Operations Research Letters 42 (2014) 166–172

Contents lists available at ScienceDirect

Operations Research Letters

journal homepage: www.elsevier.com/locate/orl

Scheduling unit-length jobs with precedence constraints of small height

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

[a] Operations Research Group, Maastricht University, PO Box 616, NL-6200 MD Maastricht, The Netherlands

[b] Department of Mathematics and Computer Science, Technische Universiteit Eindhoven, PO Box 513, NL-5600 MB Eindhoven, The Netherlands

[c] Department of Informatics, University of Bergen, PO Box 7803, N-5020 Bergen, Norway

ARTICLE INFO

Article history:
Received 30 January 2012
Received in revised form
9 December 2013
Accepted 24 January 2014
Available online 5 February 2014

Keywords:
Scheduling
Precedence constraints
Computational complexity
Polynomial-time algorithms

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.

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 [13]. 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 [6,8], whereas for any other fixed number $m \geq 3$ of machines the computational complexity of the problem is still open [14]. With respect to fixed-parameter tractability, it is known that the problem is $W[2]$-hard when parameterized by the number of machines [2].

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 [11] 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. [10] 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 [4,5], who also showed polynomial-time solvability when the number of machines is fixed and the precedence relation is a "level order" [4]. 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 [3]. By the results of Papadimitriou [12] and Gabow [7], 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 [1] 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 the 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

* Corresponding author.
E-mail addresses: a.berger@maastrichtuniversity.nl (A. Berger), a.grigoriev@maastrichtuniversity.nl (A. Grigoriev), pinar@ii.uib.no (P. Heggernes), g.r.j.v.d.zwaan@tue.nl (R. van der Zwaan).

http://dx.doi.org/10.1016/j.orl.2014.01.008
0167-6377/

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.

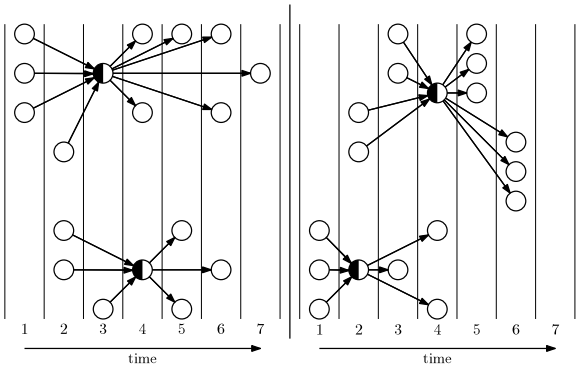

Fig. 1. (Left) Largest $\frac{\text{out-degree}}{\text{in-degree}}$, largest out-degree, largest in-degree. (Right) Optimal schedule.

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 [11], 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 vertical columns indicate the time slots; hence there can be at most three vertices per column. 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. Fig. 1 illustrates each of the rules: highest ratio between out-degree and in-degree, largest out-degree, and largest in-degree. Figs. 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.

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. Fig. 4: highest ratio between out-leaves and in-leaves, largest number of in-leaves, and largest number of out-leaves. Fig. 5: lowest ratio between out-leaves and in-leaves, smallest number of in-leaves, and smallest number of out-leaves.