Scheduling with Bully Selfish Jobs

Tami Tamir*

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.

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 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 [13]. 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 $O(nlogn)$) 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.

*School of Computer Science, The Interdisciplinary Center, Herzliya, Israel. *E-mail*: tami@idc.ac.il.

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.

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, even if they move to a different 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."

[1]Bully jobs tend to have bully parents.

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 taught "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 releaded they push a following job away from the slide even if they already started climbing". Ms. Schedule decided to elaborate on that issue of release times later (see Section 5), and to set as a primal goal the analysis of the basic problems of minimum makespan (Section 3) and minimum total flow-time (Section 4) for the precedence constraint setting she has in her kindergarten. In this paper we tell her story and reveal her results. Ms. Schedule kindly provided us with her complete work, however, due to space constraints, some of the proofs are given in the Appendix.

2.2 Complete-Bipartite Selfish Precedence-Constraints

"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 schedule respects the $K_{b,n}$ constraints if and only if no nice job starts before a bully job. In fact, a schedule may respect the $K_{b,n}$ constraints, but not be a bully-equilibrium. This can happen if for example some machine is empty while another machine is loaded with more than a single bully job – in this case the precedence constraints hold but bullies will migrate to the empty machine. In general, if the gap between the completion times of the last bullies on different machines is larger than the length of the last bully on these machines, then this last bully will migrate. However, it is easy to see that for every reasonable objective function, in particular, all objective functions I'm about to consider, having a small gap between the completion times of the last bullies on different machines is a dominant property. Therefore, w.l.o.g., I would assume the following equivalence:"

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

The Price of Bullying: Let S denote the set of all schedules, not necessarily obeying the selfish precedence-constraints. For a schedule $s \in S$, let $g(s)$ be some measure of s. For example, $g(s) = \max_j C_j(s)$ is the makespan measure, and $g(s) = \sum_j C_j$ 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, $PoB = \min_{s\in\Phi(I)} g(s)/ \min_{s\in S} g(s)$.*

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

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

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. With regular precedence constraints, an optimal solution for $P|K_{b,n}, prec|C_{max}$ consists of a concatenation of optimal solutions for each type of jobs, but with selfish precedence-constraints, this approach might not lead even to a good approximation. To clarify this point better, Ms. Schedule drew Figure 1, and pointed out to herself that gluing independent optimal solutions to each type of jobs (denote this method double-OPT) can be far by a factor of at least $2 - \frac{2}{m+1}$ from an optimal solution. The main 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 a constant number of machines.

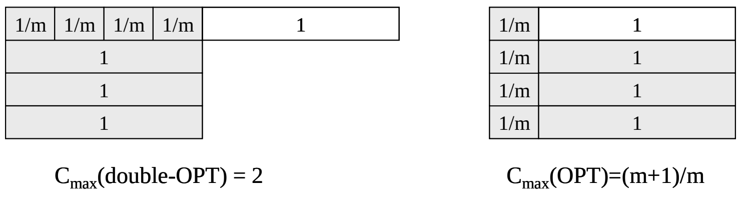

Figure 1: The approximation ratio of double-OPT and double-LPT is at least $2 - \frac{2}{m+1}$.

3.1 Adjusted 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 [12], therefore it produces a $(2 - \frac{1}{m})$-approximation to the makespan, even compared to the makespan with no selfish precedence-constraints.

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 job 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, as was detected by Ms. Schedule when considering double-OPT assignments, 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 $(\frac{4}{3} - \frac{1}{3m})$ [13] or $(1 + \frac{1}{k})$ where k is the number of jobs on the most loaded machine [5]). Note that for the instance considered in Figure 1, the double-OPT schedule is achieved also by double-LPT. As Ms. Schedule was able to show, 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 in the instance (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, $\frac{4}{3} - \frac{1}{3m} \leq 2 - \frac{2}{m+1}$, 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, $C^* < p_k\frac{m+1}{m}$, the analysis is involved, and given in the Appendix.

