Minimizing the number of machines for minimum length schedules

Gerd Finke, Pierre Lemaire, Jean-Marie Proth, Maurice Queyranne

Minimizing the number of machines for minimum-length schedules

Gerd Finke [a] Pierre Lemaire [a] Jean-Marie Proth [b]
Maurice Queyranne [a]

[a] *Leibniz-IMAG*
46 avenue Félix Viallet
38 031 Grenoble cedex
France

[b] *INRIA-Lorraine*
615 rue du Jardin Botanique, B.P. 101
F-54602 Villers-lès-Nancy cedex
France

Abstract

In this paper, we present a new objective function for scheduling on parallel machines: minimizing the number of machines for schedules of minimum length. We study its complexity and we prove the NP-completeness of this problem, even if there is no precedences or for unitary execution times. We propose several polynomial algorithms for various particular cases.

Key words: parallel scheduling, complexity, minimum number of processors
PACS:

1 Introduction

In parallel scheduling, a compromise must usually be made between efficiency (the length of a schedule) and costs (the number of processors). Whenever efficiency is critical, one is just interested in minimum-length schedules and the natural question that arises is to minimize the costs, that is to find the minimum number of processors required to perform the whole schedule in the shortest possible time.

Email address: pierre.lemaire@imag.fr (Pierre Lemaire).

This is typically what happens when designing systolic networks. A *systolic network* is a particular parallel architecture made of several identical processors regularly connected that are synchronized at every clock-top. A systolic network is designed for a particular function and, thanks to its regularity and uniformity, it is a good candidate as an efficient and specialized computing device without prohibitive costs — in particular for embedded applications [1,2].

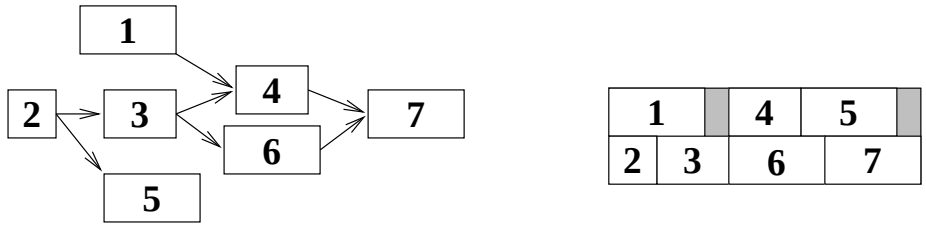
The design of a systolic network starts with the construction of the *graph of its elementary transformations*. The global function that must be computed is decomposed into elementary transformations: each of these transformations corresponds to a task that a processor will have to process, and the graph is indeed the precedence graph of these transformations. The second step is then to schedule this precedence graph: under the constraint of efficiency that the whole function is computed in the shortest possible amount of time, the cost of the systolic network — that is its number of processors — must be minimized.

In this paper, we focus on this second step. Hence we assume that the graph of the elementary transformations is given. Furthermore, in a systolic network, processors are synchronized at every clock-top and all tasks are of the same (unitary) length. We will put a particular emphasis on this special case.

In the remaining parts of this paper, we first introduce a formal definition of the problem we deal with (Section 2). We then study the complexity of this problem, in the general case (Section 3) and for unitary processing times (Section 4). In an appendix, we shall briefly discuss the practical solution of the problem.

2 Formalization

The problem of minimizing the number of machines has already been studied but in different and particular contexts (*e.g.,* with fixed processing dates and additional constraints [3] or with communication delays [4,5]). In this paper, we consider a classical scheduling environment on parallel machines: n tasks and m processors; a task T_j requires an execution time p_j and precedence constraints between tasks are represented as a directed graph *prec*. Such systems have already been extensively studied, in particular for the objective of minimizing the completion time $C_{\max}$ for a given number of machines [6,7]. We are interested, here, in the new criterion of minimizing the number of machines required to be able to schedule every task such that the whole schedule is of minimum length. More precisely, let $C = C_{\max}^{m=\infty}$ be the minimum length for a schedule using an infinite number of machines; this length C is indeed the length of a critical (longest) path in the precedence graph *prec*. We are looking



(a) (b)

Fig. 1. (a) Seven tasks with precedences and (b) a minimum-length schedule using only 2 machines.

for m^*, the smallest possible number of machines m that allows to schedule all tasks within C. Figure 1 shows an example of such a schedule.

