Scheduling unit-length jobs with precedence constraints of small height

André Berger[a], Alexander Grigoriev[a], Pinar Heggernes[b], Ruben van der Zwaan[a]

[a] *Operations Research Group, Maastricht University, PO Box 616, NL-6200 MD Maastricht, The Netherlands*
[b] *Department of informatics, University of Bergen, PO Box 7803 N-5020 Bergen, Norway*

Abstract

We consider the problem of scheduling unit-length jobs on identical machines subject to precedence constraints. We show that natural scheduling rules fail when the precedence constraints form a collection of stars or a collection of complete bipartite graphs. We prove that the problem is in fact NP-hard on collections of stars when the input is given in a compact encoding, whereas it can be solved in polynomial time with standard adjacency list encoding. On a subclass of collections of stars and on collections of complete bipartite graphs we show that the problem can be solved in polynomial time even when the input is given in compact encoding, in both cases via non-trivial algorithms.

Keywords: Scheduling, precedence constraints, computational complexity, polynomial-time algorithms

1. Introduction

The parallel machine scheduling problem with unit processing times and given precedence constraints is a well-known variant of general scheduling problems. An instance of this optimization problem consists of a set J of n jobs with unit processing times, a number m of parallel identical machines, and an acyclic digraph $D = (J, E)$, called the precedence graph, where a directed arc $(i, j) \in E$ means that job i has to be completed before job j starts. The goal is to find a schedule, basically an ordering of the jobs, that respects the precedence constraints and that minimizes the makespan, i.e., the completion time of the last job. Note that only the graph D and the integer m are needed as input, since all jobs have identical processing times.

This problem is long known to be NP-hard [1]. If the number of machines is fixed and not part of the input, then it is solvable in polynomial time for 1 and 2 machines [2, 3], whereas for any other fixed number $m \geq 3$ of machines the computational complexity of the problem is still open [4]. With respect to fixed-parameter tractability, it is known that the problem is $W[2]$-hard when parameterized by the number of machines [5].

Due to the difficulty and the importance of the problem, considerable attention has been devoted to cases where the precedence graph D is restricted to a special graph class. Hu [6] presented an algorithm which solves the problem in polynomial time if D is either an in-forest or an out-forest. On unions of in-forests and out-forests, Garey et al. [7] showed that the problem is NP-hard when the number of machines is part of the input, but polynomial-time solvable for any fixed number of machines. The polynomial-time algorithm for the latter case was improved twice by Dolev and Warmuth [8, 9], who also showed polynomial-time solvability when the number of machines is fixed and the precedence relation is a "level order" [8]. In addition, they gave a polynomial-time algorithm for the case where both the height of the precedence graph and the number of machines are bounded by a constant [10]. By the results of Papadimitriou [11] and Gabow [12], if the complement of the transitive closure of the precedence graph is an interval graph, then the problem can again be solved in polynomial time. More recently, Aho and Mákinen [13] gave a polynomial-time algorithm for a fixed number of machines and digraphs of bounded maximum degree and bounded height.

In this paper we give new results in form of an NP-hardness proof and polynomial-time algorithms for the case where the number of machines is part of the input and *not* bounded or fixed. In particular we study how the problem behaves when the precedence graph is restricted to collections of stars and collections of complete bipartite graphs. As we illustrate with several examples in Section 3, previously presented priority rules to schedule the jobs, which are known to be optimal for some classes of precedence graphs, fail to obtain an optimal schedule on collections of stars or complete bipartite graphs. A star or a complete bipartite graph can be compactly encoded as a pair of integers representing the number of vertices with in-degree 0 and out-degree 0. In Section 4, we show that on collections of stars our problem is NP-hard if the graph is compactly encoded, whereas it can be solved in polynomial time if the graph is given as an adjacency list. The hardness under compact encoding explains the failure of the simplistic rules as at least a hard number theoretical problem must be solved. In Section 5, we show that the problem can be solved in polynomial time on a subclass of collections of stars, namely collections of in–stars and out–stars, even when these are given in compact encoding. As an interesting contrast to the NP-hardness on compactly

encoded stars, in Section 6 we show that the problem can be solved in polynomial time on a collection of compactly encoded complete bipartite graphs.

2. Notation and terminology

For a positive integer n, we use $[n] = \{1, \ldots, n\}$. A *digraph* is denoted by $D = (J, E)$, where J is the set of vertices and E is the set of arcs. An arc $(i, j) \in E$ is *directed* from i to j. The *in-degree* of a vertex j, $d_{in}(j)$, is the number of arcs directed to it, and its *out-degree* $d_{out}(j)$ is the number of arcs directed from it. A vertex of in-degree (out-degree) 0 is called an *in-leaf* (*out-leaf*). A *path* in a digraph is a sequence of vertices $i_1, i_2, \ldots, i_p$ such that $(i_\ell, i_{\ell+1})$ is an arc for $1 \leq \ell \leq p-1$. A *cycle* is a path whose first vertex is the same as its last vertex. A digraph is *acyclic* if it does not contain a cycle. Every partial order corresponds to an acyclic digraph; throughout this paper we assume the input digraph to be acyclic. The *height* of a digraph is the number of vertices in a longest path, which is equivalent to the cardinality of a longest chain in the corresponding partial order.

Given a digraph $D = (J, E)$ with $|J| = n$ and an integer m representing the number of machines, a *feasible schedule* for the parallel machine scheduling problem with unit processing times is given by a mapping $\tau : J \to [n]$ that assigns jobs to time slots and satisfies:

1. $\tau(i) < \tau(j)$, for every $(i, j) \in E$, and
2. $|\{j \in J \mid \tau(j) = \ell\}| \leq m$, for every time slot $\ell \in [n]$.

