Theory Comput Syst (2012) 50:124–146
DOI 10.1007/s00224-011-9336-5

Scheduling with Bully Selfish Jobs

Tami Tamir

Published online: 11 June 2011

Abstract In job scheduling with precedence-constraints, $i \prec j$ means that job j cannot start being processed before job i is completed. In this paper we consider *selfish bully jobs* who do not let other jobs start their processing if they are around. Formally, we define the *selfish precedence-constraint* where $i \prec_s j$ means that j cannot start being processed if i has not started its processing yet. Interestingly, as was detected by a devoted kindergarten teacher whose story is told below, this type of precedence-constraints is very different from the traditional one, in a sense that problems that are known to be solvable efficiently become NP-hard and vice-versa.

The work of our hero teacher, Ms. Schedule, was initiated due to an arrival of bully jobs to her kindergarten. Bully jobs bypass all other nice jobs, but respect each other. This natural environment corresponds to the case where the selfish precedence-constraints graph is a complete bipartite graph. Ms. Schedule analyzed the minimum makespan and the minimum total flow-time problems for this setting. She then extended her interest to other topologies of the precedence-constraints graph and other special instances with uniform length jobs and/or release times.

Keywords Scheduling · Approximation algorithms · Selfish precedence-constraints

1 Bully Jobs Arriving in Town

Graham school is a prestige elementary school for jobs. Jobs from all over the country are coming to spend their childhood in Graham school. Ms. Schedule is the kindergarten teacher and everybody in town admires her for her wonderful work with the little jobs. During recess, the jobs like to play outside in the playground. Ms. Schedule is known for her ability to assign the jobs to the different playground activities in

T. Tamir (✉)
School of Computer Science, The Interdisciplinary Center, Herzliya, Israel
e-mail: tami@idc.ac.il

a way that achieves many types of objectives (not all of them are clear to the jobs or to their parents, but this is not the issue of our story).

The jobs enjoy coming to school every morning. In addition to the national curriculum, they spend lot of time learning and practicing the rules Ms. Schedule is teaching them. For example, one of Ms. Schedules's favorite rules is called LPT [12]. They use it when playing on the slides in the playground. At first, each of the n jobs announces how long it takes him to climb up and slide down. Then, by applying the LPT rule they organize themselves quite fast (in time $O(n \log n)$) in a way that enables them to return to class without spending too much time outside. Ms. Schedule once told them that she will never be able to assign them to slides in a way that really minimizes the time they spend in the playground, but promised that this LPT rule provides a good approximation.

For years, everything went well at school. The jobs and their parents were very satisfied with the advanced educational program of Ms. Schedule, and the enrollment waiting list became longer and longer. Until the *bully jobs* came to town and joined the kindergarten.

Being very polite and well-mannered, the veteran jobs prepared a warm welcome party to the bully jobs. Ms. Schedule taught them the different kindergarten rules, and for the first few days no one noticed that the new jobs are different. It was only after a week that Ms. Schedule observed that the new bully jobs were not obeying the rules. Other jobs complained that sometimes, when they were waiting in lines, bully jobs passed them and climbed up the slide even if they were not first in line. One of the nice jobs burst into tears claiming that "I'm a very fast climber, according to SPT rule [21], I need to be first in line, but all these bully jobs are bypassing me". Indeed, Ms. Schedule herself noticed that the bully jobs were bypassing others. She also noticed that as a result, the whole kindergarten timetable was harmed. The jobs had to spend much more time outside until they had all completed sliding.

Ms. Schedule decided to have a meeting with the bully jobs' parents. In this meeting, it came clear to her that she will need to make a massive change in the kindergarten rules. The parents justified the inconsiderate behavior of their kids. "Our kids are *selfish*", they said, "they will never obey your current rules. They will always bypass all the other kids. You should better not try to educate them, just accept them as they are". Ms. Schedule was very upset to hear it, she was about to tell them that their kids must obey her rules, and otherwise will be suspended from school, but she was a bit afraid of their reaction,[1] so she promised them to devise new rules for the kindergarten. The parents were satisfied and concluded: "Remember, bully jobs always bypass those that are in front of them in line. They also move from one line to another. But we, bullies, respect each other! bully jobs will not pass other bully jobs that were assigned before them in line".

Ms. Schedule came back home tired and concerned, feeling she must design new rules for her kindergarten, taking into consideration what she have just learnt about the bully jobs.

[1] Bully jobs tend to have bully parents.

2 Ms. Schedule Defining Her Goals

Ms. Schedule relaxed with a cup of good green tea. She decided that the first thing she needed is a formal definition of her new model. "In my setting", she thought, "there is a set J of jobs, and a set M of m identical machines (slides). Each job is associated with a length (sliding time) p_j. Some of the jobs are *bully* and the other are *nice*. I will denote these sets B and N respectively, $B \cup N = J$. My rules assign jobs to slides, and determine the internal order of the jobs on each slide. The bully jobs, however, do not obey my assignment. Specifically, if a bully job can reduce its waiting time by passing other jobs in line or by moving to another line it will do so. On the other hand, bully jobs respect each other. If I assign them in some order to some line then their internal order will be kept. Moreover, if a bully moves to a different line, he will be last among the bullies who are already in line. Each of my assignment methods produces a schedule s of jobs on the slides, where $s(j) \in M$ denotes the slide job j is assigned to. The completion time, or flow-time of job j, denoted C_j, is the time when job j completes its processing. The load on a slide M_i in an assignment s is the sum of the sliding times of the jobs assigned to M_i, that is $\sum_{j|s(j)=M_i} p_j$. The *makespan* of a schedule, denoted C_{max}, is the load on the most loaded slide; clearly, this is also the maximal completion time of a job."

2.1 Scheduling with Selfish Precedence-Constraints

Ms. Schedule thought that her problem, in some sense, is similar to the problem of scheduling with precedence-constraints. In scheduling with precedence-constraints, the constraints are given by a partial order precedence relation $\prec$ such that $i \prec j$ implies that j cannot start being processed before i has been completed. Selfish-precedence is different. It is given by a partial order precedence relation $\prec_s$ such that $i \prec_s j$ implies that j cannot start being processed before i is starting.

"I believe" she thought "that selfish precedence-constraints induces interesting problems that should be studied, especially in these days when it is very popular to deal with algorithmic game theory and selfish agents. A selfish job only cares about his delay and his completion time, it is OK with him that others are also doing well, but he is ready to hurt others in order to promote himself. This is exactly reflected by the fact that if $i \prec_s j$, then job i doesn't mind if job j is processed in parallel with him, as long as it doesn't start being processed before him". Ms. Schedule decided to devote some of her valuable time to consider this new type of selfish precedence-constraints. "I'm not aware of any early work on this interesting setting", she mentioned to herself.

"For a single machine, I don't expect any interesting results." Ms. Schedule figured out, "It is easy to see that with a single machine, a schedule is feasible under the constraints $\prec$ if and only if it is feasible under the constraints $\prec_s$. Therefore all the results I see in my favorite web-site [1], carry over to selfish precedence-constraints."

"In fact, there is this issue of release times, which makes the precedence-constraints different, already with a single machine" Ms. Schedule kept pondering "since bully jobs only care about *their* delay, they let other jobs be processed as long as they are not around (before their release time). It is only when they show up that

they bypass others. Upon being released a bully job pushes a nice job away from the slide even if he already started climbing". Ms. Schedule decided to elaborate on that issue of release times later (see Sect. 5), and to set as a primal goal the analysis of the basic problems of minimum makespan (Sect. 3) and minimum total flow-time (Sect. 4) for the precedence constraint setting she has in her kindergarten. She also considered the special case of unit-length jobs with arbitrary precedence-constraints setting (Sect. 6). In this paper we tell her story and reveal her results.

2.2 Complete-Bipartite Selfish Precedence-Constraints and the Price of Bullying

"What I actually face", Ms. Schedule kept thinking, "is the problem in which the selfish precedence-constraints graph is a *complete bipartite* $K_{b,n}$, where $b = |B|$, $n = |N|$, and $i \prec_s j$ for every $i \in B$ and $j \in N$.