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

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

Ms. Schedule was familiar with several PTASs for the minimum makespan problem [15, 10]. 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 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 for a constant number of machines. She used the idea of grouping small jobs, as introduced in the PTAS of Sahni for $P_m||C_{max}$ [20]. Given an instance $I = B \cup N$, let P denote the total processing time of all jobs, and let p_{max} denote the longest processing time of a job (bully or nice). Let $C = \max(P/m, p_{max})$. Note that C is a lower bound on the minimum makespan (that is $OPT(I) \geq C$).

"The first step of my scheme is to modify the instance I into a simplified instance I'. Given I, ε, partition the jobs into small jobs – of length at most εC, and big jobs – of length larger than εC. Let P_S^B, P_S^N denote the total length of small bully and small nice jobs, respectively. The modified instance I' consists of all big jobs in I together with $\left\lfloor P_S^B/(\varepsilon C) \right\rfloor$ bully jobs and $\left\lfloor P_S^N/(\varepsilon C) \right\rfloor$ nice jobs of length εC. These jobs which replace the small jobs will be denoted *agent* jobs."

"The second step is to solve optimally the problem for I'. I can do it in polynomial time: all jobs in I' have length at least εC and their total length is at most P. Therefore, the number of jobs in I' is at most *the constant* $P/\varepsilon C \leq m/\varepsilon$. An optimal schedule of a constant number of jobs can be found by exhaustive search over all $O(m^{m/\varepsilon})$ possible schedules. For every partition of the jobs to the machines, assign on every machine the bully jobs before the nice ones, and check whether it induces a feasible schedule with respect to the selfish precedence-constraints. The feasible schedule with the lowest makespan is an optimal solution for I'. The time complexity is independent of n, but grows exponentially with ε – as I expect from my PTAS. Let $OPT(I')$ be an optimal schedule of I'."

"Finally, given an optimal schedule of I', I need to transform it into a schedule of I." thought Ms. Schedule. "This stage is the one in which my PTAS and its analysis are different from Sahni's. I will keep a pool of non-scheduled small bully jobs and a pool of non-scheduled nice jobs. I will go over the machines one after the other. In $OPT(I')$, in every machine, the bully jobs appear before the nice jobs, I will replace the big jobs of I' by their corresponding jobs in I. Also, if there are $k > 0$ agent bully jobs on the machine, I will add small bully jobs from my pool - till the first time that the total load of the bully small jobs is at least $k\varepsilon C$. Similarly, I will replace the nice agent jobs on the machine, with original small nice jobs of at least the same total length. I might run out of small jobs at some point but this is fine. The total overflow on each machine compared to its load in $OPT(I')$ is at most $2\varepsilon C$ (maximal length of two small jobs, one from each type)."

"There is one more point I need to handle," Ms. Schedule kept thinking, "The feasibility of the schedule might be hurt - if the latest starting time of a bully is determined by an agent bully job J', then by replacing J' by several small bullies, later starts of bully jobs are introduced - maybe later than starts of nice jobs. To solve this problem, I can swap locations of bully jobs: If there is a big bully job in the same machine as J', I can swap them without affecting the load on the machine. If all bully jobs on J'-th machine are agents, I can exchange a group of them with a big bully on another machine, this may cause an overflow - but I can make sure that in this case there will be no overflow due to small jobs on this machine. Some day I will write the full version of this proof with all the details. For now, I will just sum up:"

Corollary 3.2 *An optimal assignment of I' induces a feasible assignment of I whose makespan is at most $OPT(I') + 2\varepsilon C$.*

To complete the analysis Ms. Schedule related $OPT(I')$ to $OPT(I)$ as follows:

Claim 3.3 *$OPT(I') \leq (1+2\varepsilon)OPT(I)$.*

Back to the scheme, the optimal assignment of I' is converted into a feasible assignment of I with makespan at most $OPT(I')+2\varepsilon C$. By the above claim, this value is at most $(1+2\varepsilon)OPT(I)+2\varepsilon C \leq (1+4\varepsilon)OPT(I)$. The desired ratio of $(1+\varepsilon)$ is achieved by selecting $\varepsilon' = \varepsilon/4$, and running the scheme for ε'.

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 divided 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 it helps her in her calculations. When applying SPT rule, the jobs sorted themselves by their sliding time, and were assigned to *heats*. The m fastest jobs formed the 1st heat, the m next jobs formed the 2nd heat and so on. The internal assignment of jobs from the same heat to slides didn't matter to Ms. Schedule.