The *makespan* of a schedule τ is the completion time of the last job, i.e., $\max_{j \in J} \tau(j)$. Hence, it is always possible to achieve a makespan of at most n. An *optimal schedule* is a schedule of minimum makespan. The problem is to compute the minimum makespan over all feasible schedules, given D and m.

An *in–tree* is a digraph, whose underlying undirected graph is a tree, such that $d_{out}(j) \leq 1$ for every $j \in J$. Similarly, an *out–tree* is a tree such that $d_{in}(j) \leq 1$ for every $j \in J$. An *in-forest* (*out-forest*) is a collection of in–trees (out–trees). An *in- and out-forest* is the union of an in-forest and an out-forest. We call D a *star* if the undirected graph underlying D is a star. The *center* of a star with at least three vertices is the only vertex that has degree at least 2. A star D is an *in–star* if it is an in–tree, and an *out–star* if it is an out–tree. A set consisting of in–stars and out–stars will be referred to as *a collection of in- and out–stars*. In this paper, a *complete bipartite graph* is a digraph, whose underlying undirected graph is complete bipartite, where all the edges are directed from one bipartition to the other.

A star with center c is uniquely defined by the pair $(d_{in}(c), d_{out}(c))$. We will call this a *compact encoding* of a star as opposed to a graph is given as an adjacency list or adjacency matrix. Similarly, a complete bipartite graph is uniquely defined by the number of its in-leaves and out-leaves. Hence, every complete bipartite graph can also be compactly encoded by a pair of integers. Note that in a compact encoding of a collection of stars or complete bipartite graphs, it is not the number n of vertices (jobs) but the number of stars or complete bipartite graphs (pairs of integers) in the collection that is an input parameter. We denote this number by k.

For simplicity we assume that the logarithms of the numbers are bounded by a polynomial in k. For an algorithm to be polynomial-time under compact encoding, it thus has to run in time polynomial in k. Clearly, although the minimum makespan might be computed in polynomial time, the corresponding schedule cannot be output explicitly, as it would require at least n steps. However, in our polynomial-time algorithms on compact input, the schedule will be implicit.

3. Natural scheduling rules fail on simple digraphs

Several natural scheduling rules do *not* produce an optimal schedule, even when restricted to collections of simple digraphs of small height. An example is Hu's algorithm [6], which solves the problem on in–trees. It schedules first the jobs having the longest directed path to any other job in the precedence graph. This rule is clearly not helpful for collections of stars or collections of complete bipartite graphs.

We provide more examples of such rules, illustrated in the figures below. In all of them we consider $m = 3$ machines. In each figure, the horizontal rows indicate the time slots; hence there can be at most three vertices per row. On the left side of the vertical line, the schedule resulting from the proposed rule is shown, whereas an optimal schedule is shown on the right side.

In the first three examples we consider collections of stars. Every presented rule describes a priority for the center vertex of each star, which dictates an ordering of the stars. The schedule is obtained by processing each center as soon as possible, in the order given by the rule. Figure 1 illustrates each of the rules: highest ratio between out-degree and in-degree, largest out-degree, and largest in-degree. Figures 2 and 3, respectively, illustrate the rules: highest ratio between in-degree and out-degree, and smallest in-degree with ties broken by largest out-degree.

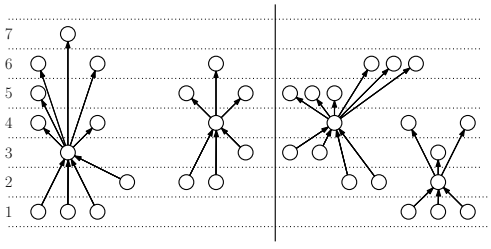

Figure 1: (Left) Largest $\frac{\text{out-degree}}{\text{in-degree}}$, Largest out-degree, Largest in-degree. (Right) Optimal schedule.

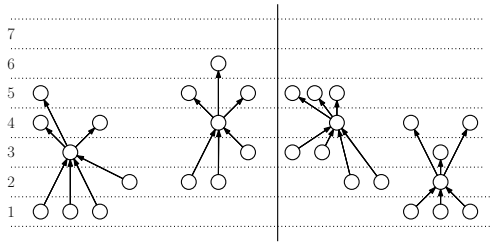

Figure 2: (Left) Largest $\frac{\text{in-degree}}{\text{out-degree}}$. (Right) Optimal schedule.

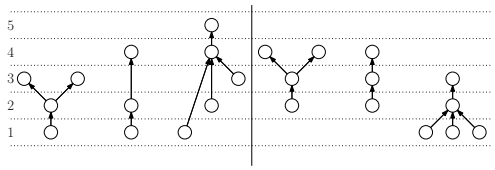

Figure 3: (Left) Smallest in-degree; ties broken by largest out-degree. (Right) Optimal schedule.

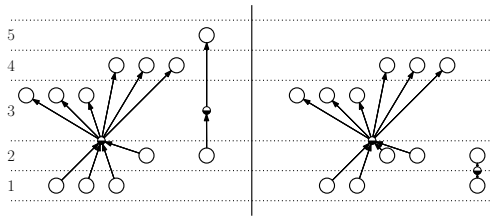

Figure 4: (Left) Largest $\frac{\text{out-degree}}{\text{in-degree}}$, Largest out-degree, Largest in-degree. (Right) Optimal schedule.

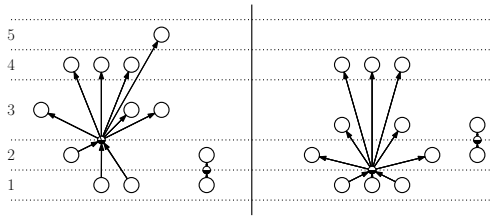

Figure 5: (Left) Smallest $\frac{\text{out-degree}}{\text{in-degree}}$, Smallest out-degree, Smallest in-degree. (Right) Optimal schedule.

