Scheduling in Parallel Machines with Small/Large Communication Delays

Renan Pires[1] , **Philippe Navaux**[1] , **Jayme Szwarcfiter**[2] , **Rosiane de Freitas**[3]
[1]Federal University of Rio Grande do Sul, Institute of Informatics
Porto Alegre, Brazil
[2]Federal University of Rio de Janeiro, COPPE, Inst. Mat. and NCE
Rio de Janeiro, Brazil
[3]Federal University of Amazonas, Institute of Computing
Manaus, Brazil
{rfpires,navaux}@inf.ufrgs.br , jayme@nce.ufrj.br , rosiane@icomp.ufam.edu.br

ABSTRACT

This work presents computational complexity results involving scheduling in parallel systems subject to very small or too large communication delays. We propose polynomial time algorithms for two cases, both when a sufficient number of machines is available. For the first algorithm, the precedence relations among tasks are restricted to a descendant tree. For the second one, we show that if task duplications are allowed, it is possible to schedule subject to arbitrary precedence relations. Moreover, we show that if the number of machines is given at execution time, the problem becomes NP-hard even if a precedence relations consist of a set of chains. This also occurs when a fixed number of only two processors are available and general precedence relations are allowed. These results are an important contribution in scheduling theory to apply in parallel and distributed systems (cluster, grid, cloud/global computing) in a large range of real world problems including industrial production, airlines planning, computer systems, and so on.

Keywords: algorithms, communication delay, computational complexity, direct acyclic graph, parallel systems, scheduling theory.

Main areas: OC - Combinatorial Optimization; TAG - Theory/Algorithms in Graphs.

1. Introduction

Scheduling constitutes a major and relevant class of problems in operations research. Applications can be found in industrial production, airlines planning, or computer systems. One of the most common aim of a scheduling problem is to minimize the processing time of the last task to be scheduled (the *makespan*), subject to some constraints.

In many applications a collection of tasks may be required to be processed on the same machine. This restriction is possibly found on several cases. For example, in an industrial production process, some materials produced in a certain station may not be allowed to be transported to others substations, being necessarily processed on the current one. Another application consists of task allocation routines in multi-processing systems. Some computer routines might generate a large quantity of data that is used by some other task. If the delay imposed by transferring the data from a processing unit to another one is too large, the scheduling of these tasks in another processor could become unfeasible. In fact, this type of constraint can represent task scheduling under communication delays that are either negligible or infinitely large.

This paper presents results subjected to this communication delays when unit time task are considered and some precedence ordering is imposed. We show that, when sufficient number of processors are available, the problem is polynomial if the precedence relation are restricted to an *out-tree* or when computing the same task in different processors is allowed. Also, we present NP-hardness proofs for two cases, the first when the number of processor is arbitrary and the second when any precedence relation is permitted.

Scheduling theory employs a usual notation, adopted throughout this paper [8] [13].

This text is organized as follows. In Section 2, we present definitions and concepts relevant to the work. The proposed algorithms and NP-hardness proof are given in Section 3. Finally, in Section 4, we present the concluding remarks.