After the bully jobs arrived it was clear to everyone that these jobs must be assigned to early heats, even if they were 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.*

Proof: An optimal schedule is achieved by first assigning the bully jobs according to SPT rule, and then assigning the rest of the jobs by SPT rule. Since any feasible schedule consists of a schedule of B followed by a schedule of N, even the little jobs were able to verify (using exchange arguments, were exchanges are possible only inside each set) that this is an optimal schedule. ■

On the other hand, for more than one slide. Ms. Schedule couldn't came up with an efficient assignment rule. She told the principal that this was 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, "Just let the bullies arrange themselves by SPT rule, and then let the nice jobs join them greedily - also according to SPT order. Let me show you the following for instances with a *single* nice job. As you will see, these selfish precedence-constraints are totaly 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. "First, using your dummy jobs, we can assume that the number of bully jobs is zm for some integer z. Next, sort the bully jobs from shortest to longest and assign them to the machines greedily - using SPT rule. The jobs $(k-1)m+1, \ldots, km$ form the k-th heat. Each machine is getting in turn one job from each heat. In particular, and this is the important part, the shortest job from each heat goes to M_1. Finally, assign the nice job to M_1. To complete the proof I will show you the following claim", the principal concluded.

Claim 4.3 *The resulting assignment is feasible and achieves minimum total flow-time.* ■

"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 Figure 2 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."

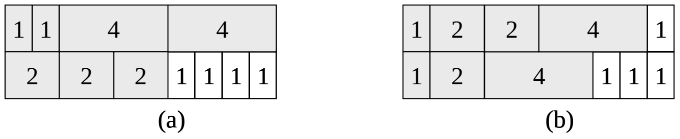

Figure 2: (a) An optimal schedule. The bully jobs (in grey) are not scheduled according to SPT, $\sum C_j = 65$. (b) The best schedule under the constraint that bully jobs obey SPT, $\sum C_j = 66$.

Theorem 4.4 *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 five minutes ago", she told the principal, "this problem is known to be NP-hard [2]. It remains NP-hard if the number of jobs and their total length are even integers. Consider an instance $\mathcal{I}$ of $2k$ jobs having lengthes $a_1 \leq a_2 \leq \ldots \leq a_{2k}$, such that $\sum_i a_i = 2B$". "For this instance", the principal broke in, "the minimum total flow time is

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

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 it is NP-hard to decide if among these optimal schedules there is a one with makespan B."

"Given $\mathcal{I}$, I will build the following input for $P2|K_{b,n}, s\text{-}prec|\sum C_j$: There are $2k+1$ bully jobs whose lengthes are $a_1, \ldots, a_{2k}$ and B. In addition, there are $n = B$ nice jobs of length 1". "It is easy to solve the problem for the bully jobs only", said the principal, "since you have an odd number of jobs, add one of your dummy jobs of length 0, and apply SPT rule. I can tell you that the resulting total flow-time will be

$$T' = (k+1)a_1 + k(a_2 + a_3) + (k-1)(a_4 + a_5) + \ldots + (a_{2k} + B)."$$

"Once again, you are right", said Ms. Schedule, "but the main issue here is that the optimal solution for the whole instance is not necessarily achieved by minimizing the total flow-time of the bully jobs. The more important question is whether there is an assignment of the bullies with a particular gap in the loads between the machines. The following claim completes my hardness proof.

Claim 4.5 *The instance $\mathcal{I}$ has a schedule with total flow-time T and makespan B if and only if the solution to $P2|K_{b,n}, s\text{-}prec|\sum C_j$ has value $T + \frac{3}{2}B^2 + \frac{5}{2}B$.* ■

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[2]. 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 [16]. 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 $\sum_{j \in A'} a_j = k$ [11]. 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 $r_j = 0$.

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

The single bully job, J_{n+1}, has length $p_{n+1} = 1$, and is released at time $r_{n+1} = k$. Thus, $n+1 \prec_s j$ for all $1 \leq j \leq n$ 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 $\sum_{j \in A'} a_j = k$. ■

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 Figure 3). 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 interval, N_k, for some $k \leq b$, determines the makespan of the whole schedule. 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.