In the last two examples we consider collections of complete bipartite graphs. Since all in-leaves of a complete bipartite graph have to be processed before any out-leaf of that graph, we draw each graph as a star by inserting a smaller half-filled center vertex representing a job of processing time 0 that separates the in-leaves from the out-leaves. We apply the same idea of processing the centers as soon as possible, subject to the order of the graphs dictated by the following rules. Figure 4: highest ratio between out-leaves and in-leaves, largest number of in-leaves, and largest number of out-leaves. Figure 5: lowest ratio between out-leaves and in-leaves, smallest number of in-leaves, and smallest number of out-leaves.

4. Complexity of scheduling stars

In this section we first show that our optimization problem is NP-hard on collections of compactly encoded stars. Our main focus in this paper is on compactly encoded inputs, however, to give a complete picture we show at the end of this section that the problem can be solved in polynomial time when the stars are given as adjacency lists.

We obtain our hardness result by showing that the decision version of our problem on compactly encoded stars is NP-complete. In particular, we show that deciding whether the minimum makespan is at most 4 is NP-complete on collections of stars if each star S_i is given as a pair of integers $(d_{in}(c_i), d_{out}(c_i))$, where c_i is the center vertex of S_i. Note that, for any collection of stars, it is easy to decide whether the minimum makespan is at most 3.

Theorem 1. *Deciding whether the minimum makespan is at most* 4 *is NP-complete on a collection of compactly encoded stars.*

Proof. The mentioned decision problem is clearly in NP. We use a reduction from the well-known NP-complete [14] problem Partition: Given positive integers $a_1, ..., a_k$ and an integer b with $\sum_{i=1}^{k} a_i = 2b$, is there a subset $T \subset [k]$ such that $\sum_{i \in T} a_i = b$?

Given an instance of Partition, we define an instance of our problem as follows. For every a_i we make a star S_i whose center has in-degree and out-degree $2ka_i$, and we let the number of machines be $m = 2kb + k$. Note that this instance can be compactly encoded in size polynomial in the size of the Partition instance. We claim that the answer to the Partition instance is yes if and only if there exists a schedule with makespan 4.

Let $T \subset [k]$ in the Partition instance, such that $\sum_{i \in T} a_i = b$. For all the stars S_i with $i \in T$, we can schedule their in-leaves in time slot 1, their centers in time slot 2, and their out-leaves in time slot 3. For the remaining stars, their jobs are scheduled in time slots 2, 3 and 4 accordingly, and thus we obtain makespan 4. Note that there will be k idle machines in time slots 1 and 4.

For the opposite direction, assume that there exists a schedule with makespan 4. This means that all in-leaves are scheduled in time slots 1 and 2, all centers are scheduled in time slots 2 and 3, and all out-leaves are scheduled in time slots 3 and 4. Let T be the set of indices i such that all in-leaves of the star S_i are scheduled in time slot 1. Clearly, $\sum_{i \in T} 2ka_i \leq 2kb + k = m$, and hence $\sum_{i \in T} a_i \leq b + 1/2$. We claim that $\sum_{i \in T} a_i = b$. Let ℓ_3 and ℓ_4 be the number of out-leaves scheduled in time slots 3 and 4, respectively. Certainly, $\ell_3 + \ell_4 = 4kb$ and thus $\ell_3 \geq 2kb - k$ since $\ell_4 \leq 2kb + k = m$. Moreover, if an out-leaf is scheduled in time slot 3, then all the in-leaves of the same star must be scheduled in time slot 1, hence $\sum_{i \in T} 2ka_i \geq \ell_3$. Combining the inequalities involving ℓ_3 yields $\sum_{i \in T} a_i \geq b - 1/2$. Since all numbers are integers it follows that there is a partition as claimed. □

Corollary 2. *It is NP-hard to approximate the minimum makespan within* $5/4$ *on a collection of compactly encoded stars.*

To tighten our hardness result, we now show that the problem becomes polynomial-time solvable when the input is given in standard encoding, i.e., as an adjacency list or adjacency matrix. The following lemma is central in several of our results.

Lemma 3 (folklore)**.** *The minimum makespan of a digraph* D *with* n *jobs and* m *machines is at most* $\lceil \frac{n}{m} \rceil + h - 1$*, where* h *is the height of* D*.*

The following notation will be used throughout this and the next section. Let $D = \{S_1, \ldots, S_k\}$ be the collection of stars that are given in the input. Let A_i and B_i be the sets of in-leaves and out-leaves of each star excluding the center, respectively, for $1 \leq i \leq k$. Let furthermore A be the union of all in-leaves, B the union of all out-leaves, and C the set of all centers. For any schedule τ we define the *switchpoint* $s_\tau = \min_{y \in B} \tau(y)$ to be the first time slot in τ when an out-leaf is scheduled.

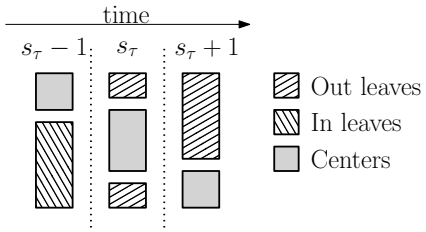

Figure 6: Switchpoint s_τ for schedule τ.

