 1.52. For known maximal processing time of a job there exists a computer-generated hard instance with approximation ratio 1.88 with 120 machines. Thus restricting the jobs to be non-increasing helps the algorithm much more than just knowing the maximal processing time of a job. This is very different from identical machines, where knowing the maximal processing time is equally powerful as knowing that all the jobs are equal, see [10].

Small Number of Machines

For two, three and sometimes four machines it is possible to give an exact formula for the competitive ratio for any speed combination [2, 3]. This is a fairly routine task which can be simplified (but not completely automated) using standard mathematical software. Once the solution is known, verification amounts to checking the given primal and dual solutions for the linear program.

Open Problems

The main remaining open problem is to develop techniques for determining or bounding the overall competitive ratio of the optimal algorithm RatioStretch. In particular, it would be interesting to obtain a tight bound in the online case.

It is also open if similar techniques can be used for the non-preemptive problem. In this case, the currently best algorithms were obtained by a doubling approach. This

means that a competitive algorithm is designed for the case when the optimum is approximately known in advance, and then, without this knowledge, it is used in phases with geometrically increasing guesses of the optimum. Such an approach probably cannot lead to an optimal algorithm for this type of scheduling problems. The best lower and upper bounds for non-preemptive scheduling on uniformly related machines are 2.438 and 5.828 for deterministic algorithms, see [1], and 2 and 4.311 for randomized algorithms, see [7, 1]. Thus it is still open whether randomized algorithms are better than deterministic.

Cross-References

List Scheduling
Load Balancing
Minimum Makespan on Unrelated Machines

Recommended Reading

1. P. Berman, M. Charikar, and M. Karpinski. On-line load balancing for related machines. *J. Algorithms*, 35:108–121, 2000.
2. T. Ebenlendr. Semi-online preemptive scheduling: Study of special cases. In *Proc. 8th Int. Conf. on Parallel Processing and Applied Mathematics (PPAM 2009), Part II*, volume 6068 of *Lecture Notes in Comput. Sci.*, pages 11–20. Springer, 2010.
3. T. Ebenlendr. *Combinatorial algorithms for online problems: Semi-online scheduling on related machines*. PhD thesis, Charles University, Prague, 2011.
4. T. Ebenlendr, W. Jawor, and J. Sgall. Preemptive online scheduling: Optimal algorithms for all speeds. *Algorithmica*, 53:504–522, 2009.
5. T. Ebenlendr and J. Sgall. Optimal and online preemptive scheduling on uniformly related machines. In *Proc. 21st Symp. on Theoretical Aspects of Computer Science (STACS)*, volume 2996 of *Lecture Notes in Comput. Sci.*, pages 199–210. Springer, 2004.
6. T. Ebenlendr and J. Sgall. Semi-online preemptive scheduling: One algorithm for all variants. *Theory Comput. Syst.*, 48:577–613, 2011.
7. L. Epstein and J. Sgall. A lower bound for on-line scheduling on uniformly related machines. *Oper. Res. Lett.*, 26:17–22, 2000.
8. R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical J.*, 45:1563–1581, 1966.
9. R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM J. Appl. Math.*, 17:263–269, 1969.
10. S. Seiden, J. Sgall, and G. J. Woeginger. Semi-online scheduling with decreasing job sizes. *Oper. Res. Lett.*, 27:215–221, 2000.