Extending the 3-field notation [6], we shall denote this problem $P|prec, C_{\max}^{m=\infty}|m$. The problem $P|prec, C_{\max}^{m=\infty}|m$ is obviously very close to, but different from, the classical $P|prec|C_{\max}$: here, the schedule length is not part of the objective but appears as a constraint. Formally the decision problem $P|prec, C_{\max}^{m=\infty}|m$ is:

Problem $P|prec, C_{\max}^{m=\infty}|m$ **:**
instance : a set T of tasks ; a processing time p_j for each task T_j ; a partial order *prec* on the tasks ; a bound m.
question : is there a schedule of T, of minimum length, that respects the precedences *prec* and uses at most m processors?

In this formulation, the fact that a solution must be of minimum length (and not just shorter than a given arbitrary bound) is essential and, in particular, distinguishes $P|prec, C_{\max}^{m=\infty}|m$ from $P|prec|C_{\max}$. However, it must be noted that this particular constraint does not take the problem out of NP, since computing the length of a longest path in an acyclic network can be done in polynomial time [8].

3 Arbitrary processing times

In the case of arbitrary processing times, the problem is NP-complete, even if there are no precedences. In this case, the critical path is simply the longest task, of length $C = \max_j p_j$. This task will be assigned to a processor, and then all remaining tasks (equivalent to objects) will have to be scheduled on the remaining $m-1$ processors (equivalent to bins of height C). This is indeed similar to the Bin-Packing problem [9], and hence similar results hold:

Proposition 1 *$P|C_{\max}^{m=\infty}|m$ is NP-complete in the strong sense.*

Proposition 2 *$P|C_{\max}^{m=\infty}|m$ is not approximable within less than 4/3. More precisely, if $P \neq NP$ then, for every approximation algorithm A solving $P|C_{\max}^{m=\infty}|m$, there is an instance I such that:*

$$m_A(I) \geq \frac{4}{3} m^*(I)$$

where $m_A(I)$ is the value of the solution provided by A, and $m^*(I)$ is the optimal value.

PROOF. We shall use the classical trick that if an algorithm A provides, for every instance I, a solution such that: $m_A(I) < \frac{4}{3} m^*(I)$, then it solves the NP-complete problem BIPARTITION.

Consider an instance of BIPARTITION: n integers $a_1, a_2, \ldots, a_n$. Let $B = \sum_{i=1}^{n} a_i$. The question is: is there a sub-set of indices $S \subset N = \{1, 2, \ldots, n\}$ such that: $\sum_{i \in S} a_i = \sum_{i \in N \setminus S} a_i = \frac{B}{2}$? For the given instance, we define the following instance for $P|C_{\max}^{m=\infty}|m$: $n+1$ tasks, the n first ones having the a_i as processing times, and the last one having length $B/2$. There are no precedences. With such a transformation, there is a schedule on 3 processors if and only if BIPARTITION has a "yes" answer; otherwise, at least 4 processors are required. An algorithm A with the guaranty that $m_A(I) < \frac{4}{3} m^*(I)$ would optimally solve such an instance and so would solve BIPARTITION. Hence it does not exist unless $P = NP$.

In the case of precedences, a stronger bound holds:

Proposition 3 *$P|prec, C_{\max}^{m=\infty}|m$ is not approximable within less than 3/2. More precisely, if $P \neq NP$ then, for every approximation algorithm A solving $P|prec, C_{\max}^{m=\infty}|m$, there is an instance I such that:*

$$m_A(I) \geq \frac{3}{2} m^*(I)$$

where $m_A(I)$ is the value of the solution provided by A, and $m^*(I)$ is the optimal value.