Lemma 4. *There is an optimal schedule* τ *that satisfies all of the following properties.*

(Q1) Jobs with no predecessors are scheduled no later than jobs with no successors;
(Q2) $\tau(x) \leq \tau(y)$ *for all* $x \in A$ *and* $y \in B$*;*
(Q3) $\tau(x) \leq \tau(y)$ *for all* $x \in A$ *and* $\tau(x) < s_\tau$ *and* $y \in C$ *and* $\tau(y) < s_\tau$*.*

Proof. Property (Q1) is easy to verify. Property (Q2) follows from (Q1), and property (Q3) follows from the observation that no jobs from B are scheduled before time s_τ therefore we can reorder the jobs such that jobs from A are scheduled no later than jobs from C. Figure 7 depicts such an optimal schedule. □

Theorem 5. *An optimal schedule for a collection of stars can be computed in time polynomial in the number of jobs.*

Proof. As this result is not within the main focus of the paper and it is given mainly to complement the hardness result, we only sketch its proof. The described dynamic programming algorithm is far from efficient, as our goal is merely to prove polynomial-time solvability. For

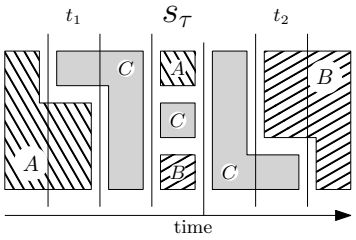

Figure 7: Typical schedule satisfying Lemma 4.

a given schedule τ satisfying Lemma 4, let $t_1(\tau)$ be the first time slot at which a center vertex is scheduled, and let $t_2(\tau)$ be the last time slot at which a center vertex is scheduled. Clearly the number of possibilities for triples $(t_1(\tau), s_\tau, t_2(\tau))$ is $O(n^3)$, and hence we can loop over all such possibilities which will add a polynomial factor to what is explained next. Consequently, we now assume that $t_1(\tau), s_\tau, t_2(\tau)$ are known (fixed). From Lemma 4 (See Figure 7) we have that the time slots are essentially partitioned into 7 different intervals. If stars $S_1, ..., S_j$ are already scheduled, the only information needed to schedule $S_{j+1}, ..., S_k$ is how many in-leaves, centers, and out-leaves are scheduled in every partition. In other words, from a dynamic programming perspective there are at most a polynomial number of states. Notice that to decide where to assign the jobs from S_j, there are 5 options to place its center, and its in-leaves can be distributed over at most 3 places and equivalently 3 places for its out-leaves. Therefore, evaluating each state we spend polynomial time, and the total running time is polynomial in n, the number of jobs. □

5. Collections of in- and out–stars

In this section we show that if D is a collection of k in– and out–stars, then the problem can be solved in time polynomial in k. Particularly, this means that the running time is polynomial in the size of the compactly encoded input.

Let $\{(d_{in}(c_1), d_{out}(c_1)), \ldots, (d_{in}(c_k), d_{out}(c_k))\}$ and m be the input to our scheduling problem. Note that either $d_{in}(c_i) \leq 1$ or $d_{out}(c_i) \leq 1$, for $1 \leq i \leq k$, since we have a collection of in– and out–stars. We assume that the input is ordered such that $d_{in}(c_i) \leq d_{in}(c_j)$, and $d_{out}(c_i) \geq d_{out}(c_j)$ if $d_{in}(c_i) = d_{in}(c_j)$, for $1 \leq i < j \leq k$. Let the sets $A_1, ..., A_k, B_1, ..., B_k, A, B$ be defined as in the previous section. Although these sets are not given explicitly, we will use them for ease of discussion below.

Lemma 6. *There is an optimal schedule* τ *that satisfies all of the following properties.*

(P1) $\tau(x) \leq \tau(y)$*, for all* $x \in A$ *and* $y \in B$*;*
(P2) $\tau(c_i) \leq \tau(c_j)$*, for all* $i < j$*;*
(P3) $\tau(x) \leq \tau(y)$*, for all* $i < j$*,* $x \in A_i$ *and* $y \in A_j$*;*
(P4) $\tau(c) \leq s_\tau + 1$*, for any center* c*.*

(P5) There is no $i \in \{1,\dots,k\}$ such that $\tau(c_i) = s_\tau + 1$ and $\tau(x) < s_\tau$ for every x in A_i.

Proof. Property (P1) follows from Lemma 4.

Suppose there is an optimal schedule τ that satisfies property (P1), such that there are two centers c_i, c_j with $i < j$ and $\tau(c_j) < \tau(c_i)$. By the ordering assumption, $|A_i| \leq |A_j|$, and if $|A_i| = |A_j|$ then $|B_i| \geq |B_j|$. We can obtain a new schedule τ' with the same makespan and with $\tau'(c_i) \leq \tau'(c_j)$ as follows. First swap c_i and c_j and then swap the jobs in A_i with $|A_i|$ jobs from A_j. If $|A_j| > 1$, then $|B_j| = 1$ and the sole job from B_j can be swapped with any job from B_i. Otherwise, i.e., if $|A_i| = |A_j| = 1$, then $|B_i| \geq |B_j|$, and all jobs from B_j can be scheduled after c_j by swapping them with $|B_j|$ many jobs from B_i. Hence τ' satisfies property (P2). Property (P1) is maintained as no job from B is moved to a time where previously no job from B was present.

Suppose there is an optimal schedule τ that satisfies properties (P1) and (P2), such that $i < j$, $x \in A_i$, $y \in A_j$, and $\tau(x) > \tau(y)$. Then $\tau(y) < \tau(x) < \tau(c_i) \leq \tau(c_j)$, from which it can be observed that swapping x and y in the schedule neither changes the feasibility nor the makespan of τ. Repeating this, we obtain property (P3). Properties (P1) and (P2) are both maintained, as no jobs from B are moved and no centers are moved.

