Open Problems in Throughput Scheduling

Jiří Sgall[⋆]

Computer Science Institute of Charles University, Faculty of Mathematics and Physics, Malostranské nám. 25, CZ-11800 Praha 1, Czech Republic.
sgall@iuuk.mff.cuni.cz

Abstract. In this talk we survey the area of scheduling with the objective to maximize the throughput, i.e., the (weighted) number of completed jobs. We focus on several open problems.

1 Introduction

Scheduling is a very wide research area with a multitude of variants that may appear confusing to non-specialists. The first rough classification is based on the objective that the algorithms try to optimize. Two most studied classes of problems try to minimize the costs generally depending on the completion times of jobs. One class minimizes a max-type objective, typically the length of the schedule (makespan). The other class minimizes a sum-type objective, typically the sum (average) of completion times or flow times (the time a job is in the system).

In this talk we focus on another area of scheduling. Our objective is to maximize the benefit. The benefit is throughput, i.e., the number of scheduled jobs, or the weighted number of scheduled jobs, in case when jobs have profits individually given by their weights. This area received considerable attention in the last decade esp. in the online case, with motivation such as packet forwarding in network switches and power management.

In turns out that some of very basic and easily formulated problems in this area are still open. We center our talk around a few of such problems. Needless to say, any such selection is necessarily biased. Our choice is motivated mostly by simplicity and certain elegance of the problems.

In each of the considered variants the jobs are given by several parameters. We use the following notation: for a job j, p_j is its processing time also called length, r_j its release time, d_j its deadline, and w_j is its weight. There may be a single machine or m machines. A schedule assigns to each job a machine and a time slot of length p_j starting not before r_j and ending no later than d_j; some jobs may remain unscheduled. The time slots on each machine have to be non-overlapping. The objective is to maximize the sum of the weights w_j of all the scheduled jobs.

[⋆] Partially supported by the Center of Excellence – Inst. for Theor. Comp. Sci., Prague (project P202/12/G061 of GA ČR) and grant IAA100190902 of GA AV ČR

We follow the standard convention that the job parameters are non-negative integers. Then we may assume that all jobs are started at integral time, as rounding the start times of all jobs down preserves feasibility.

We consider both offline and online problems. In the online setting, we let the time progress and at each time t we learn about jobs released at time t, i.e., those with $r_j = t$; at this time also all the job parameters of j become known. At time t, we may start some pending job (with $r_j \leq t$, $t+p_j \leq d_j$, and not running or completed) on one of the available machines (those not running any job). Sometimes we may also use preemption, i.e., stop the currently running job; in variants that we consider this job is then lost and cannot be even restarted.

We consider some variants of this general problem. Typically the set of possible job lengths is restricted. In the most restricted case, *unit-length jobs*, all job lengths are equal to 1. This corresponds to a special case of maximum matching on bipartite graph (where the vertices correspond to jobs and time slots), thus it can be solved to optimality offline. A less restricted case is that of *equal-length jobs* where all the job lengths are equal to some p (since release times and deadlines are integral, this is different from unit-length jobs). In several variants we have unit weights of jobs, which is equivalent to the objective of maximizing the number of scheduled jobs instead of the total weight. In some of the variants we allow parallel jobs. Then each job comes with an additional parameter $size_j$ and requires to be given the same time slot on $size_j$ machines.

2 Offline scheduling

2.1 Instances with length of jobs bounded by a constant

If we allow arbitrary processing times, the problem is (strongly) NP-hard; it is even hard to check if all jobs can be scheduled in the unweighted version on a single machine by an easy reduction from 3-PARTITION. To avoid this reason for NP-hardness, a natural option is to reduce the number of job lengths. Surprisingly, very little is known about this case.