PROOF. The proof is similar to the preceding one. To the n integers $a_1, a_2, \ldots, a_n$ and the bound B of a BIPARTITION instance corresponds an instance of $P|prec, C_{\max}^{m=\infty}|m$ with $n+4$ tasks such that: $p_i = a_i, i = 1, 2, \ldots, n$ and $p_{n+1} = p_{n+4} = B/2$ and $p_{n+2} = p_{n+3} = 1$. Furthermore T_{n+1} precedes T_{n+2} and T_{n+3}, which both precede T_{n+4}; there are no other precedences. For such an instance, there is a solution to $P|prec, C_{\max}^{m=\infty}|m$ using only 2 machines if and only if the corresponding BIPARTITION is possible (see Figure 2). Hence, an algorithm with the guaranty that $m_A(I) < \frac{3}{2} m^*(I)$ would solve BIPARTITION, and hence does not exist if $P \neq NP$.

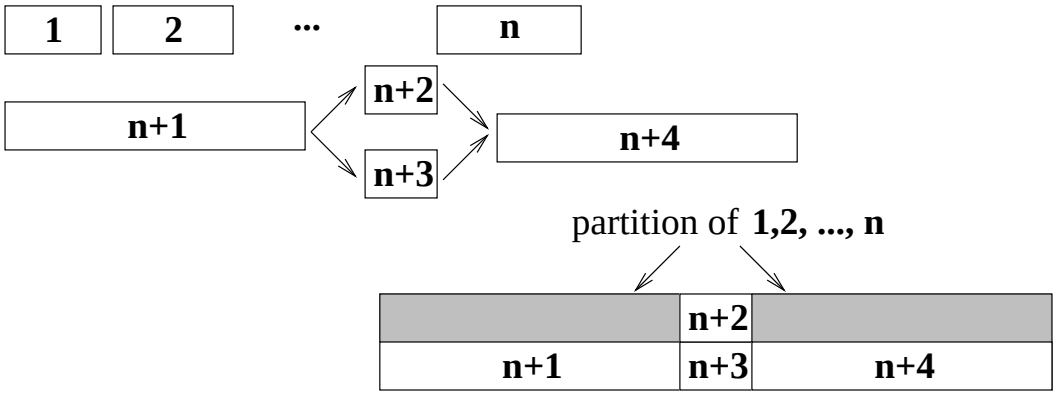


Fig. 2. An instance of $P|prec, C_{\max}^{m=\infty}|m$ corresponding to an instance of BIPARTITION.

A simple but useful lower bound on the optimal number of machines exists:

$$m^* \geq \left\lceil \frac{\sum_{j=1}^{n} p_j}{C} \right\rceil.$$

This bound holds whatever the precedences are; in the case of unitary tasks, it becomes $m^* \geq \left\lceil \frac{n}{C} \right\rceil$. Furthermore, it can be improved if the precedence graph is made of several serial components:

Proposition 4 *A lower bound for $P|prec, C_{\max}^{m=\infty}|m$ is:*

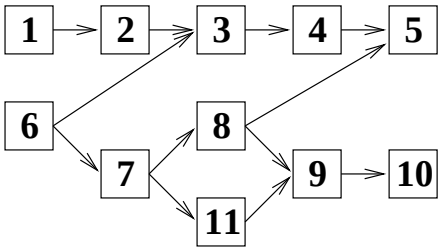
$$m^* \geq \max_{i=1..q} \left\lceil \frac{p(S_i)}{C_i} \right\rceil$$

where $\{S_1, S_2, \ldots, S_q\}$ is a serial decomposition of prec (every task of S_i precedes every task of S_{i+1}), $p(S_i)$ is the sum of the execution times over the tasks of S_i and C_i is the length of a longest path in S_i.

The principle of the serial decomposition is indeed a general phenomenon for this problem, since the components of such a decomposition are independent for the considered objective. Hence, any result for a particular class of graphs can directly be extended to the case of serial compositions of such graphs.

With arbitrary processing times, a special case is polynomially solvable where every task is part of a critical path. In this case, processing dates are fixed and it is enough to compute the instant when the greatest number of tasks have to be performed simultaneously.

Proposition 5 *The special case of $P|prec, C_{\max}^{m=\infty}|m$ where every task belongs to a critical path of prec can be solved in polynomial time $O(|prec| + n \ln n)$, and even $O(|prec| + n)$ if the processing times are unitary.*



L_1	L_2	L_3	L_4	L_5
1	**2**	**3**	**4**	**5**
6	**7**	**8**	**9**	**10**
		11		

Fig. 3. A solution if every task belongs to a critical path and $p_j = 1$.