Suppose there is an optimal schedule τ that satisfies properties (P1), (P2), and (P3). Among all such optimal schedules let τ be one that has the maximum number of centers scheduled before or at time $s_\tau + 1$. By definition of s_τ, all jobs from A are scheduled in time slot s_τ or earlier. Assume now that there is a center c such that $\tau(c) > s_\tau + 1$. We will show that without increase of the makespan we can change the schedule in such a way that $\tau(c)$ becomes at most $s_\tau + 1$. If there is $y \in B$ with $\tau(y) = s_\tau + 1$, then we can switch y and c without changing feasibility or the makespan. If there is an idle machine in time slot $s_\tau + 1$, then we can move c to time $s_\tau + 1$. If the previous two cases do not occur, then during time slot $s_\tau + 1$ only centers are scheduled. Therefore, in time slot s_τ at most $m - 1$ centers and jobs from A are scheduled and at least one job $z \in B$ with $\tau(z) = s_\tau$. The center c_ℓ with $\ell = \min\{l \mid 1 \leq l \leq k \text{ and } \tau(c_l) = s_\tau + 1\}$ can be swapped with z since all jobs in A_ℓ are scheduled before time slot s_τ. Because of property (P3) we can swap c_ℓ with z and then z with c, which gives us property (P4) as argued above. Property (P1) is maintained since jobs from B are only moved to later times. Property (P2) is maintained because the center with the lowest index is moved to $s_\tau + 1$. Property (P3) is maintained since no jobs from A are moved.

Suppose that there is an optimal schedule τ that satisfies properties (P1), (P2), (P3) and (P4). For property (P5) assume that in timeslot $s_\tau + 1$ there is a center c_i scheduled and all jobs in A_i are scheduled before timeslot s_τ. By the definition of s_τ there is at least one job y from B scheduled in timeslot s_τ. Because all predecessors of c_i are scheduled before timeslot s_τ, we can swap y and c_i without increasing the makespan. Clearly properties (P1), (P2), (P3) and (P4) remain satisfied. □

Due to Lemma 6, we can restrict ourselves to schedules τ satisfying properties (P1)-(P5). By property (P1) we have in particular that $\tau(x) \leq s_\tau$ for any $x \in A$.

To be used in this and also in the next section, the number of machines that are not assigned any jobs at a time slot will be called the *idle time* of that time slot. The *idle time of a schedule* is the sum of the idle times of all the time slots of the schedule.

Let $s[i]$ be the minimum makespan needed to schedule all jobs in $A_1, ..., A_i$ and the centers $c_1, ..., c_i$, for $1 \leq i \leq k$. Let $f[i]$ be the amount of idle time in such a schedule. Observe that $f[i] = m \cdot s[i] - (|A_1| + \ldots + |A_i| + i)$. We define $s[0] = f[0] = 0$. Let us show how $s[i]$ can be computed.

Lemma 7. *All values $s[1], \dots, s[k]$ and the center placements in the corresponding schedules can be computed altogether in total time $O(k^2)$ using the following formula: For $1 \leq i \leq k$, $s[i] =$*

$$\min_{0<\ell\leq i} \left\{ s[i-\ell] + \max\left\{0, \left\lceil \frac{\sum_{j=i-\ell+1}^{i} |A_j| - f[i-\ell]}{m} \right\rceil \right\} + \left\lceil \frac{\ell}{m} \right\rceil \right\}.$$

Proof. First we prove that the formula for $s[i]$ is correct. The proof is by induction on i, and for $i = 1$ we clearly have $s[1] = \left\lceil \frac{|A_1|}{m} \right\rceil + 1$. Now consider $i > 1$. To see that the right-hand side of the equation is an upper bound on $s[i]$, observe that the following is a feasible schedule, for every $\ell \in \{1, \dots, i\}$: $c_{i-\ell+1}, \dots, c_i$ are scheduled in the last $\left\lceil \frac{\ell}{m} \right\rceil$ time slots, $A_1, \dots, A_{i-\ell}$ and $c_1, \dots, c_{i-\ell}$ are scheduled in the first $s[i-\ell]$ time slots, and the remaining in-leaves in $A_{i-\ell+1}$, ..., A_ℓ either fit in the idle time or are scheduled the idle time plus $\left\lceil \frac{\sum_{j=i-\ell+1}^{i} |A_j| - f[i-\ell]}{m} \right\rceil$ slots.

To see that the right-hand side of the equation is also a lower bound on $s[i]$, consider an optimal schedule τ for all jobs in $A_1, ..., A_i$ and the centers $c_1, ..., c_i$ that has makespan $s[i]$. According to property (P2) of Lemma 6 we can assume that $\tau(c_{j_1}) \leq \tau(c_{j_2})$, for all $1 \leq j_1 < j_2 \leq i$. In the last time slot only centers are scheduled, and because of the previous property these are $c_{i-\ell+1}, ..., c_i$ for some $\ell \in \{1, \dots, i\}$. When we remove the centers $c_{i-\ell+1}, ..., c_i$ and the last time slot from the schedule, we obtain a feasible schedule for $A_1, \dots, A_i$ and $c_1, \dots, c_{i-l}$ with makespan $s[i]-1$. However, if we schedule $A_1, \dots, A_{i-l}$ and $c_1, \dots, c_{i-l}$ optimally with makespan $s[i-l]$ and schedule the remaining jobs in $A_{i-l+1}, \dots, A_i$ as described above, we obtain an optimal schedule for $A_1, \dots, A_i$ and $c_1, \dots, c_{i-l}$ with makespan $s[i-\ell] + \max\left\{0, \left\lceil \frac{\sum_{j=i-\ell+1}^{i} |A_j| - f[i-\ell]}{m} \right\rceil \right\}$. Such an schedule is optimal since it either has makespan $s[i-l]$ or at most one time slot has idle time (if $\sum_{j=i-\ell+1}^{i} |A_j| > f[i-\ell]$). Therefore $s[i] - \lceil \frac{l}{m} \rceil = s[i] - 1 \geq s[i-\ell] + \max\left\{0, \left\lceil \frac{\sum_{j=i-\ell+1}^{i} |A_j| - f[i-\ell]}{m} \right\rceil \right\}$, which proves the lower bound.