2. The Scheduling Problem

The scheduling problem considered in this work involves a set of identical parallel processors $P = \{P_1, ..., P_m\}$, which may be limited by a number *m* (a fixed one or given in an problem instance), or unlimited (alternatively, we can suppose to be as large as the number of available tasks). These processors are used to execute a set of tasks or jobs $J = \{J_1, ..., J_n\}$. Each job J_j ($j \in \{1, ..., n\}$) has a unit processing time e.g. [2], [3]. A job can be processed in only one processor or can have several copies placed in different processors, this possibility is referenced as tasks duplication. Each processor can process at most one job at a time, and each copy of a job can be processed by at most one machine at a time. The tasks processing ordering is restricted by a precedence relation $\prec_R$, represented by a directed graph $G = (J, E)$, where *E* is a ordered pair of tasks in *J*. A *path* is a sequence of edges in *E* for which every consecutive edges, e_i succeeded by e_{i+1}, must be so that $e_i = (u, v)$ and $e_{i+1} = (v, w)$. The set of vertices for which exists a path to some vertex v is defined as the predecessors of v. The set of vertices for which exists path from v to them is defined as the descendants of v. The immediately descendants (respectively predecessors) of v are the set of vertices u for which exists $(v, u) \in E$ (respectively $(u, v) \in E$). The precedence relation for the considered problems can be an arbitrary one, an *out-tree*, that is when every task has at most one immediately predecessor which may be referenced as the parent task, or yet a set of chains, where every task has at most one immediately predecessor and at most one immediately descendant. At last, each edge in E is associated to a communication delay, through a function $C : E \to \{0, \infty\}$, which can be either negligible or infinitely large (again, alternatively, we can suppose to be as large as the number of tasks).

2.1. Previous Works

The problem of finding the minimum makespan has been extensively studied. It is known to be NP-hard even if only two processors are considered and having no precedence relation ($P_2||C_{max}$) [7]. Also, although this problem admits a pseudo-polynomial algorithm [14], it becomes strongly NP-hard if a special precedence relation consisting of a set of chains is imposed ($P_2|chains;|C_{max}$) [4]. These results indicate that in order to obtain polynomial-time algorithms, one must restrict the problem by some other constraint. An important and well-studied additional constraint concerns the tasks processing time. When only unit execution time tasks are considered, a polynomial algorithm for arbitrary precedence relation ($P_2|prec; p_j = 1|C_{max}$) is known [6]. However, this problem becomes NP-hard if the unit processing time constraints are relaxed as to also admit two unit processing time per jobs ($P_2|prec; p_j \in \{1, 2\}|C_{max}$) [16].

Communication delays have been introduced around the earlies eighties, for modeling the cost of transferring data from one processor to another. More precisely, communication delays are assigned to pairs of tasks for which some precedence relation is described. It imposes that for each such pairs, if the first task is processed on a different processor than the second one, it may only be processed after the specified delay time. Thus, when such communication delays are considered, a scheduling problem over a unlimited number of processors ($P_\infty|prec;|C_{max}$) turns from easy to NP-hard, even if the precedence constraints are restricted to an *out-tree* ($P_\infty|$ *out-tree;* $c_{i,j}|C_{max}$) [1]. Although, by restricting the precedence relation to a tree, it is possible to find the minimum *makespan* in *pseudo-polynomial* time [5]. The problem remains NP-hard even when communication delays are also imposed to be fixed to a unit time. In fact, for this problem ($P_\infty|prec; c = 1; p_j = 1|C_{max}$) the total schedule length is to be limited to five units of time, otherwise no polynomial time algorithm is possible (except if $\mathcal{P} = \mathcal{NP}$) [9]. Also, by adding delays, it may be preferable to process the same task on different processors than to have to wait for the data to be transferred. Unfortunately, the problem remains NP-hard even if along with this assumption all tasks are imposed to be a unit execution time

$(P_\infty|prec; c; dup; p_j = 1|C_{max})$ [12], again, it has been shown that this problem admits pseudo-polynomial algorithm on the communication delay [11].

3. Proposed Results

In this section, we present our results involving the scheduling problems with large communication delays described previously, organized in two parts. The first one contains the polynomial time results, where we explain the proposed algorithms for two special cases. In the second part, the NP-hardness computational complexity results are presented for two other cases.

3.1. Polynomial Time Results

We determined the polynomial complexity of two scheduling problems that consider an unlimited number of processor

3.1.1. $P_\infty|$***out-tree***$, c_{i,j} \in \{0, \infty\}, p_j = 1|C_{max}$

In this subsection, we present the polynomial complexity proof of the scheduling problem $P_\infty|$ *out-tree* $, c_{i,j} \in \{0, \infty\}, p_j = 1|C_{max}$, involving an unlimited number of parallel processors, the minimum makespan optimization criteria, and a set of tasks subject to a unit processing time, out-tree precedence relations, and communication delays restricted to zero or infinity. Thus, first, we give the following definition:

Definition 1**:** A *inf-connected-tree* is the maximal set of vertices connected by edges associated to infinitely large delays. For some vertex v, we define *inf-connected-tree(v)* to be the *inf-connected-tree* that it belongs to.