Open problem 1 *Consider scheduling on a single machine, with the restriction that $p_j \leq C$ for some constant C. Is there a polynomial-time algorithm to compute a schedule maximizing the number of completed jobs? Is it NP-hard to compute the maximal weight of scheduled jobs for some C?*

We do not even know a polynomial-time algorithm for the unweighted variant for $C = 2$, i.e., every job needs one or two time slots. The only positive result known is that for $C = 2$, it is possible to check in polynomial time if it is possible to schedule all jobs. (Since we are scheduling all the jobs, it makes no difference whether the weights are unit or arbitrary.) This can be strengthened to the case when $p_j \in \{1, p\}$ for any fixed p. This result follows closely the methods from [12], but it appears to be new. We prove it in Section 4. Note that the restriction that one of the allowed job lengths equals 1 is essential in our proof. Even in the case

$p_j \in \{2, 3\}$ it is not known whether checking feasibility of scheduling of all jobs is polynomial!

In case of equal length jobs, the problem of maximizing the number of scheduled jobs is still far from trivial. On a single machine it is polynomial [3, 7], but on parallel machines, it is known to be polynomial only if the number of machines m is a constant [1]. The following is still open:

Open problem 2 *Consider scheduling of equal-length jobs on m parallel machines with m a part of the input. Is there a polynomial-time algorithm to compute a schedule maximizing the number of completed jobs? Is it NP-hard to compute the maximal weight of scheduled jobs?*

2.2 Unit-length parallel jobs

If we allow parallel jobs but on the other had restrict the instances to the case of unit-length jobs, the situation is very similar to the case of restricted job lengths on a single machine. Indeed, instead of using m parallel machines, one can use a single machine and scale all the release times and deadline by a factor of m; the length of parallel jobs is also scaled to $p_j = m$. This transformation does not give an exact formal reduction between these two models, but intuitively they are very close to each other.

Open problem 3 *Consider scheduling of unit-length parallel jobs on m parallel machines. Is there a polynomial-time algorithm to compute a schedule maximizing the number of completed jobs for some fixed $m \geq 2$? Is it NP-hard to compute the maximal weight of scheduled jobs if m is a part of the input and the job sizes are bounded by $size_j \leq C$ for some constant C?*

We do not even know a polynomial-time algorithm for the unweighted variant for $m = 2$, i.e., there are only two machines and naturally every job needs one or two of them.

The best positive result known is that for the so called tall/small jobs, i.e., the case when $size_j \in \{1, m\}$, we can check the feasibility of scheduling of all jobs [2, 12]. (The same proof actually works even if $size_j \in \{1, p\}$ for some p that divides m.) However, again, if $size_j \in \{2, 3\}$ it is not even known whether checking feasibility of scheduling of all jobs is polynomial.

Another interesting case is the restriction of job sizes to the powers of two, which is motivated by scheduling parallel jobs on hypercubes. Even though the number of job sizes is unbounded, this restriction also avoids the hardness proof by the reduction from partition-type problems. Here it is known that the number of completed jobs can be maximized in polynomial time if all of them are released at the same time [23], or, more generally, if the intervals $[r_j, d_j]$ are nested [24]. It would be very nice to generalize this to the general release times; this would be much stronger than a positive solution of the previous open problem for $m = 2$.

3 Online scheduling

The most natural online algorithm is often some greedy strategy. For our problems, it means that at each time we choose to schedule the pending job with the maximal weight. For equal-length jobs with unit weights or unit jobs with general weight, such an algorithm can be shown to be 2-competitive by a fairly generic argument: We "charge" each job in the optimal schedule to some job in the greedy schedule; either to the job that the greedy schedule runs at the same time (and machine), or to the same job if greedy schedule schedules it earlier than or at the same time as the optimum. Each job receives at most one charge of each type, furthermore all charges go to a job with weight at least the weight of the charged job. The challenge is then to improve the competitive ratio below 2.