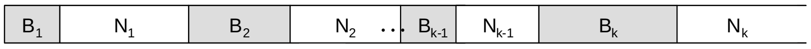

Figure 3: The structure of any feasible schedule

"Given the partition into B- and N-intervals", thought MS. Schedule, "my goal is to utilize the N-intervals in the best possible way. In fact, I need to *pack* the nice jobs into the 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

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

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 [22], as well as $P|p_i = 1, prec|\sum C_j$ [18]. 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 assign the first m jobs in the first heat, that is, schedule them in time $[0,1]$. The next heat will consist 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", she 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 [18, 19].

- 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, 9]) 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 $1|prec|\sum w_jC_j$ is known to be NP-hard, and several approxiamtion algorithms are known [17, 4].

References

[1] P. Brucker, S. Knust, Complexity results for scheduling problems *http://www.mathematik.uni-osnabrueck.de/research/OR/class/*

[2] J.L. Bruno, E.G. Coffman, and R. Sethi, Algorithms for minimizing mean flow-time, *IFIPS Congress* 74, 504-510, 1974.

[3] C. Chekuri and S. Khanna. A PTAS for the multiple knapsack problem. In *Proc. of the 11th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 213–222, 2000.

[4] C. Chekuri and R. Motwani, Precedence constrained scheduling to minimize sum of weighted completion times on a single machine, *Discrete Applied Mathematics*, vol.98(1-2):29-38, 1999.

[5] E.G. Coffman, and R. Sethi. A generalized bound on LPT sequencing. *Proc. of the Joint International Conference on Measurements and Modeling of Computer Systems, SIGMETRICS.* 1976.

[6] E.G. Coffman, and R. Sethi. Algorithms minimizing mean flow-time: schedule length properties. *Acta Informatica*6: 1-14, 1976.

[7] M. Dror, W. Kubiak, and P. Dell'Olmo. Strong-weak chain constrained scheduling. *Ricerca Operativa,* 27:35-49, 1998.

[8] J. Du, J.Y.-T. Leung, and G.H. Young. Scheduling chain-structured tasks to minimize makespan and mean flow-time. *Inform. and Comput.,* 92(2):219-236, 1991.

[9] B.T. Eck and M. Pinedo. On the minimization of the makespan subject to flowtime optimality. *Operations Research*, 41, 797-800. 1993

[10] L. Epstein, J. Sgall. Approximation schemes for scheduling on uniformly related and identical parallel machines *Algorithmica*, 39(1), 43-57, 2004.

[11] M.R. Garey and D.S. Johnson, *Computers and Intractability: A guide to the theory of NP-completeness*, W. H. Freeman and Company, San Francisco, 1979.

[12] R.L. Graham. Bounds for Certain Multiprocessing Anomalies. *Bell Systems Technical Journal*, 45:1563–1581, 1966.

[13] R.L. Graham. Bounds on Multiprocessing Timing Anomalies. *SIAM J. Appl. Math.*, 17:263–269, 1969.

[14] T.C. Hu. Parallel sequencing and assembly line problems. *Oper. Res.,* 9:841-848, 1961.

[15] D.S. Hochbaum and D.B. Shmoys. Using dual approximation algorithms for scheduling problems: Practical and theoretical results. *Journal of the ACM*, 34(1):144–162, 1987.

[16] E.L. Lawler. Optimal sequencing of a single machine subject to precedence constraints. *Management Sci.*, 19:544-546, 1973.

[17] E.L. Lawler. Sequencing jobs to minimize total weighted completion time subject to precedence constraints. *Ann. Discrete Math.,* 2:75-90, 1978.

[18] J.K. Lenstra and A.H.G. Rinnooy Kan. Complexity of scheduling under precedence constraints. *Oper. Res.,* 26(1):22-35, 1978.

[19] J.Y.-T. Leung and G.H. Young. Minimizing total tardiness on a single machine with precedence constraints. *ORSA J. Comput.,* 2(4):346-352, 1990.

[20] S. Sahni. Algorithms for scheduling independent tasks, *J. of the ACM*, 23, 555–565, 1976.

[21] W.E. Smith, Various optimizers for single-stage production. *Naval Research Logistics Quarterly.* vol. 3, pp. 5966, 1956.

