ELSEVIER

Operations Research Letters 29 (2001) 241–245

operations research letters

www.elsevier.com/locate/dsw

A very difficult scheduling problem with communication delays

Han Hoogeveen[a,1], Gerhard J. Woeginger[b,c,*,2]

[a] *Department of Computer Science, Utrecht University, P.O. Box 80089, 3508 TB Utrecht, The Netherlands*
[b] *Department of Mathematics, University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands*
[c] *Institut für Mathematik, Technische Universität Graz, Steyrergasse 30, A-8010 Graz, Austria*

Received 24 July 2001; received in revised form 1 August 2001; accepted 17 August 2001

Abstract

A set of unit-time tasks has to be processed on an unrestricted number of processors subject to precedence constraints and unit-time communication delays such that the makespan is minimized. What is the smallest number m^* such that increasing the number of processors beyond m^* cannot decrease the makespan any more? We prove that answering this problem is complete for the complexity class $\mathbf{FP}^{\mathcal{NP}[\log n]}$. Hence, the problem is at least as difficult as all the problems in $\mathcal{NP}$ and at least as difficult as all the problems in co$\mathcal{NP}$, and unless some complexity classes collapse, it is even *more* difficult than all these problems. This answers a question raised by Ivan Rival.

Keywords: Parallel computation; Computational complexity; Scheduling; Makespan; Precedence constraints; Communication delays

1. Introduction

We consider the following type of scheduling problem with communication delays. There are n tasks that have to be executed by identical parallel processors. Each processor is available from time zero onwards and can execute at most one task at a time. The number of processors is not the bottleneck of the system and is considered to be unrestricted ('use as many processors as you like'). Each task J_j ($j = 1, \ldots, n$) has a unit processing time and produces information that may be required by one or more of the other tasks. These data dependencies define a partial order, the so-called *precedence relation*, on the task set. It is represented by an acyclic directed graph G with vertices $J_1, \ldots, J_n$ and an arc $J_j \rightarrow J_k$ whenever J_k needs data from J_j. For each arc $J_j \rightarrow J_k$, we require that J_k cannot be started before J_j has been completed and the information produced by J_j has been transferred from its processor to the processor of J_k. Transferring these data from one processor to another one takes one time unit, which is called a *communication delay*; there is no delay if J_j and J_k are executed on the same processor. It is assumed that the transmission of information does not interfere with the availability of the processors.

* Corresponding author.
E-mail addresses: slam@cs.uu.nl (H. Hoogeveen), g.j.woeginger@math.utwente.nl (G.J. Woeginger).

[1] Supported by EC Contract IST-1999-14186 (Project alcom-FT).
[2] Supported by the START program Y43-MAT of the Austrian Ministry of Science.

0167-6377/01/$ - see front matter
PII: S0167-6377(01)00103-1

A *schedule* is an allocation of each task to a time slot of unit length on a processor such that processor availability constraints, precedence constraints, and communication delays are satisfied. Given a schedule, we let C_j denote the completion time of task J_j ($j = 1, \ldots, n$). The primary goal in this scheduling problem is to minimize the *length* or the *makespan* of a schedule, that is the maximum task completion time $\max_{1 \leqslant j \leqslant n} C_j$. In the notation of Veltman, Lageweg, and Lenstra [9], this problem is denoted by $P\infty|\text{prec},\ c = 1,\ p_j = 1|C_{\max}$.

Some history of this scheduling problem: Papadimitriou and Yannakakis [5] were among the first to address this special scheduling problem. They study a model with general processing times and communication delays that allows for task duplication. Hoogeveen, Lenstra, and Veltman [2] proved that it is $\mathcal{NP}$-complete to decide whether there exists a schedule with makespan at most 6, whereas deciding the existence of a schedule with makespan at most 5 can be done in polynomial time. Chenier et al. [1] gave a polynomial time algorithm for the special case where the precedence constraints form a tree, and Möhring and Schäffter [3] derived a polynomial time algorithm for the case of series-parallel precedence constraints. Many other results and references may be found in the Ph.D. Theses of Picouleau [7] and Veltman [8].