3.1 Unit-length jobs: Buffer management and dynamic queues

The online variant of unit-length jobs is probably the most extensively studied variant of throughput scheduling. It is motivated by buffer management in network elements, as the unit-length jobs represent packets of some fixed length and different weights and deadlines represent their priorities.

The generic charging argument implies a 2-competitive algorithm, but to improve this upper bound, the algorithm has to deal with the choice of scheduling either a heavy job (possibly with a large deadline) or an urgent job (possibly with a small weight). The first better than 2-competitive algorithm for this problem was randomized; its competitive ratio is $e/(e-1) \approx 1.582$ which is still the best upper bound [6, 19]. An improved deterministic algorithm needs a significantly more delicate argument; the first such algorithm has competitive ratio $64/33 \approx 1.939$ [8] and the currently best upper bound is $2\sqrt{2}-1 \approx 1.828$ [14].

In both cases this is quite far from the lower bound, which is 1.25 for randomized algorithms and $\phi \approx 1.618$ for deterministic ones. In both cases, the lower bound instances are 2-bounded, which means that each job has $d_j - r_j \leq 2$, so that it has to be scheduled either immediately upon its arrival or in the next time step. This seems to be a very restricted case, so one would expect that larger lower bounds should be possible for more general instances.

On the other hand, the 2-bounded case is generalized by the case of agreeable deadlines (also called similarly ordered) which includes all instances which can be ordered so that both the sequence of release times and the sequence of deadlines are non-decreasing with j. In this substantially more general case there exist better algorithms, namely 4/3 [18, 20] randomized one and ϕ-competitive deterministic one [21, 20]; the latter even matches the lower bound.

In light of this, the question is much less clear and we state it as our next problem. Of course, any improvement of any of the bounds would be nice, but the question whether the 2-bounded case is as hard as the general case is particularly interesting.

Open problem 4 *Does there exist an algorithm for scheduling of arbitrary unit-length jobs on a single machine that matches the performance of algorithms* *for the special case when $d_j - r_j \leq 2$? That is* 1.618*-competitive deterministic algorithm and/or* 1.25*-competitive algorithm?*

The randomized algorithms use only the relative order of the deadlines, not their actual values. This is not true for the deterministic algorithms, and for a long time the use of exact values of deadlines seemed to be essential. However, it turns out that even in the model of dynamic queues, which does allow the algorithm to use only the relative order of deadlines, a 1.897-competitive algorithm exists [5].

3.2 Equal jobs

In this section we focus on the case when all the jobs have both equal length and unit weights. The generic charging argument again implies that any greedy strategy is 2-competitive [4], and in fact on a single machine no deterministic algorithm can be better [16].

One possibility to improve the algorithm is to use randomization. We know that there exists a 5/3-competitive randomized algorithm [9] for a single machine. This algorithm actually uses only a single bit of randomness: It generates online two schedules (connected by a lock mechanism) and randomly chooses to follow one of them from the beginning. This is still far from the best lower bound, which is only 4/3 for randomized algorithms on a single machine [16].

Open problem 5 *Suppose that all jobs have equal processing time $p_j = p$ and unit weights. What is the best competitive ratio of a randomized algorithm on a single machine with the objective to maximize the (expected) number of completed jobs?*

If we have more machines, the problem appears to be easier. At the first sight this may be surprising, but it is natural: If the online algorithm receives a single job with a large deadline, it has to decide to schedule it at some point. This blocks the only available machine, and the adversary may exploit this by releasing a job with a tight deadline in the next step. In contrast, already with two machines, the algorithm can try to keep one of them as a reserve for such tight jobs. This vaguely corresponds to generating two schedules and randomly choosing one of them, although we are not aware of any formal relationship of the two problems.