If the precedence relation is restricted to be an out-tree and an unlimited number of processors are available, it is possible to recursively solve each negligible-connected subtree, that is, a maximal subtree rooted at a vertex v whose parent belongs to the same *inf-connected-tree* of the tree root, but v itself does not (see Figure 1). As all vertices of the same *inf-connected-tree* must be processed on the same processor, they are to be schedule to form a total ordering. Also, there are no reasons for them not to be processed consecutively, so if t_r is the slot allocated to the root, then the slots allocated to the vertices on the same *inf-connected-tree* are into $\{t_r, t_r + 1, ..., t_r + |T_r|\}$. We define $order$ and $priority$ as follows:

Definition 2**:** The *order* of a vertex v related by a scheduling S, $order_S(v)$, is the number of vertices from the same *inf-connected-tree* that were scheduled before v in S, plus one.

Definition 3**:** The *priority* of a vertex v related by a scheduling S, $priority_S(v)$, is defined as the maximum makespan among the *negligible-*

connected-subtrees which roots are descendant of v. If no such subtrees exists, the $priority_S(v) = 0$ (See Figure 1).

By such definitions, we verify the following lemma.

Lemma 1**:** The total schedule time, t_S, for some given scheduling S of a tree, T_r, rooted at some vertex r is equal to $\max\{order_S(v) + priority_S(v)\}, \forall v \in$ *inf-connected-tree(r)*.

Each v in *inf-connected-tree(r)* are clearly scheduled no later than $\max\{$ $order_S(u)$ $+priority_S(u)\}$, $\forall u \in$ *inf-connected-tree(r)*, as each such vertex is schedule precisely in $slot\ order_S(v)$.

Also, let v in *inf-connected-tree(r)* have some *negligible-connected-subtree*. All vertices in such subtree are schedule no latter than $order_S(v) + priority_S(v)$, as v is schedule on $slot\ order_S(v)$ and, by definition, the $priority_S(v)$ is greater or equal the $makespan$ of any *negligible-connected-subtree* descendant of v.

At last, let v be one vertex which $order_S(v) + priority_S(v)$ is maximum among the vertices in *inf-connected-tree(r)*. So, v has no immediately descendant vertex u in *inf-connected-tree(r)* such that $priority_S(v) = priority_S(u)$, because certainly $order_S(u) > order_S(v)$ and this would contradict the fact that $order_S(v) + priority_S(v)$ is maximal. If no such immediately descendant exists, either $priority_S(v) = 0$ or there must be an immediately descendant that is a root of a *negligible-connected-subtree* for which $makespan$ scheduled by S equal to $priority_S(v)$. So after v is scheduled, the processing from all *negligible-connected-subtree* which roots are immediately descendants o v finish its processing precisely on $priority_S(v)$ units of time. So, the schedule time must be at least $\max\{order_S(v) + priority_S(v)\}$.

If the *inf-connected-tree* from the root has no *negligible-connected-subtree*, the minimum schedule time is the number of vertices of the tree, as all vertices must be processed on the same processor. If the tree contains negligible-connected subtrees, in order to execute all tasks in a minimum time, a natural strategy is to start the *negligible-connected-subtrees* as soon as possible. The intuitive reasoning for this is that the computation of distinct *negligible-connected-subtrees* may occur in parallel, while the computation of vertices from the same *inf-connected-tree* cannot. Also, an argument can be made in favor to prioritize the *negligible-connected-subtree* that needs more time to complete its processing. In fact, the following theorem shows that an optimal schedule is achieved if among all ready task from the same *inf-connected-tree* we choose a vertex with maximum $priority$.