[22] J.D. Ullman. NP-complete scheduling problems. *J. Comput. System Sci.*, 10:384-393, 1975.

A Missing Proofs

Proof of Theorem 2.2: 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). In addition, the long nice job is assigned in the time interval $[m-1, 2m-1]$ on one of the machines. The makespan of any bully-equilibrium is therefore $2m-1$. 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 [12] (see also Section 3.1). ■

Proof of Theorem 2.3: 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 $z+1 = (n+b)/m$. In addition, the PoB cannot be more than n/m: For every instance, for every given schedule, in particular the best bully-equilibrium, let J_{last} denote the set of jobs that are last on some machine. The total flow-time of the jobs in J_{last} equals $\sum_j p_j$. The total flow-time of any schedule is therefore at most $\frac{n+b}{|J_{last}|}\sum_j p_j$ and at least $\sum_j p_j$. Since $|J_{last}| = m$, the PoB is bounded by $(n+b)/m$. ■

Analysis of Case 2 in the Proof of Theorem 3.1: $\alpha < \frac{1}{m}$. 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 $m - \ell_n$ 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 ℓ_n long jobs are assigned, and LPT assigns them total load of at most $m\alpha p_k$. In $\ell_n = 1$ 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 then 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.

Proof of Claim 3.3: Given a schedule of I, in particular an optimal one, a schedule for I' can be derived by replacing, on each machine separately, the small jobs with jobs of length εC, with at least the same total size. Recall that the number of bully (nice) agent jobs of length εC in I' was determined to be $\left\lfloor P_S^B/(\varepsilon C)\right\rfloor$, $(\left\lfloor P_S^N/(\varepsilon C)\right\rfloor)$. By applying the above replacement, the last start time of a bully agent job on each machine is not later than the start time of the last small bully job on this machine in the assignment of I. Also, the first start time of a nice agent job is not earlier than the first start time of a small nice job in the assignment of I. Thus, the result of this replacement is a feasible schedule of I' whose makespan is at most $OPT(I) + 2\varepsilon C$. The statement of the claim holds, since $OPT(I) \geq C$. ■

Proof of Claim 4.3: Recall that there are zm bully jobs. Denote by $p_{h,j}$ the length of the j-th job in the h-th heat, for $1 \leq h \leq z, 1 \leq j \leq m$. The nice job starts its execution at time $t_n = \sum_{h=1}^{z} p_{h,1}$. The last bully job starts its execution at time $t_b = \sum_{h=1}^{z-1} p_{h,m}$. According to the sort, for all $2 \leq h \leq m$ it holds that $p_{h,1} \geq p_{h-1,m}$. Summing all these inequalities, we get $t_n - p_{1,1} \geq t_b$. Since $p_{1,1} \geq 0$ this implies $t_n \geq t_b$, thus the selfish precedence constraints are not violated and the schedule is feasible.

Let A be the assignment derived by the algorithm and assume towards contradiction that there is a better assignment, A'. Assume also w.l.o.g, that in A', the nice job is assigned to M_1. Using exchange argument, similar to the one in the optimality proof of SPT for $P||\sum C_j$ [21], it follows that in any optimal solution, and in particular A', the number of bully jobs assigned to each machine is exactly z. If there is a single job from every heat on each machine, 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.

The only case left to consider is when there are exactly z bully jobs on each machine, but not exactly one from every heat. Let j be the smallest index such that the jobs from heat j do not split among the machines. Specifically, there is a machine M' with no job from heat j and there is a machine M'' with at least two jobs from heat j. Since the jobs on each machine are arranged in SPT order (else A' is not optimal), the two jobs from heat j are located in positions j and $j+1$ on M''. Consider a schedule in which the second of these jobs, J_a, exchanges location with the job J_b in position j on M'. Since J_b belongs to heat at least $j+1$, it holds that $p_b \geq p_a$. As in the regular proof of SPT optimality, such an exchange reduces the total flow-time of bully jobs by $p_b - p_a$. If $M' = M_1$, that is, the nice job is assigned to M' then the total change in the flow-time of all jobs is 0. Also, the schedule is feasible since the starting time of the nice job is delayed. If $M'' = M_1$ then the total flow-time is reduced by $2(p_b - p_a)$ which is negative. However, the resulting schedule might not be feasible. If indeed, then M_1 can be idle till the last job on M' starts its execution. Since this job was delayed by $p_b - p_a$ we might need to delay the nice job by at most $p_b - p_a$ - compared to its start time in A'. The total change in the flow-time of all jobs in this case is 0.