As a first step, I would like to evaluate the potential loss from having bully jobs in my kindergarten. Similar to other equilibria notions, a schedule is a *Bully equilibrium* if no bully job can reduce its completion time by migrating to another machine or bypassing nice jobs. Indeed, since bully jobs bypass all nice jobs in their line and can also migrate from one machine to another, a bully equilibrium schedule must respect the $K_{b,n}$ selfish precedence-constraints. On the other hand, a schedule might respect the $K_{b,n}$ constraints, but not be a bully-equilibrium. Nevertheless, as I show below, if this is the case, then the schedule can be trivially improved with respect to any reasonable objective function. Therefore, w.l.o.g., I would assume the following equivalence:"

Property 2.1 *W.l.o.g., a schedule is a bully equilibrium if and only if it obeys the $K_{b,n}$ selfish precedence-constraints.*

Proof As explained above, any bully equilibrium schedule obeys the $K_{b,n}$ selfish precedence-constraints. However, the other direction is not necessarily valid in any schedule. We show that if a schedule obeys the $K_{b,n}$ selfish precedence-constraints but is not a bully equilibrium, then it can be improved with respect to any reasonable objective function, in particular minimizing the makespan and the total flow-time. Assume that the precedence-constraints hold, but some bully job j can benefit from migrating from machine M_1 to machine M_2 (see Fig. 1(a)). For $i = 1, 2$, denote by B_i, N_i the sets of bully and nice jobs on M_i. W.l.o.g., j is the last bully job on M_1 (as any bully job located after j on M_1 also benefits from such a migration). As bully jobs respect each other, after the migration, j will be the last bully on M_2. Thus, it must hold that j starts its execution after the completion time of all bullies on M_2.

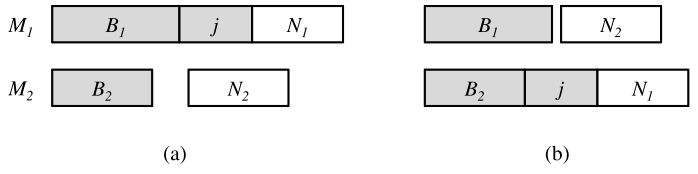

Fig. 1 Improving a non bully-equilibrium by simple migrations

Also, by the precedence-constraints, the first nice job on M_2 begins not earlier than j. Consider a schedule in which j and the jobs of N_1 migrate to M_2 and the jobs of N_2 migrate to M_1—starting at the same time as on M_2 (see Fig. 1(b)). Note that due to additional bully jobs, on additional machines, the starting time of N_1 might not be reduced by the same value as the starting time of j, but it cannot be later than its original starting time on M_1. The resulting schedule is clearly feasible. Moreover, the completion time of j and possibly of the jobs of N_1 is reduced, while the completion times of all other jobs remain the same. □

2.2.1 The Price of Bullying

Let $S(I)$ denote the set of all schedules of an instance I, not necessarily bully equilibria. For a schedule $s \in S(I)$, let $g(s)$ be some measure of s. For example, $g(s) = \max_{j \in I} C_j(s)$ is the makespan measure, and $g(s) = \sum_{j \in I} C_j(s)$ is the total flow time measure.

Definition 2.1 Let $\Phi(I)$ be the set of Bully equilibria of an instance I. The *price of bullying* (PoB) for a measure $g(\cdot)$ is the ratio between the best bully equilibrium and an optimal solution. Formally, $\mathrm{PoB}(I) = \min_{s \in \Phi(I)} g(s) / \min_{s \in S(I)} g(s)$.

Theorem 2.2 *For the objective of minimizing the makespan, the price of bullying is* $2 - \frac{1}{m}$.

Proof Consider an instance with $m(m-1)$ unit-length bully jobs and a single nice job of length m. An optimal non bully-equilibrium schedule assigns load m on each machine. On the other hand, in any bully-equilibrium, there are $m-1$ bully jobs on each machine (otherwise, some bully job can benefit from migrating from a machine with more than $m-1$ bully jobs into a machine with at most $m-2$ bully jobs). Thus, the long nice job starts its processing at time at least $m-1$, resulting in makespan at least $2m-1$. The resulting PoB is $\frac{2m-1}{m} = 2 - \frac{1}{m}$. The above instance achieves the maximal possible PoB since any bully equilibrium schedule is a possible output of List-Scheduling, which is known to provide a $(2 - \frac{1}{m})$-approximation to the minimum makespan [11] (see also Sect. 3.1). □

Theorem 2.3 *For the objective of minimizing the total flow-time, the price of bullying is* $(n+b)/m$.

Proof Consider an instance with $n = z \cdot m$ nice jobs of length ε, and $b = m$ bully jobs of length 1. An optimal, non bully-equilibrium schedule assigns on each machine z nice jobs followed by a single bully job. The total flow-time is $m + O(mz^2\varepsilon)$. In any bully-equilibrium, the bully jobs are scheduled first, one on each machine, and the total flow-time is at least $m(z+1) + O(mz^2\varepsilon)$. As ε approaches 0, the PoB approaches $z + 1 = (n+b)/m$. We show that the above instance achieves the highest possible PoB. Consider the best bully-equilibrium of any given instance. If $n + b < m$ then by assigning a single job to arbitrary $n + b$ machines we get $\mathrm{PoB} = 1$. Otherwise, we show that the total flow-time of any m jobs is at most $\sum_j p_j$. Let J_{last} be the set

of m jobs that are last on some machine. Note that when $n+b \geq m$ then no machine is idle in a best bully-equilibrium schedule, therefore J_{last} is well defined. These m jobs have total flow-time exactly $\sum_j p_j$. For any other set J_S of m jobs, it is possible to map each job in J_S to a different job in J_{last} with a not-smaller completion time. If the jobs in J_S are scheduled on different machines, then this mapping is trivial—a job scheduled on M_i is mapped to the last job on M_i. If there are several jobs in J_S on the same machine, say M_1, then there must be another machine, say M_2, from which there is no representative in J_S. Any non-last job on M_1 can be mapped to the last job on M_2. Even if M_2 completes before M_1, it must be that in any optimal bully-equilibrium, the gap between their completion times is not larger than the processing time of the last job on M_1 (otherwise, the last job on M_1 can migrate and improve the schedule). Therefore, the total flow-time of the jobs in J_S is at most the total flow-time of the jobs in J_{last}. Given that the total flow-time of any m jobs is at most $\sum_j p_j$, use averaging arguments to get that the total flow-time of any best bully-equilibrium schedule is at most $\frac{n+b}{m}\sum_j p_j$. Specifically, $n+b = km+r$ for some integers k, r. The r jobs with the highest flow time have total flow time at least $\frac{r}{m}\sum_j p_j$, any other set of m jobs has total flow time at least $\sum_j p_j$, thus the total flow time is at least $(k+\frac{r}{m})\sum_j p_j = \frac{n+b}{m}\sum_j p_j$. On the other hand, the optimal total flow time is at least $\sum_j p_j$, thus, the PoB is bounded by $(n+b)/m$. □

3 Makespan Minimization: $P|K_{b,n}, s\text{-}prec|C_{max}$

Ms. Schedule's first goal was to minimize the recess length. She wanted all jobs to have a chance to slide once. She knew that the problem $P\|C_{max}$ is strongly NP-hard, therefore, the best she could expect is a PTAS. A natural approach she considered is the following: Let $\mathcal{A}$ be an approximation algorithm for $P\|C_{max}$. Use $\mathcal{A}$ to schedule the bullies, then use $\mathcal{A}$ to schedule the nice jobs and glue the resulting schedules. The resulting algorithm will be denoted double-$\mathcal{A}$.

With regular precedence-constraints, an optimal solution for $P|K_{b,n}, prec|C_{max}$ consists of a concatenation of optimal solutions for each job-type, but with selfish precedence-constraints, this approach might not lead even to a good approximation. To clarify this point better, Ms. Schedule drew Fig. 2, and pointed out to herself that gluing independent optimal solutions for each job-type (denote this method double-OPT) can be far by a factor of at least $2-\frac{2}{m+1}$ from an optimal solution. For any number of machines m, the instance in Ms. Schedule's figure consists of a single nice job having length 1 (colored white in Fig. 2) and $2m-1$ bully jobs (colored grey), out of which, m have length $1/m$ and $m-1$ have length 1. For such an instance, if the minimum makespan problem is solved independently and optimally for each job-type, and the resulting schedules are glued, then the makespan is 2, while an optimal schedule of such an instance has makespan $(m+1)/m$. The reason for this relative poor performance of double-OPT is that in an optimal schedule, the bully jobs might better be assigned in a non-balanced way. Ms. Schedule decided to consider and analyze known heuristics and also to develop a PTAS for the problem.