For the running time, recall first that we have every number $|A_j|$ readily available as $d_{in}(c_j)$ for $1 \leq j \leq k$. As a preprocessing step, we can compute and store $\sum_{j=1}^{i} |A_j|$ for every $i \in \{1, \ldots, k\}$. Since for every i this value can be computed from the value for $i-1$ in constant time, it can be done in total $O(k)$ time. The values $s[1]$ and $f[1]$ can be computed in constant time from $s[0]$ and $f[0]$ using the formulas. Now, for every $i \in \{2, \ldots, i\}$, $f[i]$ can be computed in constant time from the previously computed values $f[0], \ldots, f[i-1]$ and $s[0], \ldots, s[i-1]$ in constant time, whereas $s[i]$ can be computed from those values in $O(i)$ time. Hence in total, all mentioned values can be computed and stored in $O(k^2)$ time and space. For center placements in the corresponding schedules, for $1 \leq i \leq k$, we can clearly compute and store $\tau_i(c_j)$ for $1 \leq j \leq i$, where τ_i is the schedule with makespan $s[i]$, in $O(i)$ time, as described above. Hence the computation and storage of all center placements for all $s[1], \ldots, s[k]$ takes also in total $O(k^2)$ time and space. □

We are now ready to prove the main theorem of this section.

Theorem 8. *The minimum makespan for a collection of k compactly encoded in- and out–stars and the corresponding implicit optimal schedule can be computed in time $O(k^2)$.*

Proof. We describe an algorithm for this purpose with the given running time. Our algorithm has k iterations. For $i \in \{1, \ldots k\}$, it runs the following steps, and after completing the k iterations, it picks the schedule that has the smallest makespan:

1. Create a schedule of the jobs $A_1, ..., A_i$ and centers $c_1, ..., c_i$ with makespan $s[i]$ and idle time $f[i]$, according to Lemma 7.
2. Greedily insert jobs $A_{i+1}, ..., A_k$ into the schedule by first filling up idle time.
3. Let $t > s[i]$ be the first timeslot after the switch point that has idle time and let q denote the amount of idle time in timeslot t.
4. Schedule as many of the centers $c_{i+1}, ..., c_{i+q-1}$ as possible in timeslot t, and the remaining centers greedily in timeslot $t+1$ and further.
5. Greedily schedule jobs from $B_1, ..., B_k$ starting at timeslot t.

Note that step 4 leaves idle time at least 1 in timeslot t (only $q-1$ centers are scheduled at time t), and thus step 5 places at least one job from B in timeslot t.

Let τ be an optimal schedule satisfying properties (P1)-(P5) of Lemma 6, and let $i^* = \max\{i \mid \tau(c_i) < s_\tau\}$, i.e., the index of the last center scheduled before the switchpoint.

By property (P1) in both schedules, τ and τ_{alg}, before or at the switchpoint all jobs in A as well as the centers $c_1, \ldots, c_{i^*}$ are scheduled. Therefore we have $s_{\tau_{alg}} = s_\tau = \left\lfloor \frac{|A|+i^*}{m} \right\rfloor + 1$.

Note in particular that no center is placed in time slot $s_\tau + 2$ or further by either schedule. This is true for τ since it satisfies property (P4), and it is true for τ_{alg} by construction since there is enough space to place all jobs $c_{i^*+1}, \ldots, c_k$ in time slots s_τ and $s_\tau + 1$ as described, by the choice of i^*.

Consequently, the difference between schedules τ and τ_{alg} lies in which centers are scheduled in timeslot s_τ and which in timeslot $s_\tau + 1$. Let $c_{i^*+1}, ..., c_p$ be scheduled by τ and let $c_{i^*+1}, ..., c_{p_{alg}}$ be scheduled by τ_{alg} in timeslot s_τ. By the description of our algorithm, $p_{alg} \geq p$, and by property (P5), $p \geq p_{alg}$. Consequently $p = p_{alg}$ and the two schedules have identical makespan. Since our algorithm outputs a schedule with makespan less than or equal to this makespan, it outputs an optimal schedule.

For the running time, instead of running step 1 separately at each iteration, we can run it as a preprocessing step and compute all values $s[1], \ldots, s[k]$ and the center placements in the corresponding schedules in $O(k^2)$ time, due to Lemma 7. For a given i, all other steps take $O(i)$, and hence the total running time is $O(k^2)$. Furthermore, outputting $\tau_{alg}(c_i)$ for each center i defines an implicit schedule and this can be done within the overall running time by Lemma 7 and the description of our algorithm. □

6. Collections of complete bipartite graphs

In this section we show that on a collection of k compactly encoded complete bipartite graphs, our optimization problem can be solved in time polynomial in k.

Let $\{(a_1, b_1), (a_2, b_2), \ldots, (a_k, b_k)\}$ and m be the input to our problem, where each (a_i, b_i) represents a complete bipartite graph G_i with a_i in-leaves and b_i out-leaves. We assume the input to be sorted so that $a_i \leq a_j$ if $i < j$. As before, for convenience we let A_i be the set of in-leaves of G_i and B_i be the set of out-leaves of G_i. We let A be the union of all in-leaves and B the union of all out-leaves, with $a = |A|$, $b = |B|$, and $a + b = n$. Note that if $a_i = 0$ or $b_i = 0$ then G_i consists of isolated independent jobs that can be scheduled to fill-in the idle time in any optimal schedule. Hence we will assume that $a_i, b_i > 0$, for $1 \leq i \leq k$. Finally, let OPT be the minimum makespan for this instance.