Theorem 1**:** A schedule that among all ready tasks choose those with maximum $priority$ is optimal. Let S be obtained by iteratively choosing vertices of

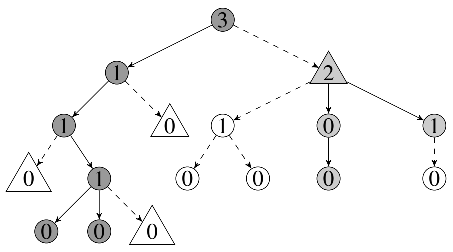

Figure 1. The figure represents an *out-tree* instance. Dashed lines represent the negligible communication delays and continuous lines correspond to the infinitely large delays. Two *inf-connected-trees* are shown, one composed by the dark grey vertices and the other by light grey ones. The triangle nodes are roots of negligible subtrees. Also inside each vertex is indicated its priority.

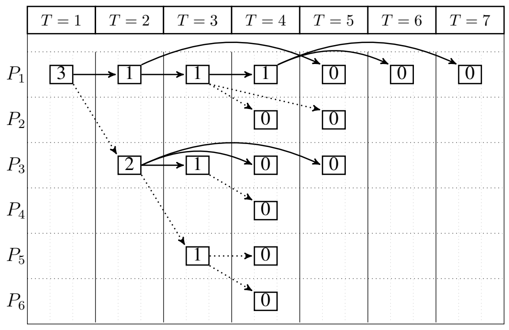

Figure 2. The figure represents an optimal schedule for the *out-tree* shown on Figure 1.

maximal $priority$ to be scheduled first between those in the same *inf-connected-tree*. Also, let t_S be its completion time.

If the tree that has no *negligible-connected-subtrees*, all vertices ought to be computed on the same processor. Therefore, any S is optimal. Otherwise, suppose the tree contains *negligible-connected-subtrees*. By induction assume them to be scheduled at its minimal scheduled time. So, by Lemma 1, $t_S = \max\{order_S(v) + priority_S(v)\}$, $\forall v \in$ *inf-connected-tree(r)*. For all v, $priority_S(v)$ is set the minimal possible value among any valid schedule. Also, the orders of the vertices are $\{$1,2, ..., $|$*inf-connected-tree(r)*$|$ $\}$. So, in order to minimize $\max\{order_S(v) + priority_S(v)\}$, one must add the maximum $priorities$ with the minimum $orders$. Clearly, S schedules in such way.

We claim this theorem implies the schedule to be optimal.

3.1.2. $P_\infty|$***prec***$, c_{i,j} \in \{0, \infty\}, dup, p_j = 1|C_{max}$

In this subsection, we present the polynomial complexity proof of a similar scheduling problem from the previous subsection. This problem admits an arbitrary prece-

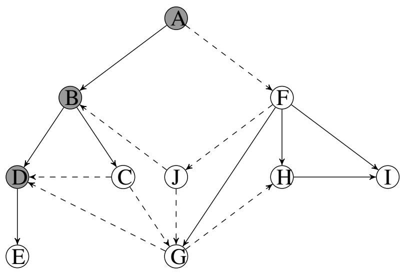

Figure 3. The figure represents a direct acyclic graph. Again, dashed lines represent negligible communication delays while continuous lines represent infinitely large ones. In grey are the *inf-connected-predecessors* of the vertex E.

dence constraint. Also, a task may be executed in different processors (duplication tasks). We first define the notation employed.

Definition 4**:** The *inf-connected-predecessors* of some vertex v, *inf-connected-predecessors*(v), is the maximum set of vertices connected by edges associated to infinitely large communication delays which v belongs.