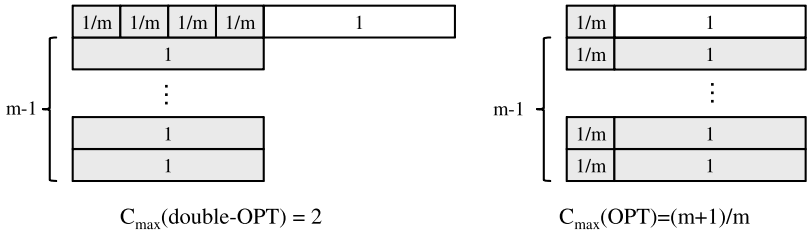

Fig. 2 A double-OPT schedule (*left*) vs. an optimal schedule (*right*). The approximation ratio is $2 - \frac{2}{m+1}$

3.1 Double List-Scheduling

Let the jobs stand in a single line, bully jobs first in arbitrary order followed by the nice job in arbitrary order. Then assign them to slides greedily according to this order. Each job goes to the first available slide. The starting times of the jobs form a non-decreasing sequence; in particular, every bully job is starting not later than every nice job. Therefore, the resulting schedule is feasible. Moreover, it is a possible execution of the known List Scheduling algorithm [11], therefore it produces a $(2 - \frac{1}{m})$-approximation to the makespan, even compared to the makespan with no selfish precedence-constraints. This bound is tight since an instance might include only jobs of one type (bully or nice), therefore, the known lower bound for list scheduling, of $2 - \frac{1}{m}$, applies here.

3.2 Double-LPT

Let the jobs stand in a single line, bully jobs first in non-increasing sliding-time order, followed by the nice jobs in non-increasing sliding-time order. Then assign them to slides greedily according to this order. Each job goes to the first available slide. As this is a possible execution of the adjusted list-scheduling algorithm, the resulting assignment is feasible. However, the actual approximation ratio of this heuristic is not much better than the $(2 - \frac{1}{m})$-guaranteed by list-scheduling, and does not resemble the known bounds of LPT (of $(\frac{4}{3} - \frac{1}{3m})$ [12] or $(1 + \frac{1}{k})$ where k is the number of jobs on the most loaded machine [5]). Since this algorithm follows the double-$\mathcal{A}$ approach, its approximation ratio cannot be better than $2 - \frac{2}{m+1}$. In particular, note that for the instance described in Fig. 2, the double-OPT schedule is achieved also by double-LPT.

Ms. Schedule was able to show that this ratio of $2 - \frac{2}{m+1}$ is the worst possible for double-LPT. Formally,

Theorem 3.1 *The approximation ratio of double-LPT for* $P|K_{b,n}, s\text{-}prec|C_{max}$ *is* $2 - \frac{2}{m+1}$.

Proof Let A_{LPT} be the assignment produced by double-LPT, and let J_k, of length p_k, be the job determining the makespan. W.l.o.g, J_k is the last job the double-LPT order (removing the later jobs can only reduce the optimal makespan). Let C_{LPT}, C^* denote the makespan of A_{LPT} and an optimal schedule respectively. If $J_k \in B$ then

all jobs are bullies and the regular analysis of LPT can be applied here. Since for every $m > 0$ it holds that $\frac{4}{3} - \frac{1}{3m} \leq 2 - \frac{2}{m+1}$, the claim is valid.

Therefore, assume $J_k \in N$. As all machines are busy at time $C_{LPT} - p_k$, it holds that $\sum_j p_j \geq (m-1)(C_{LPT} - p_k) + C_{LPT}$. Therefore, $C_{LPT} \leq \frac{1}{m}\sum_j p_j + p_k\frac{m-1}{m}$. Clearly, $C^* \geq \frac{1}{m}\sum_j p_j$. Let α be the value such that $C^* = p_k(1+\alpha)$.

Case 1: $\alpha \geq \frac{1}{m}$. Thus, $C^* \geq p_k\frac{m+1}{m}$. In this case, $C_{LPT} \leq \frac{1}{m}\sum_j p_j + p_k\frac{m-1}{m} \leq C^*(1 + \frac{m-1}{m} \cdot \frac{m}{m+1}) = C^*(2 - \frac{2}{m+1})$.

Case 2: $\alpha < \frac{1}{m}$. In this case, $C^* < p_k\frac{m+1}{m}$, and the analysis is involved. Let ℓ_n be the number of nice jobs of length at least p_k (including J_k). It must hold that $\ell_n \leq m$, since otherwise at least two such jobs are assigned together in any schedule and in particular $C^* \geq 2p_k > (1+\alpha)p_k$. We know that $C^* = (1+\alpha)p_k$, therefore, in an optimal schedule, each of these ℓ_n jobs begins its processing at time at most αp_k. Moreover, this implies that in the optimal schedule, no bully job may start its processing later than αp_k. Let M_n be the set of machines that are assigned the ℓ_n long nice jobs in an optimal schedule. Each of these machines is assigned bully jobs of total length at most αp_k. Each of the other $m - \ell_n$ machines is assigned bully jobs of total length at most αp_k and maybe one additional bully job. We can therefore conclude that excluding the longest $m - \ell_n$ ones, the total length of bully jobs is at most $m\alpha p_k$.

So how does a double-LPT schedule of such an instance look like? First, the $m - \ell_n$ longest bully jobs are assigned, one to each machine, and then the remaining bully jobs are assigned. We know that ℓ_n machines remain empty after the $m - \ell_n$ long bully jobs are assigned, and that LPT assigns them bully jobs of total load of at most $m\alpha p_k$. If $\ell_n = 1$ then J_k is added to load at most $m\alpha p_k$ and the makespan of LPT is $p_k(1 + m\alpha)$. Given that $C^* = p_k(1+\alpha)$, the ratio is $\frac{1+m\alpha}{1+\alpha} < 2 - \frac{2}{m+1}$ for any $\alpha < 1/m$. If $\ell_n > 1$ then we use the fact that LPT guarantees that the gap in the load between any two of the ℓ_n machines is at most αp_k. Thus, the load on each of these ℓ_n machines after LPT assigns the bully jobs is at most $m\alpha p_k/\ell_n + \alpha p_k \leq p_k(m\alpha/2 + \alpha)$. When J_k joins a machine, the total load on it becomes at most $p_k(1 + m\alpha/2 + \alpha)$. The approximation ratio is therefore less than $(1 + m\alpha/2 + \alpha)p_k/(1+\alpha)p_k$. For any $\alpha < 1/m$, this ratio is less than 1.5.

As demonstrated in Fig. 2, the above analysis is tight for any $m \geq 2$. The tight bound is achieved for $\alpha = 1/m$. □

3.3 A PTAS for $P|K_{b,n}, s\text{-}prec|C_{max}$

Ms. Schedule was familiar with several PTASs for the minimum makespan problem [8, 13]. She was even working on implementing one with the little jobs in their Drama class, hoping to have a nice show for the end-of-year party. However, knowing that double-OPT may be far from being optimal, she understood that with selfish precedence-constraints a similar double-PTAS approach will not lead to approximation-ratio better than $2 - \frac{2}{m+1}$, independent of ε. "I must develop a new PTAS, in which the assignment of bully and nice jobs is coordinated", she thought.

Ms. Schedule was able to solve the problem by combining the dual-approximation scheme idea from the PTAS of Hochbaum and Shmoys for $P\|C_{max}$ [13], and the idea

of grouping small jobs, as introduced in the PTAS of Sahni for instances with a constant number of machines ($P_m \| C_{max}$) [20]. She explained her PTAS to the school's principal.

"As you surely know, the idea in the PTAS of Hochbaum and Shmoys [13] is to relate a schedule on m machine to a packing in m bins. I am going to adopt this idea. However, I need to define a new variant of the bin packing problem, in which items of two types are packed", she started her explanation. "I will denote this problem *two-type bin-packing* ($2T$-BP). The input for $2T$-BP is a collection of items whose sizes are in $[0, 1]$. Some of the items are of type-B, the others are of type-N. Formally, let $I = \{p_1 \dots p_z\}$ be the sizes in a set of z items, where $0 \leq p_j \leq 1$. As in regular BP, the goal is to pack all the items into bins $\{B_1, B_2, \dots, B_k\}$ such that the number of bins, k, is minimized and the packing is feasible, that is, for all i, $1 \leq i \leq k$, $\sum_{j \in B_i} p_j \leq 1$. In addition, the following condition should hold:

Condition C_{2T} Let β_i denote the number of type-B items packed in bin i. Then, there exists a $0 \leq \gamma \leq 1$ such that for all i, $1 \leq i \leq k$, the total size of the smallest $\beta_i - 1$ type-B items packed in bin i, is at most γ, and the total size of type-N items packed in bin i is at most $1 - \gamma$.

This condition implies that for some $0 \leq \gamma \leq 1$, for every bin i, it is possible to place the type-B items at the bottom of the bin, such that the total capacity allocated to all but the largest type-B item in the bin is at most γ. In other words, assume that you place the items one above the other starting from the bottom of the bin, then Condition C_{2T} implies that all type-B items are placed such that their low y-coordinate is at most γ, and all type-N items are placed such that their low y-coordinate is at least γ."