From Lemma 3 it follows that there are only two possible values for the minimum makespan:

$$\left\lceil \frac{a+b}{m} \right\rceil \leq OPT \leq \left\lceil \frac{a}{m} \right\rceil + \left\lceil \frac{b}{m} \right\rceil \leq \left\lceil \frac{a+b}{m} \right\rceil + 1.$$

Consequently, if $\lceil \frac{a}{m} \rceil + \lceil \frac{b}{m} \rceil = \lceil \frac{a+b}{m} \rceil$, then it is trivial to find the optimal schedule. Hence from now on we assume that $\lceil \frac{a}{m} \rceil + \lceil \frac{b}{m} \rceil = \lceil \frac{a+b}{m} \rceil + 1$. Let $t_A \equiv a \mod m$, and $t_B \equiv b \mod m$. If $t_A = 0$ or $t_A + t_B > m$ then clearly $\lceil \frac{a}{m} \rceil + \lceil \frac{b}{m} \rceil = \lceil \frac{a+b}{m} \rceil$. Thus our assumption implies that $0 < t_A + t_B \leq m$.

Without loss of generality, by the same arguments as in Lemma 4, we consider only schedules where all jobs in A

are scheduled no later than any job in B. In particular, we define a specific schedule τ of this type as follows: Schedule first all the jobs in A_1, then all the jobs in A_2, and so on, until all the jobs in A_k are scheduled. Then schedule jobs from B_1 in the earliest time slot which is available for these jobs, and continue with B_2 and so on, until all the jobs are scheduled. We know by the examples from Section 3 that τ is not necessarily optimal. We will show that either τ is optimal or we can modify it in polynomial time to obtain an optimal schedule τ. If the makespan of τ is $\lceil \frac{a+b}{m} \rceil$, then we know by Lemma 3 that τ is optimal. Hence for the rest of the section we assume that τ has makespan $\lceil \frac{a+b}{m} \rceil + 1$.

Similar to what we did in Section 4, we define the *switchpoint* of τ to be the first time slot s_τ in which a job from B is scheduled.

Lemma 9. *If one of the following conditions is satisfied then τ is an optimal schedule.*

(R1) $a \leq m$ or $b \leq m$ or $k = 1$;
(R2) there is no idle time in time slot s_τ;
(R3) the makespan of τ is $s_\tau + 1$.

Proof. Due to our assumption that every vertex in B has an arc directed to it, in any optimal schedule the first time slot cannot contain any jobs from B and the last time slot cannot contain any job from A. Hence if τ satisfies (R1), it follows that τ is an optimal schedule.

Assume that τ does not satisfy (R1). Hence, $a > m$, $b > m$, and $k > 1$. Let $s = \lceil \frac{a}{m} \rceil$. Since $t_A \neq 0$, time slot s of τ contains at least 1 and at most $m - 1$ jobs from A. Note that $s_\tau = s$ or $s_\tau = s + 1$. Assume that τ satisfies (R2); hence time slot s_τ has no idle time. If $s_\tau = s$, this means that s contains jobs both from A and from B, and hence there is no idle time in any time slot of τ except possibly the last. Consequently, τ is optimal. If $s_\tau = s + 1$, then since no job from B_1 could be scheduled in time slot s, there are jobs from A_1 in time slot s. Since a_1 is the smallest among all a_i, this means that $s = 1$ or $k = 1$. In both cases τ is optimal.

Assume finally that τ does not satisfy (R1) or (R2), and it satisfies (R3). By assumption $b > m$ and $n > 1$ and any optimal schedule must use time slot $s_\tau + 1$, and hence τ is optimal. □

Assume for the rest of the section that none of the properties (R1)-(R3) is satisfied by τ. In particular, we have idle time in time slot s_τ of τ and the makespan of τ is at least $s_\tau + 2$. Furthermore, recall that the makespan of τ is $\lceil \frac{a+b}{m} \rceil + 1$. Let h be the smallest index such that time slot s_τ of τ contains a job from A_h. More formally:

$$h = \min\{i \mid \tau(x) = s_\tau \text{ and } x \in A_i\}$$

Lemma 10. *Let $h = k$. If $a_k \leq (s_\tau - 1)m$ then $OPT = \lceil \frac{a+b}{m} \rceil$, and τ can be transformed into a schedule having this makespan in time $O(k)$. If $a_k > (s_\tau - 1)m$ then τ is an optimal schedule.*

Proof. Because $h = k$, the only jobs from A in time slot s_τ are from A_k. Since time slot s_τ has idle time and $b > m$, we can conclude that $\sum_{i=1}^{k-1} b_i < m - t_A$. Hence only jobs from B_k (and *all* jobs from B_k) are scheduled by τ in time slot $s_\tau + 1$ or later. Since we assume that property (R3) is not satisfied, we have $b_k \geq m$.

Suppose that $a_k \leq (s_\tau - 1)m$. We can schedule all jobs in A_k before all other jobs and before time slot s_τ, which allows us to fill time slot s_τ with jobs from B_k, and consequently we get a schedule with makespan $\lceil \frac{a+b}{m} \rceil$. This can clearly be done in time $O(k)$, since we do not move the jobs explicitly, but we can do it implicitly since all jobs of A_k go together.

Suppose that $a_k > (s_\tau - 1)m$. Then the jobs in A_k cannot be scheduled before time slot s_τ, and no job from B_k can be scheduled before or in time slot s_τ. This means that τ is optimal. □

Lemma 11. *If $h \leq k-1$, then $OPT = \lceil \frac{a+b}{m} \rceil$ and τ can be transformed into a schedule having this makespan in time $O(k)$.*

Proof. Because $h \leq k - 1$ all jobs from $A_{h+1}, \ldots, A_k$ are scheduled in time slot s_τ by τ, and s_τ contains a job from A_h. Since there is idle time in s_τ, all jobs from $B_1, B_2, \ldots, B_{h-1}$ are scheduled by τ in time slot s_τ. Consequently, $\sum_{i=1}^{h-1} b_i < m - t_A$. Since we assumed that property (R3) is not satisfied, $\sum_{i=h}^{k} b_i \geq m$. Recall that $t_A + t_B \leq m$, and the number of jobs from A in time slot s_τ is t_A.