By allowing duplication tasks, it is possible to schedule same task to be processed in several processors. As the number of processors is unlimited, it is feasible to designate a processor that would be dedicated to process a copy of each task at its minimum scheduling time. The purpose is not to delay the schedule of the *negligible-connected-descendants*. Although, it is necessary, in order to process any copy of a task, to have a copy from each *inf-connected-predecessors* of this vertex processed at the same processor. The schedule of these tasks can proceed recursively, according to the order of minimum scheduled times (see Figure 3).

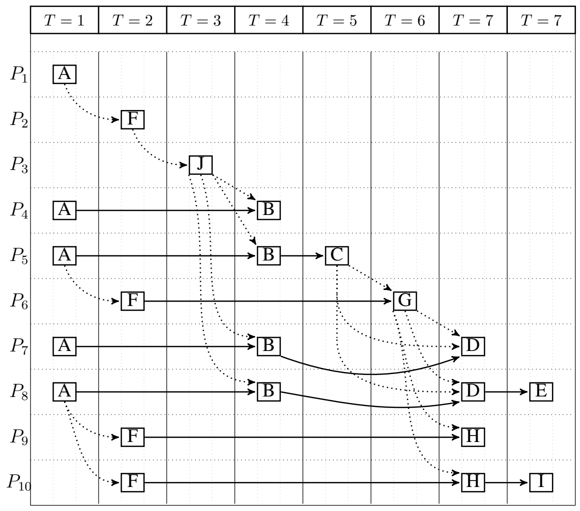

Figure 4. The figure represents the corresponding schedule from the direct acyclic graph on Figure 3.

Theorem 2**:** The process described above obtains an optimal scheduling.

A vertex that has no predecessor is clearly scheduled at its minimum schedule time on its assigned processor. For the others, suppose that every predecessor has been scheduled at its minimum schedule time on some other processor. Then, the set of vertices in *inf-connected-predecessors* which have minimum schedule time is trivially scheduled in the optimality. Recursively, the next set of minimum schedule time predecessors will be scheduled on the first empties slots after their minimum schedule time.

We claim this theorem implies the schedule to be optimal.

3.2. *NP-hardness* Results

In this section, we describe NP-hardness cases of the problem of scheduling unit tasks with large communication delays.

3.2.1. $P|\textbf{\textit{chains}}, c_{i,j} \in \{0, \infty\}, p_j = 1|C_{max}$

The problem of scheduling under these communication delays on a limited number of processors indicated at the instance of the problem is closely related to the problem studied by Sevastyanov when task preemption are considered along with non-negligible migration delays [15]. We show our problem is NP-hard by reduction from the *3-Partition Problem*, which is known to be strongly NP-hard. In fact, we consider the decision version of both problems, defined as follow:

3-PARTITION PROBLEM
In: Given an instance, $\langle A \rangle$, of a set of positive integers, such that $|A| = 3m$ for some m and for each $a_i \in A$ $\frac{Sum}{4} < a_j < \frac{Sum}{2}$ where Sum is the sum of elements of A.
Out: To decide whether it is possible for A to be partitioned in m subsets such that the sum of the elements of every subset is equal.

ARBITRARY PROCESSOR PROBLEM
In: An instance is described as $\langle Chains, C, P, t \rangle$, such that $Chains$ is a set of $chains$, C corresponds to the communication delays assigned to par of consecutive vertices in some chain the value 0 or ∞, P is the number of available processors, and t is an integer.
Out: To decide whether there exists a schedule whose makespan is $\leq$ t.

Theorem 3**:** The ARBITRARY PROCESSOR PROBLEM is in NP-complete.

The problem is clearly in *NP* by using the scheduled itself as a certificate.

Given any instance of the 3-PARTITION PROBLEM let us construct an instance of the ARBITRARY PROCESSOR PROBLEM by the following manner:

For every element $a_i \in A$, add a $chain_i \in Chains$ containing a_i jobs and such that $C(J_j, J_k) = \infty$ for every consecutive tasks in $chain_i$. Also, define $P = m$ and $t = \frac{Sum}{m}$. Recall that Sum is $\sum\limits_{a_i \in A} a_i$.