"The exact solutions of $P|K_{b,n}, s\text{-}prec|C_{max}$ and $2T$-BP relate in the following way", Ms. Schedule continued. "Given an instance for the minimum makespan problem, it is possible to schedule all the jobs on m machines with makespan C_{max} if and only if it is possible to pack in m bins all the items in a $2T$-BP instance, where the size of item j is p_j / C_{max}, and the packing-type of item j corresponds to the job-type of j; i.e., type-B (type-N) items corresponds to bully (nice) jobs".

In order to fully understand this relation, Ms. Schedule showed the principal Fig. 3, which demonstrates this relation for $m = 4$ and $C_{max} = d$. Type-B items and bully jobs are colored grey, type-N items and nice jobs are colored white. The principal noted to himself that the selfish precedence-constraints are kept if and only if Condition C_{2T} holds: the starting time of each of the nice jobs is at least γ, which is the latest starting time of a bully job—given by the highest level in which a type-B item is placed in some bin. Therefore, the packing fulfills Condition C_{2T} if and only if the schedule fulfills the selfish precedence-constraints.

"Next, let $OPT_{2TBP}(I)$ be the number of bins in an optimal solution for $2T$-BP, and let $C^*_{max}(I, m)$ be the minimal possible makespan of an instance I on m machines. Denote by $\frac{I}{d}$ the $2T$-BP input derived from I in which the item sizes are the job lengths divided by d. I already argued that:

$$OPT_{BP}\left(\frac{I}{d}\right) \leq m \quad \Leftrightarrow \quad C^*_{max}(I, m) \leq d.$$

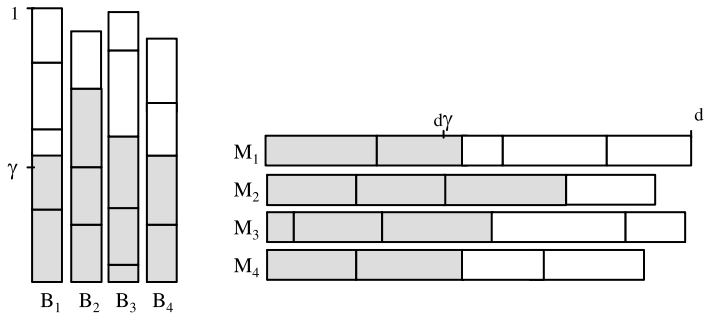

Fig. 3 A $2T$-BP packing (*left*) and the corresponding feasible schedule (*right*)

So all I need to do in order to complete the PTAS is to develop a *dual* approximation scheme for $2T$-BP. For an instance I, I seek a solution with at most $OPT_{2TBP}(I)$ bins, where each bin is filled to capacity at most $1+\varepsilon$. In other words, I relax the bin capacity constraint by a factor of $1+\varepsilon$. The proof of the following observation is identical to the corresponding proof in [13], you might know it". "Of course I know it", the principal broke in, "it is based on a binary search of the minimum makespan."

Observation 3.2 *If there exists a dual-PTAS for $2T$-BP, then there exists a PTAS for $P|K_{b,n}, s\text{-}prec|C_{max}$.*

"Hence, let me now describe the dual-PTAS for $2T$-BP", said Ms. Schedule. "Given ε, partition the items into *small* items—of size at most ε, and *big* items—of size larger than ε. Note that unlike the dual PTAS in [13], for regular BP, the small items cannot be added greedily to a packing of the big items, as in [13]. This is problematic because if big type-B items are packed till level γ, then, in order to fulfill Condition C_{2T}, any bin opened in the greedy stage can accommodate small type-N items of total size at most $1-\gamma$, and if there are many small type-N items and γ is large, then I might end up with many bins that are only filled to capacity $1-\gamma$, independent of ε. I therefore need a different approach for the small items."

Handling the Small Items "The first step of my scheme is to modify the instance I into a simplified instance I'. Given I, ε, let P_S^B, P_S^N denote the total size of small type-B and type-N items, respectively. The modified instance I' consists of all big items in I together with $\lfloor P_S^B/\varepsilon \rfloor$ type-B items and $\lfloor P_S^N/\varepsilon \rfloor$ type-N items of size ε. These items, which replace the small items, will be denoted *agent* items."

"The second step is to find a packing of I' in $OPT_{2TBP}(I)$ bins of size $1+2\varepsilon$. I will get to that soon. Finally, given this packing of I', I need to transform it into a packing of I." Ms. Schedule continued. "I will keep a pool of non-packed small type-B items and a pool of non-packed small type-N items. I will go over the bins one after the other. In the packing of I', in every bin, the type-B items are placed below the type-N items. I will first replace the big items of I' by their corresponding items in I. Next, if there are $k>0$ agent type-B items in the bin, I will add small type-B items from my pool—till the first time that the total size of the small type-B

items is at least $k\varepsilon$. Similarly, I will replace the type-N agent items in the bin, by original small type-N items of at least the same total size. I might run out of small items at some point but this is fine. The total overflow on each bin compared to its content in the packing of I' is at most 2ε (maximal size of two small items, one from each type)."

"There is one more point you need to consider," The principal broke in, "The feasibility of the packing might be hurt—if the largest type-B item in a bin is an agent item, then when this item is replaced by several small type-B items, later starts of type-B items are introduced—that is, the value of the corresponding γ is increased—which might hurt Condition C_{2T} of a feasible $2T$-BP". "I am glad you are following", said Ms. Schedule in delight. "To solve this problem, I can shift upward (by having a small empty space) the minimal level on which type-N items are placed. A shift of at most ε will do the work. I get that in every bin, the maximal increase in the minimal level on which type-N items can be placed, due to both an overflow of a small type-B item and a shift to guarantee the γ-condition, is at most ε. In addition, as explained above, an additional overflow of ε might be causes due to the replacement of the type-N agent items. Summing up,"

Corollary 3.3 *A two-type packing of I' in m bins of size z induces a feasible two-type packing of I in m bins of size at most $z + 2\varepsilon$.*

In addition, Ms. Schedule related in a similar way the optimal solutions for I' and I as follows:

Claim 3.4 *A two-type packing of I in m bins of size z induces a feasible two-type packing of I' in m bins of size at most $z + 2\varepsilon$.*

Proof Given a packing of I in bins of size z, a packing of I' can be derived by replacing, in each bin separately, the small items with agent jobs of size ε, with at least the same total size. Recall that the number of type-B (type-N) agent items of size ε in I' was determined to be $\lfloor P_S^B/(\varepsilon)\rfloor$, $(\lfloor P_S^N/(\varepsilon)\rfloor)$. By applying the above replacement, and performing shifting to starting locations of type-N items (if new, late, starting locations of type-B items were introduced, as explained in the paragraph above Corollary 3.3) we get a packing of I' into the same number of bins whose sizes might increase to $z + 2\varepsilon$. □

"Combining Corollary 3.3 and Claim 3.4, I get the following structure for my dual-PTAS. Stay tuned for step 3, which I still need to explain to you."

A dual-PTAS for $2T$-BP:

1. For a given ε', let $\varepsilon = \varepsilon'/4$.
2. Convert I into an instance I' with no items smaller than ε.
3. Pack I' in $OPT_{2TBP}(I')$ bins of size $1 + 2\varepsilon$.
4. Convert the packing of I' into a packing of I in $OPT_{2TBP}(I)$ bins of size $1 + 4\varepsilon$.

A dual-PTAS for a $2T$-BP Instance with Big Items (Step 3 above) Let I' be an instance of $2T$-BP with all-big items. Recall that, for a given $\varepsilon > 0$, our dual approximation-scheme needs to find a packing of all items using at most $OPT_{2TBP}(I')$ bins, such that the total size of the items packed in each bin is at most $1 + 2\varepsilon$.

Divide the range $[\varepsilon, 1]$ into intervals of size ε^2. This gives $S = \lceil \frac{1-\varepsilon}{\varepsilon^2} \rceil$ intervals. Denote by l_i the endpoints of the intervals. We now examine a packed bin. Since the minimal item size is ε, the bin can contain at most $\lfloor \frac{1}{\varepsilon} \rfloor$ items. Denote by X_i^B, X_i^N the number of type-B and type-N items in the bin, whose sizes are in the interval $(l_i, l_{i+1}]$. X_i^B, X_i^N are in the range $[0, \lfloor \frac{1}{\varepsilon} \rfloor)$. Let the vector $(X_1^B, \ldots, X_S^B, X_1^N, \ldots, X_S^N)$ denote the *configuration* of the bin.

Definition 3.1 A configuration is γ*-feasible* if

1. $\sum_{i=1}^{S} (X_i^B + X_i^N) l_i \leq 1$.
2. If X_i^B is positive for at least one index $1 \leq i \leq S$, then let j be the maximal index for which $X_j^B > 0$. Then $\sum_{i=1}^{j-1} (X_i^B) l_i + (X_j^B - 1) l_j \leq \gamma$.
3. $\sum_{i=1}^{S} X_i^N l_i \leq 1 - \gamma$.