Our result: A schedule that yields the minimum makespan is called an *optimum schedule*. Clearly, in any optimum schedule the n tasks use at most n processors. Ivan Rival suggested to investigate the problem of computing the *smallest* number m^* of processors that allows an optimum schedule (cf. the last paragraph in [1]). By modifying the techniques of Hoogeveen et al. [2], it is easy to show that Rival's problem is $\mathcal{NP}$-hard, but it is by no means clear why the problem should be contained in the complexity class $\mathcal{NP}$. In fact, we will prove that a stronger result holds: Rival's problem is contained in the complexity class $\mathbf{FP}^{\mathcal{NP}[\log n]}$, and it is complete for this class. This provides an exact classification of this problem from a computational complexity point of view. It also demonstrates that Rival's problem is *very* difficult: Unless several classes in the polynomial hierarchy collapse (which is considered to be very unlikely), the problem is more difficult than any $\mathcal{NP}$-complete and any co$\mathcal{NP}$-complete problem.

Organization of the paper: Section 2 provides all relevant information on computational complexity theory as needed for this paper. It defines the class $\mathbf{FP}^{\mathcal{NP}[\log n]}$ and explains its relations to other complexity classes. Section 3 gives the complexity proof for Rival's problem.

2. Preliminaries on computational complexity

This section gives some definitions and results from computational complexity theory as needed in the rest of the paper. We expect the reader to be familiar with the basic concepts of complexity theory. For more precise information, the reader is referred to Papadimitriou's book [4].

A decision problem, that is, a problem to which the answer is either 'Yes' or 'No', lies in the complexity class $\mathcal{NP}$ if and only if its Yes-instances have *certificates* of polynomial size that can be verified in polynomial time. For example, the Yes-instances of the decision problem 'Given a set of tasks with precedence constraints and a number d, is it true that the minimum makespan for this instance is at most d' have the corresponding optimum schedules as certificate: One only needs to check that the schedule obeys the precedence constraints and communication delays, that every processor processes at most one task at a time, and that the makespan is indeed at most d. This clearly can be done in polynomial time and puts the problem into $\mathcal{NP}$. Symmetrically, a decision problem is said to lie in the complexity class co$\mathcal{NP}$ if and only if its No-instances have certificates that can be verified in polynomial time.

Researchers in theoretical computer science have introduced a hierarchy of complexity classes, the so-called *polynomial hierarchy*. It consists of an infinite number of increasingly difficult classes, starting at the 'easiest' classes $\mathcal{NP}$ and co$\mathcal{NP}$ and extending up to PSPACE, the class of problems that can be solved within polynomial space. Near the bottom of the polynomial hierarchy sits $\mathbf{FP}^{\mathcal{NP}[\log n]}$, a complexity class that was first studied by Papadimitriou and Zachos [6]. A problem is in $\mathbf{FP}^{\mathcal{NP}[\log n]}$ if its instances x are decided by a polynomial time Turing machine, which is allowed $O(\log |x|)$ calls to an $\mathcal{NP}$ oracle (here $|x|$ denotes the length of the encoding of instance x). Intuitively speaking, a problem in $\mathbf{FP}^{\mathcal{NP}[\log n]}$ has

to be solved by a polynomial time algorithm that is allowed to make $O(\log|x|)$ calls to a subroutine that is able to solve any problem in $\mathcal{NP}$; the algorithm is not charged for the time needed by the subroutine.

A problem X is *hard* for a complexity class $\mathcal{C}$, if all problems in $\mathcal{C}$ are polynomial time reducible to X. Problem X is *complete* for $\mathcal{C}$, if X lies in $\mathcal{C}$ and also is hard for $\mathcal{C}$. Intuitively speaking, the complete problems for $\mathcal{C}$ are the most difficult problems in $\mathcal{C}$.