Let $chain_i$ be any of the considered chains. As every task of $chain_i$ must be processed in the same processor due to the communication delays, a valid scheduling naturally creates a partition of the chains into the processors. The partition of A induced by the association of each elements a_i with $chain_i$ is a valid partition for the 3-PARTITION PROBLEM. This is because the total number of $slots$ up to t equals to $P \times t$ which is the total of tasks. This implies that to every processor is assigned $\frac{Sum}{m}$ tasks from accurately three distinct chains.

Also, if there is some valid partition for the 3-PARTITION PROBLEM, this partition can be used to schedule the chains in $Chains$ by processing on the same processors chains associated with elements from the same partition. Clearly, this scheduling respects the constraints imposed by C and the time t.

3.2.2. $P_2|\textbf{\textit{prec}}, c_{i,j} \in \{0, \infty\}, p_j = 1|C_{max}$

When an arbitrary precedence relations can be imposed, scheduling unit tasks under large communication delay constraints becomes a NP-hard problem for any number of processors other than one. Moreover, the decision problem corresponding to this problem is NP-complete. This is consequence of the NP-completeness of a problem studied by Jansen [10]:

SCHEDULING TYPED TASK
In: An instance is given by $\langle Chains, Pred, t\rangle$, such that $Chains$ is a set of chains, $Pred$ is a function that assigns one of the two available processors (referenced as P_1 P_2), and t an integer.
Out: To decide whether there is some schedule having makespan $\leq t$.

The problem handled in this subsection is defined as below:

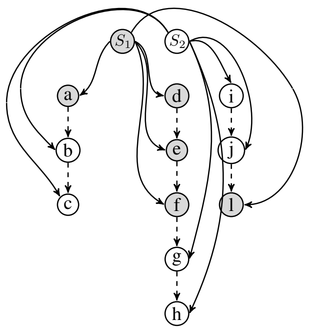

Figure 5. The figure describes a transformation from a set of three chains. Each task may be either grey or left unfilled, representing the two groups of tasks with the predefined processors. The tasks S_0 and S_1 are added along with infinitely large communication edges represented by continue lines, while the edges that connect any pair of tasks belonging to the same chain are dashed, representing negligible communicating delays.

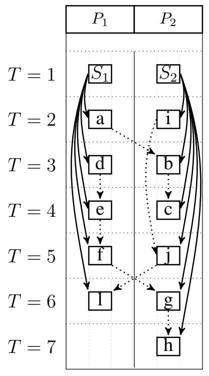

Figure 6. The figure shows the optimal scheduling for the Figure 5 along with the edges. Notice that continuous edges must not connect tasks scheduled in distinct processors.

SCHEDULING UNDER ARBITRARY PRECEDENCE
In: $\langle G = (V, E), C, t\rangle$, where G represents a directed acyclic graph, C represents communication delays between pairs of tasks ($C : E \rightarrow \{0, \infty\}$), and t is an integer.
Out: To decide whether there is a schedule having makespan $\leq t$.

The following theorem shows the *NP-completeness* of the SCHEDULING UNDER ARBITRARY PRECEDENCE RELATIONS problem.

Theorem 4**:** Scheduling under arbitrary precedence relations is $NP - complete$.

Again, the problem is in *NP* as the scheduling itself may be used as a certificate. We propose a reduction from the SCHEDULING TYPED TASK will be used to show that the problem is in $NP - hard$. Given any instance of $\langle Chains, Pred, t\rangle$ from the SCHEDULING TYPED TASK problem, let be $G = (V, E)$ such that V is the set of tasks in each chain with two additional tasks, named S_0 and S_1. E also contains the edges in $Chains$ along with edges that originates on S_0 (respectively S_1) and is directed to every task that has P_1 (respectively P_2) as the predefined processor. For edge e that originates in S_0 or S_1, set $C(e) = \infty$, for the others set the delays to zero. For the integer t, define it to be the same as for the instance of the SCHEDULING TYPED TASKS problem (See Figure 6). For the given construction, after S_0 and S_1 are scheduled, it is clear that the constraints imposed are equivalent on both problems.