PROOF. A possible solution is built as follows. First, compute for each task its starting and finishing dates (which are fixed): this can be done in time linear in the number of edges and the number of tasks, that is $O(|prec| + n)$ in the worst case, for an acyclic graph. Then, sort the list of these $2n$ dates in non-decreasing order (this takes $O(n \ln n)$) and traverse it to determine how many tasks are executed during each of the $2n$ intervals of time: the maximum value is m^*, the value one is looking for. The whole procedure runs in time $O(|prec| + n \ln n)$.

In the case of unitary tasks, we call L_i the set of the tasks executed between time $i-1$ and i, defined for $i = 1, \ldots, C \leq n$. A single traversal of the graph *prec* is enough to compute, instead of the execution dates of each task, the cardinality of each L_i. The optimal value is then found in $O(n)$ as $m^* = \max_i |L_i|$. The whole procedure runs in time $O(|prec| + n)$ (see Figure 3).

4 Unitary processing times (and preemption)

We consider now the case where every task has unitary processing time: $P|prec, p_j = 1, C_{\max}^{m=\infty}|m$. This problem is closely linked to the preemptive case.

First, remark that, in the decision problem, the number of machines m can be bounded by the number of tasks: the size of an instance is determined by the size of the precedence graph, $|prec| = O(n^2)$, and $P|prec, p_j = 1, C_{\max}^{m=\infty}|m$ is not a numerical problem. Hence, the notion of NP-completeness and NP-completeness in the strong sense are equivalent.

The related problem $P|prec, p_j = 1|C_{\max}$ is NP-complete in the general case. For a fixed number of processors ($Pm|prec, p_j = 1|C_{\max}$), the status is open for $m \geq 3$ [6], but it is solvable in polynomial time if there are only 2 processors [10]. Hence it is not surprising that $P|prec, p_j = 1, C_{\max}^{m=\infty}|m$, even with unitary tasks, cannot be solved optimally with rules such as processing a task as early or as late as possible (see Figure 4): indeed, this problem is NP-complete.

Proposition 6 *$P|prec, p_j = 1, C_{\max}^{m=\infty}|m$ is NP-complete.*

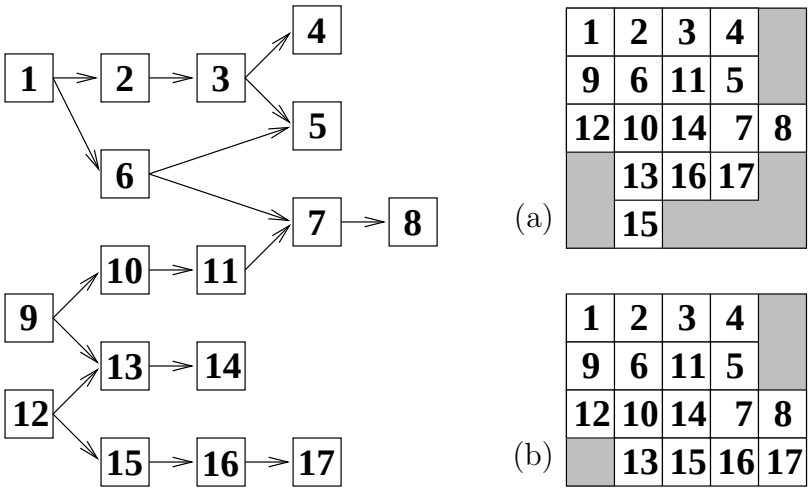


Fig. 4. Seventeen unitary tasks with precedences and their (a) earliest time schedule and (b) optimal schedule.

PROOF. We shall reduce $P|prec, p_j = 1|C_{\max}$ to $P|prec, p_j = 1, C_{\max}^{m=\infty}|m$. Let I be an instance of $P|prec, p_j = 1|C_{\max}$: a set of n tasks of unitary length, with a precedence graph $prec$, a bound C on the completion time and a bound m on the number of processors. From this instance, we define an instance I' for $P|prec, p_j = 1, C_{\max}^{m=\infty}|m$ as follows: a number of processors $m' = m + 1$, a set of $n + C$ tasks and a new precedence graph $prec'$ composed by $prec$ and an additional chain between the additional C dummy tasks.