For any bin *Bin* whose packing forms a γ-feasible configuration, the first condition in Definition 3.1 implies that the total size of the items in the bin is bounded by

$$\sum_{j \in Bin} p_j \leq \sum_{i=1}^{S} (X_i^B + X_i^N) l_{i+1} \leq \sum_{i=1}^{S} (X_i^B + X_i^N)(l_i + \varepsilon^2) \leq 1$$

$$+ \varepsilon^2 \sum_{i=1}^{S} (X_i^B + X_i^N) \leq 1 + \varepsilon^2 \cdot \frac{1}{\varepsilon} \leq 1 + \varepsilon.$$

Similarly, the second and third conditions in Definition 3.1 imply that the total size of all but the largest type-B items packed in the bin, is at most $\gamma + \varepsilon$, and the total size of type-N items packed in the bin is at most $1 - \gamma + \varepsilon$.

Therefore, it is sufficient to solve the instance with all item sizes rounded down to sizes in $\{l_1, \ldots, l_S\}$. By definition of $2T$-BP, in such a solution there exists a $\gamma \in [0, 1]$ such that all bins' configurations are γ-feasible. Given a solution of $2T$-BP for the rounded instance, by replacing each rounded item with the item originating it, we get a feasible solution for I' in bins of size $1 + 2\varepsilon$. One ε is added to the bottom of the bins—where type-B items are located, and one ε is added to the top of the bin—where type-N items are located.

"We are getting to the end", Ms. Schedule continued, feeling that the principal is losing his patient. "All I need in order to complete the dual-PTAS is a dynamic-programming algorithm that solves $2T$-BP exactly for the rounded instance. Let b_i, (n_i) be the number of type-B (type-N) items in I' whose sizes are in the interval $(l_i, l_{i+1}]$.

Note that all item sizes in the rounded instance are multiples of ε^2, thus, the total size of any subset of items is a multiple of ε^2. This implies that any feasible solution

for $2T$-BP fulfills Condition C_{2T} for $\gamma = k\varepsilon^2$, for some integer $0 \le k \le \lfloor 1/\varepsilon^2 \rfloor$. Let $BINS_\gamma(b_1, b_2, \ldots, b_S, n_1, n_2, \ldots, n_S)$ be the minimal number of bins required to pack, for all $1 \le i \le S$, b_i items of type-B and n_i items of type-N having size l_i, in γ-feasible configurations. Let $\mathcal{C}_\gamma$ denote the set of all γ-feasible configurations. Observe that, by a standard dynamic-programming recursion,

$$\begin{aligned} &BINS_\gamma(b_1, b_2, \ldots, b_S, n_1, n_2, \ldots, n_S) \\ &\quad = 1 + \min_{\mathcal{C}_\gamma} BINS_\gamma(b_1 - X_1^B, b_2 - X_2^B, \ldots, b_S - X_S^B, n_1 - X_1^N, n_2 \\ &\qquad - X_2^N, \ldots, n_S - X_S^N). \end{aligned}$$

I minimize over all possible vectors $(X_1^B, X_2^B, \ldots, X_s^B, X_1^N, X_2^N, \ldots, X_s^N)$ that correspond to a γ-feasible packing of the first bin (counted by the constant 1), and the best way to pack the remaining items (this is the recursive call). Thus, given γ, the dynamic-programming procedure builds a table of size $\max_i(n_i, b_i)^{2S}$, where the calculation of each entry requires $O(|C_\gamma|)$ time, which is a constant. By repeating the DP for all $\lfloor 1/\varepsilon^2 \rfloor$ possible values of γ, and selecting the minimal value of $BINS_\gamma(b_1, b_2, \ldots, b_S, n_1, n_2, \ldots, n_S)$, we get an optimal two-type packing of the rounded instance. This packing induces a packing of I' in $OPT_{2TBP}(I')$ bins of size $1 + 2\varepsilon$, as we need."

"Wonderful!", said the principal, "I must notify the PTA[2] that we have a PTA-Scheme for the minimum makespan".

4 Minimizing Total Flow-Time: $P|K_{b,n}, s\text{-}prec|\sum C_j$

Before the bully jobs arrived, one of Ms. Schedule favorite rules was SPT [21]. She used it when she wanted to minimize the total flow-time of the jobs. Ms. Schedule kept in her cupboard a collection of dolls that she called *dummy jobs* and used them from time to time in her calculations. Whenever Ms. Schedule wanted the jobs to use SPT rule, she first added to the gang some of her dummy jobs, so that the total number of (real and dummy) jobs divides m. The dummy jobs did not require any time in the slides (i.e., their sliding time was 0) so it was never clear to the little jobs why the dummies were needed, but Ms. Schedule explained them that they help her in her formal proofs. When applying SPT rule, the jobs sort themselves by their sliding time (from shortest to longest), and are assigned to slides one after the other, according to this order. In other words, the jobs are assigned to *heats*. The m fastest jobs form the 1st heat, the m next jobs form the 2nd heat and so on. The internal assignment of jobs from the same heat to slides doesn't matter to Ms. Schedule.

After the bully jobs joined the kindergarten it was clear to everyone that these jobs must be assigned to early heats, even if they are slow. For a single slide, it was not difficult to find a schedule achieving minimum total flow-time.

Theorem 4.1 *The problem* $1|K_{b,n}, s\text{-}prec|\sum C_j$ *is polynomially solvable.*

[2]Parent-Teacher Association.

Proof An optimal schedule is achieved by double-SPT. That is, assign first the bully jobs according to SPT rule, and then assign the nice jobs by SPT rule. Since any feasible schedule consists of a schedule of the bullies followed by a schedule of the nice job, even the little jobs were able to verify (using exchange arguments, were only internal-set exchanges are possible) that this is an optimal schedule. □

On the other hand, for more than one slide. Ms. Schedule couldn't come up with an efficient assignment rule. She told the principal that this is one of the problems she will never be able to solve. The principal couldn't accept it. "I know you are having a difficult quarter with these bullies, but you should try harder. I suggest to simply extend the assignment rule for a single slide", he told Ms. Schedule, "this double-SPT algorithm should work for every number of machines. Let me show you the following for instances with a *single* nice job. As you will see, these selfish precedence-constraints are totally different from the regular ones".

Theorem 4.2 *$P2|K_{b,1}, prec|\sum C_j$ is NP-hard, but $P|K_{b,1}, s\text{-}prec|\sum C_j$ is poly-solvable.*

Proof "For the hardness of $P2|K_{b,1}, prec|\sum C_j$" said the principal, "I will use a reduction from the bi-criteria problem $P2\|F_h(C_{max}/\sum C_j)$. In this problem, it is desired to minimize the total flow-time as the primary objective and minimize the makespan as the secondary objective. This problem is known to be NP-hard [2]. Since the single nice job can only start its execution after all preceding jobs complete their execution, it is easy to see that this bi-criteria problem can be reduced to our problem."

"And now, let me show you my optimal algorithm for $P|K_{b,1}, s\text{-}prec|\sum C_j$", the principal continued.

$\mathcal{A}_{prin}$: The Principal's Optimal Algorithm for $P|K_{b,1}, s\text{-}prec|\sum C_j$

1. Add dummy bullies to have zm bully jobs for some integer z.
2. Assign the bullies in SPT order. For $k = 1 \ldots z$, The jobs $(k-1)m+1, \ldots, km$ form the k-th heat. Each machine is getting one job from each heat. The shortest bully in every heat (i.e., jobs $1, m+1, \ldots, (z-1)m+1$) is assigned to M_1.
3. Assign the nice job to M_1.

"To complete the proof I will show you the following claim", the principal concluded.

Claim 4.3 *$\mathcal{A}_{prin}$ produces a feasible assignment that achieves minimum total flow-time.*

□

Proof Let C_1^B denote the completion time of the bullies assigned to M_1. According to the algorithm, the nice job starts its execution at time C_1^B. The schedule is feasible since the last bully job starts its execution not later than C_1^B, as otherwise, by migrat-

ing to M_1 the total completion time of the bullies can be reduced—contradicting the optimality of SPT for the total flow time of the bullies.

Any schedule generated by $\mathcal{A}_{prin}$ fulfills the following properties:

1. The shortest job from every heat is assigned to machine M_1. A single job from every heat is assigned to every other machine.
2. The nice job is assigned as last to machine M_1.