For two machines, it is known that the optimal competitive ratio of a deterministic algorithm is 3/2 [11, 17]. For more machines, much less is known. The lower bound for deterministic algorithm approaches 6/5 from above for $m \to \infty$ [11]. The best algorithm has a competitive ratio that approaches $e/(e-1) \approx 1.582$ from above for $m \to \infty$ [10]. This algorithm actually works in a much more restricted model in which upon the release of each job it is immediately decided if and when it is scheduled. We note that in this restricted model with immediate decision, no better algorithm for $m \to \infty$ is possible even for unit-length jobs [13].

Open problem 6 *Suppose that all jobs have equal processing time $p_j = p$ and unit weights. For $m \to \infty$, find either a better than $e/(e-1)$-competitive deterministic algorithm or a lower bound larger than $6/5$.*

3.3 Fixed start times

A special case of the general case is the problem of *interval scheduling*, where all the jobs are tight, i.e., $p_j = d_j - r_j$. This means that upon its release, a job must be started or it is lost. A classical study [22] shows that for variety of cases a 4-competitive preemptive deterministic algorithm exists and this is tight; this includes the case of unit-length jobs (with arbitrary weights) and jobs with weight equal to their length.

This 4-competitive algorithm was recently extended to a variant of parallel machines with speeds (uniformly related machines) [15]. Here each machine has a speed s_i and a job j needs time slot of length p_j/s_i to be run on this machine. Instead of formulating the problem as interval scheduling (which has no clear meaning, as tight jobs cannot be defined), we use the other equivalent formulation: Each job has to be started on one of the machines immediately upon its release, or else it is lost. We call this variant *scheduling with fixed start times.*

We conclude by a restricted variant of scheduling with fixed start times where we cannot even determine the competitive ratio of the greedy algorithm. In this case we have equal jobs once again (i.e., equal lengths and unit weights). Obviously, if we have a single machine or parallel machines with equal speeds, any greedy strategy is optimal. However, this is no longer true with speeds. Actually, with speeds there is no better than 3/2-competitive algorithm for $m \to \infty$ [15]. Any greedy algorithm is 2-competitive by the generic charging argument. However, no better upper bound is known for $m \geq 3$ for any algorithm. The most natural greedy algorithm always starts a job on the fastest available machine. This turns out to be 4/3-competitive and optimal for $m = 2$, but for $m \to \infty$ we only know that the competitive ratio is at least $25/16 = 1.5625$.

Open problem 7 *Suppose that all jobs have equal processing times and unit weights. For $m \to \infty$ machines with speeds, find a better than 2-competitive deterministic algorithm for scheduling with fixed start times.*

4 Checking feasibility with two job lengths

Now we return to the offline problem and give a polynomial-time algorithm for deciding whether all the jobs can be scheduled if the lengths are restricted to $\{1, p\}$ for some p.

Theorem 1. *Suppose that our input consists of p and an instance with $p_j \in \{1, p\}$ for all jobs. Then there exists an algorithm which in polynomial time decides if there exists a feasible schedule that schedules all jobs.*

Proof. Let us call the jobs with $p_j = 1$ short and the jobs with $p_j = p$ long. For two times s, t let $A_{s,t} = \{j \mid p_j = 1 \wedge s \leq r_j \wedge d_j \leq t\}$, $a_{s,t} = |A_{s,t}|$, $B_{s,t} = \{j \mid p_j = 2 \wedge s \leq r_j \wedge d_j \leq t\}$, and $b_{s,t} = |B_{s,t}|$. I.e., $a_{s,t}$ and $b_{s,t}$ denote the number of short jobs and long jobs, respectively, that need to be scheduled (started and completed) between s and t.

We may assume that $r_j + p_j \leq d_j$ for all jobs; otherwise the instance is infeasible. We may also assume that

$$a_{s,t} \leq t - s \tag{1}$$

for all t and s, as otherwise the instance is again infeasible.