There are several 'natural' problems that are complete for the class $\mathbf{FP}^{\mathcal{NP}[\log n]}$, e.g. Clique Size (Given a graph, determine the size of its largest clique) or Bin Packing Number (Given a set of items of size at most one, determine the minimum number of unit-size bins into which these items can be packed). The general belief is that the decision versions of $\mathbf{FP}^{\mathcal{NP}[\log n]}$-complete problems are neither in $\mathcal{NP}$ nor in co$\mathcal{NP}$; otherwise part of the polynomial hierarchy would collapse, which is considered to be a very unlikely event. In Section 3, we will prove that Rival's scheduling problem is $\mathbf{FP}^{\mathcal{NP}[\log n]}$-complete. $\mathbf{FP}^{\mathcal{NP}[\log n]}$-hardness will be proved by a polynomial time reduction from the Max 3-Sat Size problem, which is known to be $\mathbf{FP}^{\mathcal{NP}[\log n]}$-complete [4, Chapter 17.1].

Max 3-Sat Size: Given a set of clauses with exactly three literals per clause, find the maximum number of clauses that can be satisfied simultaneously by a truth assignment.

3. The complexity of the scheduling problem

In this section, we take a closer look at Rival's problem as described in the introduction. For a set of tasks with precedence relation G and for a nonnegative integer m, let $\mathrm{Opt}(G,m)$ denote the minimum makespan achievable by a schedule that obeys all precedence constraints, communication delays, processor availability constraints, while using only m processors. For the sake of completeness, we define $\mathrm{Opt}(G,0)=+\infty$. Note that $\mathrm{Opt}(G,m)$ is a non-increasing function in m. Rival's problem may now be reformulated as follows.

> Given a set of n tasks with precedence relation G, compute the number m^* for which $\mathrm{Opt}(G,m^*)=\mathrm{Opt}(G,n)$ and $\mathrm{Opt}(G,m^*-1)>\mathrm{Opt}(G,n)$.

Lemma 3.1. *Rival's scheduling problem is contained in* $\mathbf{FP}^{\mathcal{NP}[\log n]}$.

Proof. First we observe that the problem 'Given a set of tasks with precedence relation G and numbers m and d, is it true that $\mathrm{Opt}(G,m)\leqslant d$' is in $\mathcal{NP}$, since the corresponding schedule may be used as certificate for the Yes-instances. We will devise a polynomial time algorithm that solves Rival's problem while asking at most $2\lceil\log_2 n\rceil$ such questions; this clearly proves the lemma.

In the first phase, the algorithm computes the optimum makespan $d^*=\mathrm{Opt}(G,n)$ for the instance. Since $1\leqslant d^*\leqslant n$ holds, this is easily done by binary search over the range $[1,\ldots,n]$ while making at most $\lceil\log_2 n\rceil$ calls to the $\mathcal{NP}$-solving subprogram. In the second phase, the algorithm performs another binary search over the number of processors. This time, the goal is to find the smallest m^* for which $\mathrm{Opt}(G,m^*)\leqslant d^*$ holds. □

In the remainder of the paper, we give an $\mathbf{FP}^{\mathcal{NP}[\log n]}$-hardness reduction for Rival's problem. This reduction is based on similar ideas as used by Picouleau [7] and by Hoogeveen et al. [2]. We start from an instance for Max 3-Sat Size. Let this instance have $U=\{x_1,\ldots,x_u\}$ as its set of u logical variables, which appear in this instance, either negated or unnegated, and let $C=\{c_1,\ldots,c_v\}$ be the set of v clauses over U in this instance, where every clause consists of exactly three literals.

From this Max 3-Sat Size instance, we will construct in polynomial time a set of $n=5u+10v+9$ precedence constrained tasks. The optimum makespan of the constructed scheduling instance will be nine. The tasks are divided into three groups: Structure tasks, variable tasks and clause tasks.

First there are $u+3v+9$ *structure tasks*: Nine of them are the tasks $s_1,s_2,\ldots,s_9$ where s_i has to precede s_{i+1} for $1\leqslant i\leqslant 8$. The remaining $u+3v$ structure tasks are called *dummy* tasks; they all precede task s_3. Observe that in any schedule with makespan nine, the chain of tasks $s_1,\ldots,s_9$ is continuously processed on a single processor such that task s_i occupies time slot i, $1\leqslant i\leqslant 9$. The dummy tasks block time slot 1 of $u+3v$ other processors.