Without loss of generality, we may assume that $C \leq n$: hence, I' is a polynomial transformation of I. Furthermore, I has a "yes" answer if and only if I' has one too: if C is shorter than the length of a critical path, then there is no solution for I, nor for I'. If C is at least the length of a critical path of $prec$ then the C dummy tasks form a critical path in $prec'$, and we may assume that these tasks are all performed by the $m + 1^{th}$ processor. This leaves the n real tasks of I' to be scheduled on m processors, which is possible if and only if I has a solution.

As a conclusion, $P|prec, p_j = 1|C_{\max}$ reduces to $P|prec, p_j = 1, C_{\max}^{m=\infty}|m$; since the former is NP-complete [6,7], so is the latter.

The preemptive case $P|prec, pmtn, C_{\max}^{m=\infty}|m$ is also NP-complete. Indeed, the same proof holds, using the NP-complete problem $P|prec, pmtn, p_j = 1|C_{\max}$ [6,7].

For some particular graphs, polynomial results on $Pm|prec,\ p_j = 1|C_{\max}$ can be adapted to solve $P|prec, p_j = 1, C_{\max}^{m=\infty}|m$. In particular, Hu proposed an $O(n)$ algorithm for trees [11,12] which, in our case, leads to the following result.

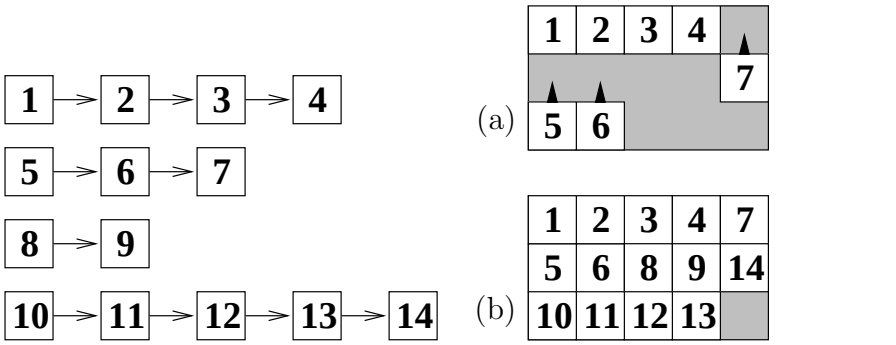


Fig. 5. Solution of $P|chains, p_j = 1, C_{\max}^{m=\infty}|m$; (a) second iteration; (b) optimal solution.

Proposition 7 *The decision problem* $P|tree,\ p_j = 1, C_{\max}^{m=\infty}|m$ *can be solved in time* $O(n)$*. The optimization problem* $P|tree, p_j = 1, C_{\max}^{m=\infty}|m$ *can be solved in time* $O(n \ln(n))$*.*

PROOF. Hu's algorithm [11] solves optimally $P|tree, p_j = 1|C_{\max}$ in time $O(n)$. Then the value has just to be compared to the length of a critical path (that can be computed in time $O(n)$ for a tree) to solve the decision problem. To find the optimal value m^*, the smallest number of processors such that Hu's algorithm finds a solution of length C, a binary search over $\{1, 2, \ldots, n\}$ can be done, which provides an optimal procedure for $P|tree, p_j = 1, C_{\max}^{m=\infty}|m$ running in time $O(n \ln(n))$.

A similar approach is valid for other kinds of graphs. In particular, the case $Pm|opposing\ forest, p_j = 1|C_{\max}$ has been solved, first by Garey, Johnson, Tarjan and Yannakakis for $m = 3$ [13], and then by Dolev and Warmuth [14]. However, their $O(n^{2m-2} \ln(n))$ algorithm is exponential in the number of machines and hence implies a non-polynomial solution when applied to $P|opposing forest, p_j = 1, C_{\max}^{m=\infty}|m$.