We now formulate a linear program (2)–(5) that should describe the feasible schedules. It has no objective function, we are only interested in its feasibility. The intended meaning of the variable x_t is the number of long jobs started strictly before time t. For convenience, we set $x_t = 0$ for any $t \leq 0$. Let D be the maximal deadline.

$$\forall t \in \{1, \dots, D\}: \qquad x_t - x_{t-1} \geq 0 \tag{2}$$

$$\forall t \in \{p, \dots, D\}: \qquad x_t - x_{t-p} \leq 1 \tag{3}$$

$$\forall s, t \in \{0, \dots, D\},\ \ s + p \leq t: \qquad x_{t+1-p} - x_s \leq \left\lfloor \frac{t - s - a_{s,t}}{p} \right\rfloor \tag{4}$$

$$\forall s, t \in \{0, \dots, D\},\ \ s + p \leq t: \qquad x_{t+1-p} - x_s \geq b_{s,t} \tag{5}$$

Inequalities (2) make sure that x_t form a non-decreasing sequence, (3) that at most one long job is started in any p adjacent steps and thus the long jobs do not overlap. Inequalities (4) and (5) make sure that sufficiently many slots are available for short and long jobs in each interval.

The crucial observation is that the matrix of our linear program is totally unimodular. Indeed, every row contains at most single -1 and at most single $+1$ and such matrices are known to be totally unimodular. This implies that if the linear program is feasible, there exists an integral solution.

We now prove that the linear program is feasible if and only if there exist a feasible schedule.

Suppose we have a feasible schedule. As noted in the introduction, we may assume that all jobs are started at integral times. Then for (integral) s such that the schedule starts a long job at time s we put $x_{s+1} = x_s + 1$ and for all other s we put $x_{s+1} = x_s$. It is easy to check that this is a feasible solution of our linear program. Inequalities (2) and (3) are trivial. For (4) note that $x_{t+1-p} - x_s$ long jobs are started and completed in the slots $s, \dots, t-1$, thus they occupy $p(x_{t+1-p} - x_s)$ of these $t - s$ slots and at least $a_{s,t}$ of these slots are taken by the short jobs. Thus $p(x_{t+1-p} - x_s) \leq t - s - a_{s,t}$ and (4) follows from the integrality of x. Similarly, (5) follows from the fact that at least $b_{s,t}$ long jobs are started and completed in the slots $s, \dots, t-1$ in the feasible schedule.

Suppose now that the linear program is feasible and let us fix its integral solution x. The solution x determines at which time slots a long or short job can start. More precisely, let $S_1 = \{r \in \{0,\dots,D-1\} \mid x_{r+1-p} = x_{r+2-p} = \dots = x_r = x_{r+1}\}$ and $S_2 = \{r \in \{0,\dots,D-1\} \mid x_{r+1} > x_r\}$. Note that S_2 are exactly the slots where a long job should start and S_1 are exactly the slots where no long job started at time $r \in S_2$ can be running. Now we construct the schedule greedily increasing r from $r = 0$. If $r \in S_1$ then we start a pending short job with the smallest deadline. If $r \in S_2$, then we start a pending long job with the smallest deadline. In both cases we break ties arbitrarily. The definitions of S_1 and S_2 together with (2) and (3) guarantee that no two jobs overlap in the schedule.

It remains to show that the schedule completes all the jobs before their deadlines. For a contradiction, assume that j is a job with $d_j = t$ that is not completed before t and t is minimal. We distinguish two cases.

Case 1: j is a long job. Let $s-1 \in S_2$ be the largest time before t such that at time $s-1$ either no job is started or a job with the deadline larger than t is started. Let $s = 0$ if no such time exists. This implies that $r_j \geq s$ as otherwise we would have started j at time $s-1$. At any time in $S_2 \cap \{s,\dots,t-p\}$, a long job with a deadline at most t is started by the choice of s; also any of these jobs is released at s or later (as otherwise we would have started it at $s-1$). Thus all these jobs and j belong to $B_{s,t}$. We have $b_{s,t} \geq 1 + |S_2 \cap \{s,\dots,t-p\}| = 1 + x_{t+1-p} - x_s$, contradicting (5).