To prove the optimality of $\mathcal{A}_{prin}$, we show that any schedule fulfilling these properties is optimal. More specifically, a schedule that does not fulfill the above properties is either not optimal, or can be can be changed (vie job migrations and swaps) into a schedule fulfilling the properties without hurting its total flow time. Let s' be a schedule in which one of the above properties does not hold. Assume that the nice job is assigned to M_1 in s'. If property 1 holds but not the second, then by simple exchange argument it is possible to see that the optimal schedule is achieved when the shortest job from each heat is on M_1: inter-heat exchanges do not affect the total flow-time of bully jobs and can enable an earlier start time of the nice job.

In order to analyze the case in which property 1 does not hold, we need the following observation.

Observation 4.4 *Let $n = zm$ and $b = 1$. Any optimal schedule can be transformed into an optimal schedule in which the number of bully jobs on any machine is exactly z.*

Proof Let s be a schedule such that some machine M^+ is assigned at least $z+1$ bully jobs. Since the total number of bully jobs is zm, there must be a machine M^- with at most $z-1$ bully jobs. Let J_1 be the first job on M^+. Consider the schedule in which J_1 is moved to be first on M^-. The flow time of all jobs remaining on M^+, that is, at least z jobs, is reduced by p_1. The flow time of all jobs on M^-, that is, at most $z-1$ jobs, is increased by p_1. The total flow time of the bully jobs is therefore reduced by at least p_1. Consider the nice job. If it is the last job on M^+, then it might not benefit (at all or partially) from the migration of J_1, because its start time might be limited due to bullies on other machines. If it is not on M^+ then its start time might be delayed by at most p_1. Thus, the flow time of the nice job might be increased by at most p_1. All together, the total flow time in the resulting schedule is not worse than s.

By repeating such migrations as long as some machine is assigned more than z bully jobs we get a schedule obeying the above property, having a lower or not higher total flow time. □

"With this Observation, I can consider the case in which Property 1 does not hold for s'", said the principal. W.l.o.g, we can assume that there are exactly z bullies on every machine in s'—otherwise, s' is not optimal. Let j be the smallest index such that the jobs from heat j do not split among the machines. Specifically, there is at least one machine M_b with no job from heat j and there is at least one machine M_a with at least two jobs from heat j. If the jobs on M_a are not arranged in SPT order, then s' can be improved by inter-machine exchanges (as in the proof of SPT). Thus, we assume that two jobs from heat j are located in positions j and $j+1$ on M_a.

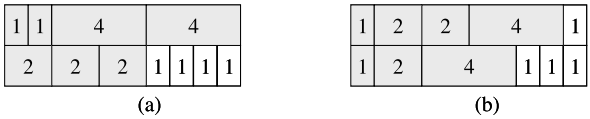

Fig. 4 (**a**) An optimal schedule. The bully jobs (in *grey*) are not scheduled according to SPT, $\sum C_j = 65$. (**b**) The best schedule under the constraint that bully jobs obey SPT, $\sum C_j = 66$

Consider a schedule in which the second of these jobs, J_a, exchanges location with the job J_b in position j on M_b. Since J_b belongs to heat at least $j+1$, it holds that $\Delta = p_b - p_a \geq 0$. By Observation 4.4, there are $z-j-1$ bully jobs after J_a on M_a, and $z-j$ bully jobs after J_b on M_b. Thus, exchanging J_a and J_b reduces the total flow-time of the bully jobs by Δ. If the nice job is assigned to M_a, then its starting time is delayed by Δ and the total change in the flow-time of all jobs might be 0. Also, the schedule is feasible since the starting time of the nice job is delayed. If the nice job is not assigned to M_a then its starting time might need to be delayed by at most Δ due to the delayed starting time of the last bully job on M_a. In this case the total flow-time is reduced by some value in $[0, \Delta]$.

We conclude that if s' does not fulfill property 1 then it is possible to swap pairs of jobs such that the resulting schedule is feasible, it has a reduced (or not increased) total flow time, and it fulfills property 1. □

"Now that we have a proof for a single nice job" said the principal, "we only need to extend it by induction for any number of nice jobs". Ms. Schedule was not impressed. She drew Fig. 4 on the whiteboard in the principal's office and said: "For more than a single nice job, your algorithm is not optimal". The principal looked at her doubtingly, but she continued, "as you can see, the total flow-time of the bully jobs is not necessarily minimal in an optimal schedule. Interestingly, while for a single nice job there is a distinction between regular and selfish precedence-constraints, for many nice jobs, the problem is NP-hard in both settings."

Theorem 4.5 *The problem* $P2|K_{b,n}, s\text{-}prec|\sum C_j$ *is NP-hard.*

Proof Ms. Schedule was using a reduction from the bi-criteria problem $P2\|F_h(C_{max}/\sum C_j)$. "As you claimed ten minutes ago", she told the principal, "this problem is known to be NP-hard [2]. It clearly remains NP-hard if the number of jobs and their total length are even integers. Consider an instance $\mathcal{I}$ of $2k$ jobs having lengths $a_1 \leq a_2 \leq \cdots \leq a_{2k}$, such that $\sum_i a_i = 2\gamma$". "For this instance", the principal broke in, "the minimum total flow time is

$$T = k(a_1 + a_2) + (k-1)(a_3 + a_4) + \cdots + (a_{2k-1} + a_{2k}),$$

and it is obtained by SPT". "You are right", Ms. Schedule nodded. "However, there are many ways to achieve this value. The hardness of $P2\|F_h(C_{max}/\sum C_j)$ tells us that in particular, it is NP-hard to decide if among these optimal schedules there is a balanced one—with makespan γ."

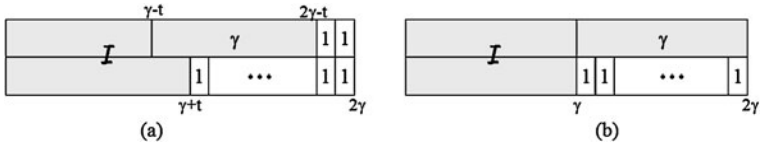

Fig. 5 (**a**) A general and (**b**) an optimal solution for the reduced instance

"Given $\mathcal{I}$, I will build the following input for $P2|K_{b,n}, s\text{-}prec|\sum C_j$: There are $b = 2k + 1$ bully jobs whose lengths are $a_1, \ldots, a_{2k}$ and γ. In addition, there are $n = \gamma$ nice jobs of length 1". The following claim completes my hardness proof.

Claim 4.6 *The instance $\mathcal{I}$ has a schedule with total flow-time T and makespan γ, if and only if the solution for $P2|K_{b,n}, s\text{-}prec|\sum C_j$ has value $T + \frac{3}{2}\gamma^2 + \frac{5}{2}\gamma$.* □

Proof Consider an optimal schedule for $P2|K_{b,n}, s\text{-}prec|\sum C_j$. The bully jobs on each machine are processed in SPT order (otherwise, use exchanges to improve the schedule), therefore, the bully job of length γ is the last bully job on some machine. Assume that the two machines complete processing the jobs originated from $\mathcal{I}$ at time $\gamma - t$ and $\gamma + t$ for some $t \geq 0$ (see Fig. 5(a)). The long bully job is scheduled in time interval $[\gamma - t, 2\gamma - t]$, else the first machine is idle for a while after it completes the jobs of $\mathcal{I}$ and the schedule cannot be optimal (it can be improved by switching the content of the machines starting at times $\gamma - t$ and $\gamma + t$). Also, $\gamma - t$ nice jobs are processed one after the other starting at time $\gamma + t$ on the second machine and t nice jobs are processed one after the other, after the long bully jobs, starting at time $2\gamma - t$ on the first machine. Otherwise, again, the schedule cannot be optimal (it can be improved by balancing the completion times of the machines). Let $C_{\mathcal{I}}$ denote the total flow-time of the bully jobs originated from $\mathcal{I}$ in the optimal schedule. The total flow-time is therefore $C_{\mathcal{I}} + (2\gamma - t) + [(\gamma + t + 1) + (\gamma + t + 2) + \ldots + (2\gamma)] + [(2\gamma - t + 1) + \cdots + (2\gamma)] = C_{\mathcal{I}} + t(\gamma - t - 1) + \frac{3}{2}\gamma^2 + \frac{5}{2}\gamma$.

Assume that $\mathcal{I}$ has a schedule s with total flow-time T and makespan γ. Consider the schedule in which the jobs of $\mathcal{I}$ are scheduled as in s, the long bully job is scheduled as last on one machine, and all nice jobs are schedule as last, one after the other, on the second machine (see Fig. 5(b)). This schedule fits the above description of an optimal schedule with $t = 0$. Given that $C_{\mathcal{I}} = T$, the long bully job completes at time 2γ and the nice jobs at times $(\gamma + 1), (\gamma + 2), \ldots, 2\gamma$, the total flow-time is $T + \frac{3}{2}\gamma^2 + \frac{5}{2}\gamma$.