For the preemptive case, Muntz and Coffman [15] proposed an $O(n^2)$ algorithm for $P|pmtn, forest|C_{\max}$. Straightforwardly:

Proposition 8 *The decision problem* $P|pmtn, forest, C_{\max}^{m=\infty}|m$ *can be solved in time* $O(n^2)$*. The optimization problem* $P|pmtn, forest, C_{\max}^{m=\infty}|m$ *can be solved in time* $O(n^2 \ln(n))$*.*

If, instead of a tree, the precedence graph is a set of chains, a variant of McNaughton's algorithm [16] provides a simple solution, as shown in Figure 5.

Proposition 9 *An optimal schedule for* $P|chains, p_j = 1, C_{\max}^{m=\infty}|m$ *can be*

built in time $O(n)$. The optimal value is:

$$m^* = \left\lceil \frac{n}{C} \right\rceil .$$

PROOF. The announced value is a lower bound for m^* (proposition 4). To reach it, one can apply the following procedure: for each chain, schedule as many as possible of the end of the chain on the last processor; if the chain cannot be completely scheduled, add a new processor and schedule on it, as early as possible, the extra tasks.

The arguments in this proof are also valid for arbitrary processing times if preemptions are allowed. As a consequence, the algorithm above applies to $P|chains, pmtn, C_{\max}^{m=\infty}|m$.

5 Conclusions

We have presented a new objective for the classical parallel scheduling environment with precedences: to minimize the number of processors required for a minimum-length schedule. This new objective is practical and arises in industrial applications.

We have proved that the problem is NP-complete even if there are no precedences or for unitary processing times. We have proposed several polynomial procedures for particular cases.

For a practical solution one may use constraint programming. As we briefly present in the appendix A, even a naive implementation is enough to solve large random instances. However, further developments are required to design a specific constraint programming system that could handle the hardest instances. Some work on better lower and upper bounds would be valuable too.

A Towards a practical solution using constraint programming

One may remark that a task usually falls into one of two main configurations. On one hand, it may be in conflict with other tasks for a same time-window and any decision concerning one task implies quickly the decisions for the other tasks, without much freedom. On the other hand, it may be rather free, in a time-zone without many other tasks, and its presence/absence does not really

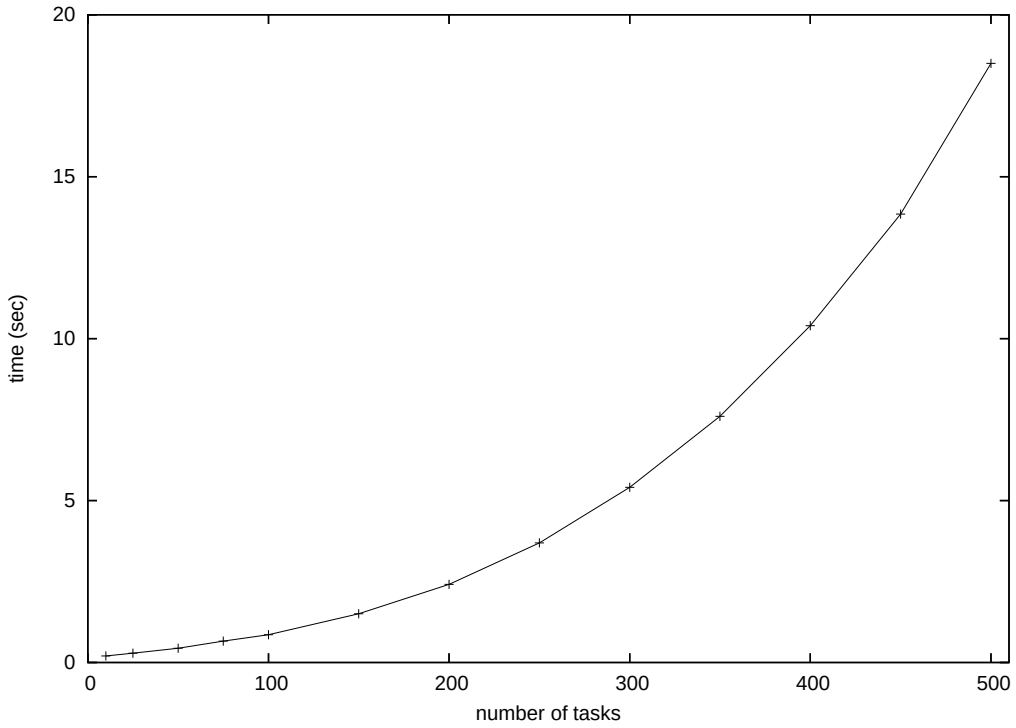


Fig. A.1. Computational times for solving random instances (average for 10 instances) on a Pentium IV with 256Mo RAM, running under Linux/Debian.