4. Concluding remarks

This paper describes results on minimizing the *makespan* on scheduling problems subjected to communication delays fixed as 0 or ∞. There has been demonstrated that, when enough processors are available and tasks execution time restricted to a unit of time, it is possible to find an optimal schedule in polynomial time, if the precedence is restricted to an *out-tree* ($P_\infty|$ *out-tree;* $c_{i,j} \in \{0, \infty\}; p_j = 1|C_{max}$) or if task duplications are allowed ($P_\infty|prec; c_{i,j} \in \{0, \infty\}; dup; p_j = 1; |C_{max}$).

Also, we have shown a few NP-hardness results for unit time execution tasks. The first considers that the number of available processors is part of the instance even if the precedence are restricted to a set of chains ($P|chains; c_{i,j} \in \{0, \infty\}; p_j = 1; |C_{max}$). The second considers that a fixed number of pair of processors are available ($P_2|prec; c_{i,j} \in \{0, \infty\}; p_j = 1; |C_{max}$). These results leave open some questions: is it NP-complete to decide when there is a fixed number of processors, and when the precedence relation is somehow restricted.

References

[1] P. Chretienne. Tree scheduling with communication delays. *Discrete Applied Mathematics*, 49(1-3):129–141, Mar. 1994.

[2] R. de Freitas Rodrigues, M. C. Dourado, and J. L. Szwarcfiter. Scheduling problems with multi-purpose parallel machines. *Discrete Applied Mathematics*, 164:313–319, 2014.

[3] M. C. Dourado, R. d. F. Rodrigues, and J. L. Szwarcfiter. Scheduling unit time jobs with integer release dates to minimize the weighted number of tardy jobs. *Annals of Operations Research*, 169:81–91, 2009.

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

[5] D. W. Engels, J. Feldman, D. R. Karger, and M. Ruhl. Parallel processor scheduling with delay constraints. In *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pages 577–585. Society for Industrial and Applied Mathematics, 2001.

[6] M. Fujii, T. Kasami, and K. Ninomiya. Optimal Sequencing of Two Equivalent Processors. *SIAM Journal on Applied Mathematics*, 17(4):784–789, July 1969.

[7] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman New York, 1979.

[8] R. Graham, E. Lawler, J. Lenstra, and R. Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

[9] J. Hoogeveen, J. Lenstra, and B. Veltman. Three, four, five, six, or the complexity of scheduling with communication delays. *Operations Research Letters*, 16(3):129–137, Oct. 1994.

[10] K. Jansen. Analysis of scheduling problems with typed task systems. *Discrete Applied Mathematics*, 52(3):223–232, Aug. 1994.

[11] H. Jung, L. Kirousis, and P. Spirakis. Lower bounds and efficient algorithms for multiprocessor scheduling of dags with communication delays. In *Proceedings of the First Annual ACM Symposium on Parallel Algorithms and Architectures - SPAA '89*, pages 254–264, New York, New York, USA, 1989. ACM Press.

[12] C. H. Papadimitriou and M. Yannakakis. Towards an architecture-independent analysis of parallel algorithms. *SIAM Journal on Computing*, 19(2):322–328, Apr. 1990.

[13] R. d. F. Rodrigues. *Caracterizaes e algoritmos para problemas clssicos de escalonamento*. PhD thesis, UFRJ, Rio de Janeiro, 2009.

[14] M. H. Rothkopf. Scheduling Independent Tasks on Parallel Processors. *Management Science*, 12(5):437–447, Jan. 1966.

[15] S. V. Sevastyanov, R. a. Sitters, and a. V. Fishkin. Preemptive scheduling of independent jobs on identical parallel machines subject to migration delays. *Automation and Remote Control*, 71(10):2093–2101, Oct. 2010.

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