For the other direction, assume that the optimal solution for $P2|K_{b,n}, s\text{-}prec| \times \sum C_j$ has value $T + \frac{3}{2}\gamma^2 + \frac{5}{2}\gamma$. That is, $C_{\mathcal{I}} + t(\gamma - t - 1) + \frac{3}{2}\gamma^2 + \frac{5}{2}\gamma = T + \frac{3}{2}\gamma^2 + \frac{5}{2}\gamma$, implying $C_{\mathcal{I}} + t(\gamma - t - 1) = T$. By the discussion Ms. Schedule had with the principal, we know that $C_{\mathcal{I}} \geq T$. Also, $t \leq \gamma$, thus, the only way to fulfill the above equation is by having $t = 0$ and $C_{\mathcal{I}} = T$. This induces a schedule of $\mathcal{I}$ having total flow-time T and makespan γ. □

5 Selfish Precedence-Constraints with Release Times

One significant difference between regular and selfish precedence-constraints is the influence of *release times*. If a job i is not around yet, other jobs can start their processing, even if i precedes them. However, if i is released while a job j such that $i \prec_s j$ is processed, then i pushes j a way and starts being processed right away (assuming that no job who precedes i was also released). Job j will have to restart its processing on some other time (independent of the partial processing it already experienced). This affect of release times is relevant for any precedence-constraints topology, not only for complete bipartite graphs.

Example Let $J = \{J_1, J_2, J_3\}$, $p_1 = p_2 = 2$, $p_3 = 1, r_1 = r_2 = 0, r_3 = 3$, and $J_3 \prec_s J_1$, $J_3 \prec_s J_2$. Then it is possible to process J_1 in time $[0, 2]$. Indeed $J_3 \prec_s J_1$, but J_3 is not around yet along the whole processing. J_2 may start its processing at time 2, but will be pushed away by J_3 upon its release at time 3. J_3 will be processed in time $[3, 4]$, and J_2 will be processed in time $[4, 6]$. Two processing units are required for J_2 even-though it was allocated one already.

Ms. Schedule noticed that when recess begins, the nice jobs were always out in the playground on time, while the bully jobs tended to arrive late to the playground.[3] She therefore decided to consider the case in which for every nice job $j \in N, r_j = 0$, while bully jobs have arbitrary release times. She denoted this type of instance by $r_j(B)$. Recall that upon an arrival of a bully job, he must start sliding right away (unless there are other bullies sliding, because, as we already know, bully jobs respect each other). Ms. Schedule decided to consider the minimum makespan problem for this setting.

5.1 Hardness Proof for a Very Simple Instance

It is known that $1|prec, r_j|C_{max}$ is solvable in polynomial time for any precedence-constraints graph [15]. This is not the case with selfish precedence-constraints: The problem is NP-hard already for $K_{b,n}$. In fact, already for the special case of $K_{1,n}$, which is an out-tree of depth 1, and when all nice jobs are available at time $t = 0$.

Theorem 5.1 *The problem* $1|K_{1,n}, s\text{-}prec, r_j(B)|C_{max}$ *is NP-hard*

Proof Ms. Schedule used a reduction from the subset sum problem. Let $A = \{a_1, a_2, \ldots, a_n\}$ be a set of items and let k be the target subset sum. It is NP-hard to decide whether A has a subset A' such that $\sum_{j \in A'} a_j = k$ [10]. Given A, k, construct the following instance of $1|K_{1,n}, s\text{-}prec, r_j(B)|C_{max}$. There are n nice jobs, $N = \{J_1, \ldots, J_n\}$. For all $1 \le j \le n$, $p_j = a_j$ and $r_j = 0$. The single bully job, J_{n+1}, has length $p_{n+1} = 1$, and is released at time $r_{n+1} = k$. Thus, $J_{n+1} \prec_s J_j$ for all $1 \le j \le n$ and these are the only precedence-constraints.

It is easy to verify that there exists a schedule for which $C_{max} = 1 + \sum_{j=1}^{n} a_j$ if and only if there exists a subset $A' \subseteq A$ such that $\sum_{j \in A'} a_j = k$. □

[3] Because they were busy pushing and calling names everybody on their way.

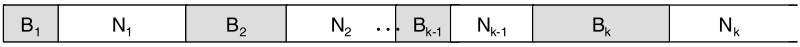

Fig. 6 The structure of any feasible schedule

5.2 A PTAS for $1|K_{b,n}, s\text{-}prec, r_j(B)|C_{max}$

Ms. Schedule decided to develop a PTAS for a single slide for the problem she is facing. Her first observation was that any feasible schedule of this type alternates between sliding time of bullies and nice jobs (see Fig. 6). Formally, the schedule consists of alternating B-intervals and N-intervals. A B-interval begins whenever a bully job arrives and no other bully is sliding, and continues as long as some bully job is around. The N-intervals are simply the complement of the B-intervals. During N-intervals, nice jobs may slide. In particular, during the last N-interval (after all bullies are done) nice jobs who are still in line can slide. The finish time of this last N-interval, N_k, for some $k \leq b$, determines the makespan of the whole schedule. If all nice jobs completed their processing before B_k then N_k is empty and the schedule is optimal. Given the release times and the sliding times of the bullies, the partition of time into B- and N-intervals can be done in a straightforward way—by assigning the bully jobs greedily one after the other whenever they are available.

"Given the partition into B- and N-intervals", thought MS. Schedule, "my goal is to utilize the first $k-1$ N-intervals in the best possible way. In fact, I need to *pack* the nice jobs into the first $k-1$ N-intervals, leaving as few idle time of the slide as possible. The slide might be idle towards the end of an N-interval, when no nice job can complete sliding before a bully shows up. Given $\varepsilon > 0$, my PTAS consists of the following steps:

1. Assign the bully jobs greedily. This determines the B- and N-intervals. Let k be the number of B-intervals.
2. Build the following instance for the multiple-knapsack problem:
   - $k-1$ knapsacks of sizes $|N_1|, |N_2|, \ldots, |N_{k-1}|$.
   - n items, where item j has size and profit p_j (sliding time of nice job j).
3. Run a PTAS for the resulting multiple-knapsack problem [3], with ε as a parameter.
4. Assign the nice jobs to the first $k-1$ N-intervals as induced by the PTAS. That is, jobs that are packed into a knapsack of size $|N_i|$ will be scheduled during N_i. Assign the remaining nice jobs, which were not packed by the PTAS, to N_k with no intended idle.

Let C_{ALG} denote the makespan of the schedule produced by the PTAS. Let C^* denote the optimal minimum makespan.

Claim 5.2 *Let $\varepsilon > 0$ be the PTAS parameter, then $C_{ALG} \leq (1+\varepsilon)C^*$.*

Proof Let C_B denote the completion time of the last bully job. That is, C_B is the finish time of interval B_k. If the makespan of the PTAS is determined by a bully job

(i.e., all nice jobs were packed into the first $k-1$ N-intervals and N_k is empty), then $C_{ALG} = C^* = C_B$ and the PTAS is optimal.

Otherwise, let $P(N)$ denote the total length of nice jobs in the instance. Let S^*, S denote the total length of nice jobs assigned to $|N_1|, |N_2|, \ldots, |N_{k-1}|$ in an optimal schedule and by the multiple-knapsack PTAS respectively. Then $C^* = C_B + (P(N) - S^*)$ and $C_{ALG} = C_B + (P(N) - S)$. This implies that minimizing the makespan is equivalent to maximizing the total length of nice jobs packed before B_k. This is exactly the objective of the multiple knapsack problem, where the profit from packing an item equals to its size. Since S is determined using a PTAS for the multiple knapsack problem [3], we have $S \geq (1-\varepsilon)S^*$. The implied approximation ratio of the PTAS for $1|K_{b,n}, s\text{-}prec, r_j(B)|C_{max}$ is

$$\frac{C_{ALG}}{C^*} = \frac{C_B + (P(N) - S)}{C_B + (P(N) - S^*)} \leq \frac{C_B + (P(N) - (1-\varepsilon)S^*)}{C_B + (P(N) - S^*)}$$
$$= 1 + \frac{\varepsilon S^*}{C_B + (P(N) - S^*)} \leq 1 + \varepsilon.$$

The last inequality follows from the fact that $S^* \leq C_B$ and $S^* \leq P(N)$. □

6 Selfish Precedence-Constraints of Unit-Length Jobs

Summer arrived. The jobs prepared a wonderful end-of-year show. The parents watched proudly how their jobs were simulating complex heuristics. No eye remained dry when the performance concluded with a breathtaking execution of a PTAS for the minimum makespan problem. At the end of the show they all stood and saluted the jobs and Ms. Schedule for their efforts.

Ms. Schedule decided to devote the summer vacation to extending her research on selfish precedence-constraints. During the school year, she only had time to consider the complete bipartite-graph case, and she was looking forward for the summer, when she will be able to consider more topologies of the precedence graph.