Case 2: j is a short job. Let t' be the first time in S_1 such that $t' \geq t$. Note that j is not completed at time t'. Let $s-1 \in S_1$ be largest time before t such that at time $s-1$ either no job is scheduled or a job with the deadline larger than t is scheduled. Let $s = 0$ if no such time exists. This implies that $r_j \geq s$ as otherwise we would have started j at time $s-1$. Similarly as in Case 1, at any time in $S_1 \cap \{s,\dots,t-1\} = S_1 \cap \{s,\dots,t'-1\}$, a short job with a deadline at most t is scheduled by the choice of s; also any of these job is released at s or later (as otherwise we would have started it at $s-1$). Thus all these jobs and j belong to $A_{s,t} \subseteq A_{s,t'}$. Furthermore, since $s, t' \in S_1$, the number of time slots in $\{s,\dots,t'-1\} \setminus S_1$ is equal to $p(x_{t'+1-p} - x_s)$. We have $a_{s,t'} \geq 1 + |S_1 \cap \{s,\dots,t'-1\}| = 1 + (t'-s) - p(x_{t'+1-p} - x_s)$. It follows that

$$x_{t'+1-p} - x_s \geq \frac{1 + t' - s - a_{s,t'}}{p} > \frac{t' - s - a_{s,t'}}{p}.$$

This contradicts (4) if $s + p \leq t'$ or (1) if $s + p > t'$ (note that then $s \geq t' + 1 - p$ and $x_{t'+1-p} - x_s \leq 0$ by (2)).

The last problem that we need to solve is that the linear program as we have formulated it may have a superpolynomial size if p or the release times or the deadlines are superpolynomial. This is only a technical issue, as there are clearly only polynomially many "important" times. More precisely, we can modify any feasible schedule so that each job starts either at its release time or at the completion time of some other job (the schedule with the lexicographically minimal sequence of start times will satisfy this). Then all the jobs are started at times $r_j + \alpha \cdot p + \beta$ for some job j and $\alpha, \beta \in \{0,\dots,n\}$ where n is the number of jobs. There are $O(n^3)$ of such times. For all the other t, we set $x_{t+1} = x_t$.

By substituting these identities, we get a linear program with only polynomially many variables and also polynomially many distinct constraints, and it can be even written down efficiently. Thus feasibility can be checked in polynomial time.

In fact, we have not been very efficient in our proof. As shown in [12], instead of solving a linear program, we can use a shortest path computation, as the matrix of the linear program is an adjacency matrix of a directed graph. □