We conclude that if A' is different from A, then it is possible to do a sequence of exchanges, each of which reduces or does not increase the total flow time, leaves the schedule feasible and results in the schedule A produces by the principal's algorithm. ■

Proof of Claim 4.5: In any optimal schedule for $P2|K_{b,n}, s\text{-}prec|\sum C_j$, the bully jobs on each machine are processed according to SPT rule (otherwise, use exchanges to improve the schedule), therefore, the bully job of length B is the last bully job on some machine. Assume that the two machines complete processing the jobs originated from $\mathcal{I}$ at time $B - t$ and $B + t$ for some $t \geq 0$ (see Figure 4(a)). The long bully job is scheduled in time interval $[B - t, 2B - t]$, else the first machine is idle after it completes the jobs of $\mathcal{I}$ and the schedule cannot be optimal. Also, $B - t$ nice jobs are processed one after the other starting at time $B + t$ on the second machine and t nice jobs are processed one after the other, after the long bully jobs, starting at time $2B - t$ on the second machine. Otherwise, again, the schedule cannot be optimal. 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 $C_{\mathcal{I}}+2B-t+(B+t+1)+(B+t+2)+\ldots+(2B)+(2B-t+1)+\ldots+2B = C_{\mathcal{I}}+t(B-t-1)+\frac{3}{2}B^2+\frac{5}{2}B$.

Assume that $\mathcal{I}$ has a schedule s with total flow-time T and makespan B. 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 Figure 4(b)). This schedule fits the above description of an optimal schedule with $t = 0$. Moreover, by the discussion Ms. Schedule had with the principal, we know that in this schedule, $C_{\mathcal{I}} = T$. The long bully job completes at time $2B$ and the nice jobs at times $(B + 1), (B + 2), \ldots, 2B$. The total flow-time is therefore $T + \frac{3}{2}B^2 + \frac{5}{2}B$.

Assume that every schedule of $\mathcal{I}$ with total flow-time T has makespan larger than B. If in the optimal schedule $t = 0$, then the jobs originated from $\mathcal{I}$ are not scheduled by SPT and (using the regular analysis of SPT), and $C_{\mathcal{I}} > T$. This implies that the total flow-time is more than $T + \frac{3}{2}B^2 + \frac{5}{2}B$.

Assume that in the optimal schedule $t > 0$. By the discussion Ms. Schedule had with the principal, we know that $C_{\mathcal{I}} + (2B - t) \geq T'$. Note that $T' - T = B + a_1 + a_3 + a_5 + \ldots + a_{2n-1} > B$. Therefore, if $t > 0$ then

$$\begin{aligned}\sum C_j &= C_{\mathcal{I}} + t(B-t-1) + \frac{3}{2}B^2 + \frac{5}{2}B \geq T' - (2B-t) + \frac{3}{2}B^2 + \frac{B}{2} + Bt \\ &> T + \frac{3}{2}B^2 + (\frac{3}{2}+t)B + t - t^2 \geq T + \frac{3}{2}B^2 + \frac{5}{2}B.\end{aligned}$$

All together, we get that the minimal possible total flow-time is $T + \frac{3}{2}B^2 + \frac{5}{2}B$, and this value can be achieved if and only if $\mathcal{I}$ has a schedule with total flow-time T and makespan B. ■

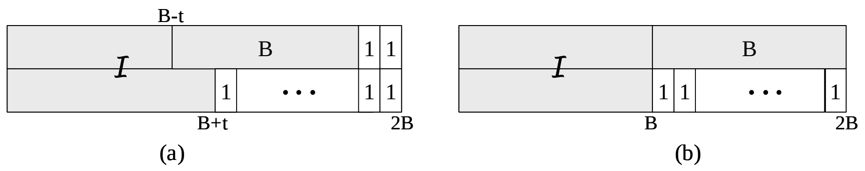

Figure 4: (a) A general and (b) an optimal solution for the reduced instance.