That evening, she wrote in her notebook: The good thing about bully jobs is that they do not avoid others be processed simultaneously with them. Among all, it means that the scheduler is more flexible. If for example we have two jobs and two machines, they can be processed simultaneously even if one of them is bully. With regular precedence-constraints, many problems are known to be NP-hard even if jobs have unit-length or if the precedence-constraints have limited topologies. For example, $P|p_i = 1, prec|C_{max}$ is NP-hard [23], as well as $P|p_i = 1, prec|\sum C_j$ [17]. These hardness results are not valid for selfish precedence-constraints. Formally,

Theorem 6.1 *The problems $P|p_i = 1,\ s\text{-}prec|C_{max}$ and $P|p_i = 1,\ s\text{-}prec|\sum C_j$ are polynomially solvable.*

Proof Let n, m be the number of jobs and machines, respectively. Consider any *topological sort* of the selfish precedence-constraints graph. Since the graph is induced by a partial order relation, such a sort always exist. An optimal schedule simply assigns

the first m jobs in the first heat, that is, schedule them in time $[0, 1]$. The next heat consists of the next m jobs in the topological sort, and so on. The makespan of this schedule is $\lceil n/m \rceil$, which is clearly optimal. This schedule is also optimal with respect to total flow-time, as it is a possible output of algorithm SPT on the same input without the selfish precedence-constraints. The schedule is feasible: if $i \prec_s j$ then i appears before j in the topological sort. Therefore, i is not assigned to a later heat than j. They may be assigned to the same heat, which is acceptable by job i. □

7 Summary and Discussion

A new school year was about to begin. The jobs had wonderful time in the summer and were very excited to return to 1st-grade at Graham school. Ms. Schedule summarized her results for the 1st-grade teacher, Ms. Worst-case, who was full of concerns towards getting the bully jobs to her room.

"I focused on selfish precedence-constraints given by a complete bipartite graph", Ms. Schedule started, "essentially, this models the bully-equilibrium problem we have at school. I first analyzed the price of bullying for the two objectives I found most important: minimum makespan and total-flow time. Next, I analyzed the well-known heuristics List-Scheduling and LPT, and I developed a PTAS for the minimum makespan problem. I then considered the problem of minimizing the total flow-time. I have a hardness proof for instances with many nice jobs and an optimal algorithm for instances with a single nice job. I suggest that you consult with the principal regarding this problem. He is not as dumb as he seems.

If the bully jobs keep being late also in 1st grade, you can use my PTAS for minimizing the makespan when bullies have release times. Finally, while for regular precedence-constraints, many problems are NP-hard already with unit-length jobs and very restricted topologies of the precedence graph, I showed that with selfish precedence-constraints, and any precedence graph, minimizing both the makespan and the total flow-time can be solved in linear-time.

I trust you to consider the following open problems during the next school year:"

- The only objectives I considered are minimum makespan and total flow-time. It would be very interesting to consider instances in which jobs have due dates, and the corresponding objectives of minimizing total or maximal tardiness and lateness. For regular precedence-constraints, these problems are known to be NP-hard already for unit-length jobs and restricted topologies of the precedence-constraints graph [17, 18].
- As I've just told you, the hardness of minimizing the total flow depends on the number of nice job. Can you find the precise value of n for which the problem becomes NP-hard? This value might be a constant or a function of m, b, or the sliding times. Also, as the problem is closely related to the bi-criteria problem $P2\|F_h(C_{max}/\sum C_j)$, it is desirable to check if heuristics suggested for it (e.g., in [6, 7]) are suitable also for our setting.
- It would be nice to extend my PTAS for $1|K_{b,n}, s\text{-}prec, r_j(B)|C_{max}$ for parallel slides. Note that for this setting, a late-arriving bully pushes a way only a single nice job (of his choice, or not, depending on your authorization).

- As is the case with other scheduling problems, it would be nice to extend the results to uniformly related or unrelated machines, and to consider additional precedence-constraints graphs, such as chains and in/out-trees.
- Another natural generalization for the total flow-time objective, is when jobs have weights. For regular precedence-constraints, the problem $1|prec|\sum w_j C_j$ is known to be NP-hard, and several approximation algorithms are known [4, 16].

A more general open problem is the following: Given are jobs and a precedence-constraints graph. The edges of the precedence-constraints graph are weighted, the weight of an edge (i, j) specifies the minimal gap between the starting times of i and j. In regular precedence-constraints $w(i, j) = p_i$. In other words, if i precedes j then j can start being processed at least $w(i, j)$ time units after i *starts* being processed. "I believe", said Ms. Schedule,"that this problem reflects natural scenarios arising in real-world applications such as production systems. The most relevant problem I found is scheduling with precedence delays [9, 19], which refers to the case where $w(i, j) \geq p_i$. Some experimental results for solving the problem via IP and branch and bound are considered in [22]. In my work with the bully jobs, I considered the case $w(i, j) = 0$. I recently found out that this case was studied in [14] for instances in which the precedence-constraints graph consists of chains. What happens when $0 < w(i, j) < p_i$, i.e., some overlap is allowed?".

References

1. Brucker, P., Knust, S.: Complexity results for scheduling problems. http://www.mathematik.uni-osnabrueck.de/research/OR/class/
2. Bruno, J.L., Coffman, E.G., Sethi, R.: Algorithms for minimizing mean flow-time. In: IFIPS Congress, vol. 74, pp. 504–510 (1974)
3. Chekuri, C., Khanna, S.: A PTAS for the multiple knapsack problem. In: Proc. of the 11th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 213–222 (2000)
4. Chekuri, C., Motwani, R.: Precedence constrained scheduling to minimize sum of weighted completion times on a single machine. Discrete Appl. Math. **98**(1–2), 29–38 (1999)
5. Coffman, E.G., Sethi, R.: A generalized bound on LPT sequencing. In: Proc. of the Joint International Conference on Measurements and Modeling of Computer Systems, SIGMETRICS (1976)
6. Coffman, E.G., Sethi, R.: Algorithms minimizing mean flow-time: schedule length properties. Acta Inform. **6**, 1–14 (1976)
7. Eck, B.T., Pinedo, M.: On the minimization of the makespan subject to flowtime optimality. Oper. Res. **41**, 797–800 (1993)
8. Epstein, L., Sgall, J.: Approximation schemes for scheduling on uniformly related and identical parallel machines. Algorithmica **39**(1), 43–57 (2004)
9. Finta, L., Liu, Z.: Single machine scheduling subject to precedence delays. Discrete Appl. Math. **70**(3), 247–266 (1996)
10. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman, San Francisco (1979)
11. Graham, R.L.: Bounds for certain multiprocessing anomalies. Bell Syst. Tech. J. **45**, 1563–1581 (1966)
12. Graham, R.L.: Bounds on multiprocessing timing anomalies. SIAM J. Appl. Math. **17**, 263–269 (1969)
13. Hochbaum, D.S., Shmoys, D.B.: Using dual approximation algorithms for scheduling problems: Practical and theoretical results. J. ACM **34**(1), 144–162 (1987)
14. Kim, E., Posner, M.E.: Parallel machine scheduling with S-precedence constraints. IIE Trans. **42**, 525–537 (2010)

15. Lawler, E.L.: Optimal sequencing of a single machine subject to precedence constraints. Manag. Sci. **19**, 544–546 (1973)
16. Lawler, E.L.: Sequencing jobs to minimize total weighted completion time subject to precedence constraints. Ann. Discrete Math. **2**, 75–90 (1978)
17. Lenstra, J.K., Rinnooy Kan, A.H.G.: Complexity of scheduling under precedence constraints. Oper. Res. **26**(1), 22–35 (1978)
18. Leung, J.Y.-T., Young, G.H.: Minimizing total tardiness on a single machine with precedence constraints. ORSA J. Comput. **2**(4), 346–352 (1990)
19. Queyranne, M., Schulz, A.S.: Approximation bounds for a general class of precedence constrained parallel machine scheduling problems. SIAM J. Comput. **35**(5), 1241–1253 (2006)
20. Sahni, S.: Algorithms for scheduling independent tasks. J. ACM **23**, 555–565 (1976)
21. Smith, W.E.: Various optimizers for single-stage production. Nav. Res. Logist. Q. **3**, 59–66 (1956)
22. Sucha, P., Hanzalek, Z.: Scheduling of tasks with precedence delays and relative deadlines—framework for time-optimal dynamic reconfiguration of FPGAs. In: The 20th Int. Conf. on Parallel and Distributed Processing (IPDPS)(2006)
23. Ullman, J.D.: NP-complete scheduling problems. J. Comput. Syst. Sci. **10**, 384–393 (1975)