For every variable x_i, there are four *variable tasks* called a_i, x_i, $\overline{x_i}$ and b_i. The structure task s_4 precedes

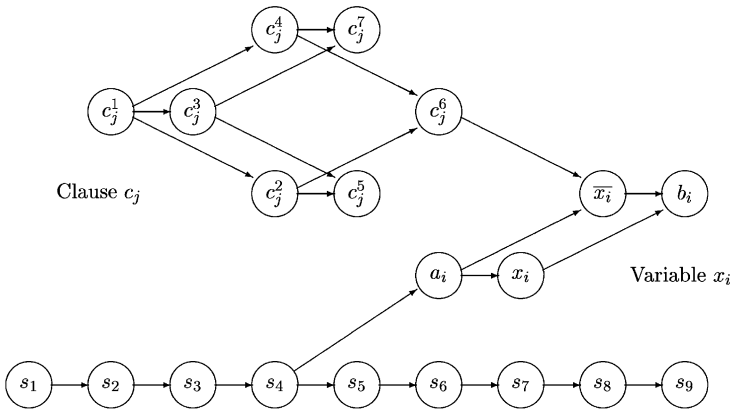

Fig. 1. The nine structure tasks $s_1,\ldots,s_9$, seven tasks corresponding to clause c_j, and four tasks corresponding to variable x_i. Time proceeds horizontally. Tasks on the same horizontal line are scheduled on the same machine.

a_i, a_i precedes the two tasks x_i and $\overline{x_i}$, and these two tasks precede b_i. We say that task x_i corresponds to the unnegated literal x_i and that $\overline{x_i}$ corresponds to the negated literal. In any schedule with makespan nine, task a_i has to be processed during time slot 6, one of x_i and $\overline{x_i}$ is processed during time slot 7 and the other one during time slot 8, and task b_i is processed during time slot 9.

For every clause c_j, there are seven *clause tasks* c_j^k ($k=1,\ldots,7$). Task c_j^1 precedes c_j^2, c_j^3, and c_j^4. Task c_j^2 precedes the tasks c_j^5 and c_j^6, task c_j^3 precedes the tasks c_j^5 and c_j^7, and task c_j^4 precedes the tasks c_j^6 and c_j^6. Finally, the task corresponding to the first (respectively, second and third) literal in clause c_j is a successor of the task c_j^5 (respectively, c_j^6, and c_j^7).

There are no other precedence relation between the tasks. This completes the description of the scheduling instance. For an illustration see Fig. 1. This figure corresponds to an instance in which $\overline{x_i}$ is the second literal contained in the clause c_j.

Claim 3.2. *The optimum makespan of the constructed scheduling instance is nine.*

Proof. Consider the following schedule with makespan 9. All clause tasks c_j^1 are processed during time slot 1, all tasks c_j^2, c_j^3, and c_j^4 during time slot 3, and all tasks c_j^5, c_j^6, and c_j^7 during time slot 5. All variable tasks a_i are scheduled during time slot 6, x_i during time slot 7, $\overline{x_i}$ during time slot 8, and b_i during time slot 9. The structure tasks are scheduled in the obvious way. □

Proposition 3.3. *There exists an optimum schedule in which task c_j^1 is processed during time slot* 2 *if and only if at least one of the tasks corresponding to a literal in clause c_j is processed during time slot* 8. *Moreover, there exists an optimum schedule in which each such task c_j^1 is processed during time slot* 2.

Proof. We start with the 'only if' part. Let x_i be the first literal in c_j and suppose that the task x_i is processed during time slot 7. As time slot 6 on the same machine is blocked by the task a_i, the predecessor c_j^5 of x_i is processed during time slot 5 at the latest. Similarly, the tasks c_j^6 and c_j^7 are processed during time slot 5 as the latest. Hence, the tasks c_j^2, c_j^3, and c_j^4 are processed during time slot 3 at the latest, which implies that task c_j^1 is processed during time slot 1.