References

1. Baptiste, P., Brucker, P., Knust, S., Timkovsky, V.: Ten notes on equal-execution-time scheduling. 4OR **2** (2004) 111–127
2. Baptiste, P., Schieber, B.: A note on scheduling tall/small multiprocessor tasks with unit processing time to minimize maximum tardiness. J. Sched. **6** (2003) 395–404
3. Baptiste, P.: Polynomial time algorithms for minimizing the weighted number of late jobs on a single machine with equal processing times. J. Sched. **2** (1999) 245–252
4. Baruah, S.K., Haritsa, J., Sharma, N.: On-line scheduling to maximize task completions. J. Comb. Math. Comb. Comput. **39** (2001) 65–78
5. Bieńkowski, M., Chrobak, M., Dürr, C., Hurand, M., Jeż, A., Jeż, Ł., Stachowiak, G.: Collecting weighted items from a dynamic queue. To appear in ACM Trans. Algorithms
6. Chin, F.Y.L., Chrobak, M., Fung, S.P.Y., Jawor, W., Sgall, J., Tichý, T.: Online competitive algorithms for maximizing weighted throughput of unit jobs. Journal of Discrete Algorithms **4** (2006) 255–276
7. Chrobak, M., Dürr, C., Jawor, W., Kowalik, Ł., Kurowski, M.: A note on scheduling equal-length jobs to maximize throughput. J. Sched. **9** (2006) 71–73
8. Chrobak, M., Jawor, W., Sgall, J., Tichý, T.: Improved online algorithms for buffer management in qos switches. ACM Trans. Algorithms **3**(4) (2007) Article No. 50 (19 pages)
9. Chrobak, M., Jawor, W., Sgall, J., Tichý, T.: Online scheduling of equal-length jobs: Randomization and restarts help. SIAM J. Comput. **36** (2007) 1709–1728
10. Ding, J., Ebenlendr, T., Sgall, J., Zhang, G.: Online scheduling of equal-length jobs on parallel machines. In: Proc. 15th European Symp. on Algorithms (ESA). Volume 4698 of Lecture Notes in Comput. Sci., Springer (2007) 427–438
11. Ding, J., Zhang, G.: Online scheduling with hard deadlines on parallel machines. In: Proc. 2nd International Conf. on Algorithmic Aspects in Information and Management (AAIM). Volume 4041 of Lecture Notes in Comput. Sci., Springer (2006) 32–42
12. Dürr, C., Hurand, M.: Finding total unimodularity in optimization problems solved by linear programs. In: Proc. 14th European Symp. on Algorithms (ESA). Volume 4168 of Lecture Notes in Comput. Sci., Springer (2006) 53–64
13. Ebenlendr, T., Sgall, J.: A lower bound for scheduling of unit jobs with immediate decision on parallel machines. In: Proc. 6thInternational Workshop in Approximation and Online Algorithms(WAOA'08). Volume 5426 of Lecture Notes in Comput. Sci., Springer (2009) 43–52
14. Englert, M., Westerman, M.: Considering suppressed packets improves buffer management in QoS switches. In: Proc. 18th Symp. on Discrete Algorithms (SODA), ACM/SIAM (2007) 209–218

15. Epstein, L., Jeż, Ł., Sgall, J., van Stee, R.: Online scheduling of jobs with fixed start times on related machines. In: RANDOM-APPROX. Lecture Notes in Comput. Sci., Springer (2012) To appear.
16. Goldman, S.A., Parwatikar, J., Suri, S.: Online scheduling with hard deadlines. J. Algorithms **34** (2000) 370–389
17. Goldwasser, M.H., Pedigo, M.: Online, non-preemptive scheduling of equal-length jobs on two identical machines. In: Proc. 10th Scandinavian Workshop on Algorithm Theory (SWAT). Volume 4059 of Lecture Notes in Comput. Sci., Springer (2006) 113–123
18. Jeż, Ł.: Randomized algorithm for agreeable deadlines packet scheduling. In: Proc. 27th Symp. on Theoretical Aspects of Computer Science (STACS). (2010) 489–500
19. Jeż, Ł.: One to rule them all: A general randomized algorithm for buffer management with bounded delay. In: Proc. 19th European Symp. on Algorithms (ESA). Volume 6942 of Lecture Notes in Computer Science., Springer (2011) 239–250
20. Jeż, Ł., Li, F., Sethuraman, J., Stein, C.: Online scheduling of packets with agreeable deadlines. To appear in ACM Trans. Algorithms
21. Li, F., Sethuraman, J., Stein, C.: An optimal online algorithm for packet scheduling with agreeable deadlines. In: Proc. 16th Symp. on Discrete Algorithms (SODA), ACM/SIAM (2005) 801–802
22. Woeginger, G.J.: On-line scheduling of jobs with fixed start and end times. Theoret. Comput. Sci. **130** (1994) 5–16
23. Ye, D., Zhang, G.: Maximizing the throughput of parallel jobs on hypercubes. Inform. Process. Lett. **102** (2007) 259–263
24. Zajíček, O.: A note on scheduling parallel unit jobs on hypercubes. Int. J. on Found. Comput. Sci. **20** (2009) 341–349