matter for the considered objective. Both cases are appropriate for constraint programming (this can be seen as a natural extension to real instances of the algorithm of Proposition 5).

We tried this approach for systolic networks (*i.e.,* with unitary tasks), using a generic finite domain solver [17]. Tested on randomly generated precedence graphs (a precedence between two tasks T_i and T_j, $i < j$, exists with given probability), this procedure performs well. As shown in Figure A.1, and in spite of an exponentially growing running-time, instances with several hundreds of tasks are solved optimally in a few seconds on a standard PC.

References

[1] H. T. Kung, C. E. Leiserson, Systolic arrays for VLSI, in: C. A. Mead, L. A. Conway (Eds.), Introduction to VLSI systems, Addison-Wesley, 1980.

[2] N. Ling, M. A. Bayoumi, Specification and verification of systolic arrays, World Scientific Publishing, 1999.

[3] L. G. Kroon, M. Salomon, L. N. van Wassenhove, Exact and approximation algorithms for the tactical fixed interval scheduling problem, Operations Research 45 (4) (1997) 624–638.

[4] A. Moukrim, On the minimum number of processors for scheduling problems with communication delays, Annals of Operations Research 86 (1999) 455–472.

[5] A. Moukrim, Upper bound on the number of processors for scheduling with interprocessor communication delays, Mathematical Methods of Operations Research 52 (1) (2000) 99–113.

[6] J. Błażewicz, K. H. Ecker, E. Pesch, G. Schmidt, J. Węglarz, Scheduling Computer and Manufacturing Processes, Springer-Verlag, 1996.

[7] B. Chen, C. N. Potts, G. J. Woeginger, A Review of Machine Scheduling: Complexity, Algorithms and Approximability, in: D.-Z. Du, P. Pardalos (Eds.), Handbook of Combinatorial Optimization, Kluwer Academic Publishers, 1998, pp. 21–169.

[8] M. R. Garey, D. S. Johnson, Computers and Intractability (A Guide to the Theory of NP-Completeness), W.H. Freeman And Company, 1979.

[9] E. G. Coffman Jr, M. R. Garey, D. S. Johnson, Approximation Algorithms for Bin-Packing: a survey, in: D. S. Hochbaum (Ed.), Approximation Algorithms for NP-hard Problems, PWS Publishing Company, 1997, Ch. 2, pp. 46–93.

[10] E. G. Coffman Jr, R. L. Graham, Optimal scheduling for two-processor systems, Acta Informatica 1 (1972) 200–213.

[11] T. Hu, Parallel Sequencing and Assembly Line Problems, Operations Research 9 (1961) 841–848.

[12] J. A. McHugh, Hu's Precedence Tree Scheduling Algorithm: A Simple Proof, Naval Research Logistics Quarterly 31 (1984) 409–411.

[13] M. R. Garey, D. S. Johnson, R. E. Tarjan, M. Yannakakis, Scheduling opposing forests, SIAM Journal of Algebraic Discrete Methods 4 (1983) 72–93.

[14] D. Dolev, M. K. Warmuth, Scheduling flat graphs, SIAM Journal of Computing 14 (1985) 638–657.

[15] R. R. Muntz, E. G. Coffman, Jr, Preemptive scheduling of real time tasks on multiprocessor systems, Journal of the Association for Computing Machinery 17 (1970) 324–338.

[16] R. McNaughton, Scheduling with deadlines and loss functions, Management Sciences 6 (1959) 1–12.

[17] D. Diaz, The GNU Prolog compiler (`gprolog`), http://www.gnu.org/software/gprolog/gprolog.html and http://pauillac.inria.fr/∼diaz/gnu-prolog/.