The 'if'-part follows immediately from Fig. 1. The 'Moreover' part follows from the observation that we have an infinite number of processors, as none of the tasks c_j^5, c_j^6, and c_j^7 needs to be executed immediately before its successor on the same machine. □

Claim 3.4. *If there exists a truth assignment that satisfies v^* of the clauses of the* Max 3-Sat Size *instance, then there exists an optimum schedule that uses only $u+4v-v^*+1$ processors.*

Proof. Consider this truth assignment. If variable x_i is True, then schedule task $\overline{x_i}$ during time slot 7 and task x_i during time slot 8. If variable x_i is False, then schedule task x_i during time slot 7 and task $\overline{x_i}$ during time slot 8. Proposition 3.3 implies that there exists an optimum schedule such that for every satisfied clause c_j its clause task c_j^1 is processed during time slot 2.

For executing all clause tasks, we need $3v$ processors, which can also be used to process some of the variable tasks: we need no more than $u+3v$ processors to execute all clause and variable tasks. Hence, the number of used processors is determined by the number of tasks that are processed in time slot 1, which are the $u+3v$ dummy task, task s_1, and the tasks c_k^1 corresponding to clauses c_k not satisfied by the truth assignment. Hence, we need no more than $u+4v-v^*+1$ processors. □

Claim 3.5. *If there exists a schedule with makespan nine that uses $u+4v-v^*+1$ processors, then there exists a truth assignment that satisfies v^* of the clauses.*

Proof. Consider the truth setting where x_i is True if and only if its corresponding task is processed during time slot 8. Since the $u+3v$ dummy tasks and task s_1 are processed during time slot 1, at most $v-v^*$ of the clause tasks c_j^1 can be scheduled during time slot 1 and at least v^* of them must be scheduled during time slot 2. Hence, Proposition 3.3 implies that at least v^* clauses are satisfied. □

Theorem 3.6. *Rival's scheduling problem is* $\mathbf{FP}^{\mathcal{NP}[\log n]}$*-complete.*

Proof. By Lemma 3.1, the problem is contained in $\mathbf{FP}^{\mathcal{NP}[\log n]}$. Let v^* denote the maximum number of clauses that can be satisfied simultaneously by a truth assignment for the Max 3-Sat Size instance, and let m^* denote the solution to the constructed instance of Rival's problem. By Claims 3.2, 3.4, and 3.5, the equality $v^*+m^*=u+4v+1$ holds for the constructed instance. Thus, one can easily compute v^* from m^*. □

Acknowledgements

We would like to thank Frits C.R. Spieksma for helpful discussions.

References

[1] Ch. Chenier, J. Urrutia, N. Zaguia, Scheduling tasks with communication delays on parallel processors, Order 12 (1995) 213–220.
[2] J.A. Hoogeveen, J.K. Lenstra, B. Veltman, Three, four, five, six, or the complexity of scheduling with communication delays, Oper. Res. Lett. 16 (1994) 129–137.
[3] R.H. Möhring, M.W. Schäffter, Scheduling series-parallel orders subject to 0/1 communication delays, Parallel Comput. 25 (1999) 23–40.
[4] C.H. Papadimitriou, Computational Complexity, Addison-Wesley, New York, 1994.
[5] C.H. Papadimitriou, M. Yannakakis, Towards an architecture-independent analysis of parallel algorithms, SIAM J. Comput. 19 (1990) 322–328.
[6] C.H. Papadimitriou, S.K. Zachos, Two remarks on the complexity of counting, Proceedings of the Sixth GI Conference of Theoretical Computer Science, Lecture Notes in Computer Science, Vol. 145, Springer, Berlin, 1983, pp. 269–276.
[7] C. Picouleau, Etude de problèmes les systèmes distribués, Ph.D. Thesis, Univ. Pierre et Marie Curie, Paris, France, 1992.
[8] B. Veltman, Multiprocessor scheduling with communication delays, Ph.D. Thesis, CWI, Amsterdam, The Netherlands, 1993.
[9] B. Veltman, B.J. Lageweg, J.K. Lenstra, Multiprocessor scheduling with communication delays, Parallel Comput. 16 (1990) 173–182.