Suppose that $t_B \leq m/2$. Hence if we can schedule at least $m/2$ jobs from B in time slot s_τ, we will obtain a schedule with makespan $\lceil \frac{a+b}{m} \rceil$. Since $\sum_{i=h}^{k} b_i \geq m$, then either $b_h \geq m/2$ or $\sum_{i=h+1}^{k} b_i \geq m/2$. By the way τ is constructed, we know that $a_h \leq \sum_{i=h+1}^{k} a_i \leq m$. Thus we can either move all the jobs in A_h or all the jobs in $A_{h+1} \cup \ldots \cup A_k$ to the first time slot, and be able to schedule at least $m/2$ jobs from B in time slot s_τ.

Suppose that $t_B > m/2$. Then we know that $t_A < m/2$, and hence $a_h \leq \sum_{i=h+1}^{k} a_i < m/2$. Therefore, we can move all jobs in $A_h \cup A_{h+1} \cup \ldots \cup A_k$ to the first time slot of the schedule. This will allow us to fill time slot s_τ with jobs from $B_h \cup B_{h+1} \cup \ldots \cup B_k$ and obtain makespan $\lceil \frac{a+b}{m} \rceil$.

In either case, the resulting schedule can be computed implicitly in time $O(k)$: we simply output an ordering of graphs representing the order of the the jobs from A. Then, the ordering of the jobs from B always follows the ordering of the jobs from A. □

Theorem 12. *The minimum makespan for a collection of k compactly encoded complete bipartite graphs can be computed and an optimal schedule can be computed implicitly in time $O(k \log k)$.*

Proof. We describe an algorithm for this purpose. Our algorithm initially checks whether or not $\lceil \frac{a}{m} \rceil + \lceil \frac{b}{m} \rceil =$

$\lceil \frac{a+b}{m} \rceil$, and outputs in the case of affirmative this as the minimum makespan. Otherwise it starts by implicitly computing the schedule τ defined above. Then it checks whether τ satisfies one of the properties (R1)-(R3) of Lemma 9. If one of these properties is satisfied then the algorithm simply outputs the makespan of τ and stops. If not, it continues to compute h as described above. Then it transforms τ into an optimal schedule according to Lemma 10 and Lemma 11. During these modifications, idle time is removed by moving jobs to earlier time slots wherever possible. This completes the description of the algorithm.

The correctness of the algorithm follows from Lemmas 9, 10, 11, and the preceding discussion. The described initial schedule τ can be implicitly constructed in polynomial time, as it amounts to a number of arithmetic operations that is polynomial in k. In fact the step that dominates the running time in the construction of τ is the sorting of the a_i, which can be done in $O(k \log k)$ time. The rest of the steps sum up to $O(k)$ time by Lemmas 10 and 11 and since each of the conditions (R1)-(R3) can easily be checked in $O(k)$ time. □

7. Open problems

It would be interesting to study the complexity of our problem on collections of graphs belonging to superclasses of stars or complete bipartite graphs. One such graph class is the class of chain graphs. In the undirected version, a bipartite graph is a *chain graph* if the vertices of the two independent sets A and B can be ordered by neighborhood inclusion, i.e., there is an order $x_1, x_2, \ldots$ of the vertices in A such that the neighbors of x_i form a subset of the neighbors of x_{i+1} for $i = 1, 2, \ldots$. A can be ordered in this way if and only if B can be ordered in this way at the same time. We can now define *directed* chain graphs as digraphs whose underlying undirected graphs are chains, and all edges are directed from A to B. What is the computational complexity of scheduling unit-length jobs on parallel identical machines if the precedence graph is a collection of chain graphs?

Acknowledgements

We are grateful to an anonymous referee whose comments and suggestions have greatly improved the presentation of the paper.

References

[1] J. Ullman, NP-complete scheduling problems, Journal of Computer and System Sciences 10 (3) (1975) 384 – 393.

[2] M. Fuji, T. Kasami, N. Ninomiya, Optimal sequence of two equivalent processors, SIAM J. Appl. Math. 17 (1969) 784 – 789.

[3] H. N. Gabow, An almost-linear algorithm for two-processor scheduling, J. ACM 29 (3) (1982) 766–780.

[4] G. J. Woeginger, Open problems in the theory of scheduling, Bulletin of the European Association for Theoretical Computer Science 76 (2002) 67 – 83.

[5] H. L. Bodlaender, M. R. Fellows, W[2]-hardness of precedence constrained k-processor scheduling, Operations Research Letters 18 (1995) 93 – 97.

[6] T. C. Hu, Parallel sequencing and assembly line problems, Operations Research 9 (6) (1961) 841 – 848.

[7] M. R. Garey, D. S. Johnson, R. E. Tarjan, M. Yannakakis, Scheduling opposing forests, SIAM. J. on Algebraic and Discrete Methods 4 (1983) 72 – 93.

[8] D. Dolev, M. K. Warmuth, Profile scheduling of opposing forests and level orders, SIAM. J. on Algebraic and Discrete Methods 6 (1985) 665–687.

[9] D. Dolev, M. K. Warmuth, Scheduling flat graphs, SIAM J. Comput. 14 (3) (1985) 638–657.

[10] D. Dolev, M. K. Warmuth, Scheduling precedence graphs of bounded height, J. Algorithms 5 (1) (1984) 48–59.

[11] C. H. Papadimitriou, M. Yannakakis, Scheduling interval-ordered tasks, SIAM J. Comput. 8 (3) (1979) 405–409.

[12] H. N. Gabow, A linear-time recognition algorithm for interval dags, Inf. Process. Lett. 12 (1) (1981) 20–22.

[13] I. Aho, E. Mákinen, On a parallel machine scheduling problem with precedence constraints, Journal of Scheduling 9 (2006) 493 – 495.

[14] M. R. Garey, D. S. Johnson, Computers and Intractability, W. H. Freeman and Co., 1978.