Improved Lower Bounds for Scheduling Ordered Tasks on a Bounded Number of Processors*

Manoel Campêlo† and Ricardo Corrêa†
Mestrado em Ciência da Comp., Univ. Fed. do Ceará
Campus Pici, Bloco 910, 60455-760 Fortaleza, CE, Brazil
{mcampelo,correa}@lia.ufc.br

Nelson Maculan‡
COPPE/Sistemas, Univ. Fed. do Rio de Janeiro
CP 68511, 21945-970 Rio de Janeiro, RJ, Brazil
{maculan@cos.ufrj.br}

Fábio Protti†
Núcleo de Comp. Eletrônica, Univ. Fed. do Rio de Janeiro
CP 2324, 20001-970 Rio de Janeiro, RJ, Brazil
{fabiop@nce.ufrj.br}

Abstract

The precedence-constrained scheduling problem with unit computation and communication costs is considered. Being the exact solution of this problem computationally hard to obtain, we present two new methods for computing lower bounds. In the first method, we define a decomposition of the poset describing the parallel program and we apply dynamic programming. For the second method, a linear relaxation of a new ILP formulation with $O(n^2)$ variables and constraints is used. Some experimental results are reported.

Keywords: Precedence-constrained scheduling, partially ordered sets, integer programming, linear relaxation, divide-and-conquer.

1 Introduction

In this paper, we consider the *Precedence-Constrained Scheduling Problem* (*PCSP*, for short) with unit computation and communications costs, where the parallel program is represented by a set $N = \{1, \cdots, n\}$ of non-preemptive tasks and a partial order $\prec$ on N. Given a set $Q = \{1, \cdots, q\}$ of identical processors, the aim is to determine a processor location and a starting time of each task in N such that the execution time (or *makespan*) is *minimized*. We assume that the processors do not share any memory space, being the inter-processor communications provided by a message-passing mechanism on an interconnection network. In addition, it is assumed that the message-passing mechanism is able to carry out an arbitrary number of messages simultaneously. However, two or more distinct tasks cannot be concurrently executed on a single processor. The PCSP is known to be in NP-Hard [16].

Scheduling the tasks of a parallel program is one of the most important problems for achieving high performance in implementing parallel applications. The importance of PCSP in practice has been motivated the quest for heuristics for solving arbitrary instances approximately (for surveys

*This work is partially supported by a CNPq (Brazil)/CNRS (France) project. Corresponding author: M. Campêlo.

†Partially supported by the Conselho Nacional de Desenvolvimento Científico e Tecnológico, CNPq, Brazil.

‡Partially supported by CNPq, and Fundação de Amparo à Pesquisa do Estado do Rio de Janeiro, Brazil.

on such heuristics, see [5, 12] and references therein). Despite this effort and the many heuristic approaches that have been tried to tackle it, there are relatively few methods available for solving the PCSP exactly [1, 4, 11]. In our view, the central difficulty is that of computing tight lower bounds efficiently so that an enumerative method can enumerate a number of feasible schedules which is not prohibitively high.

There are some few methods for computing lower bounds that are available in the literature. The most simple are based on either the structure of the task graph (critical path), or on the ratio between the number of tasks and the number of processors available. Both lower bounds are extremely poor. Fernández and Bussell proposed a method to improve a lower bound obtained with the two previous methods as a function of the minimum load that must be added to lower in order to execute the tasks when the number of processors is bounded [7]. This method, which was modified in [8, 9, 10], is more effective than the previous ones when the number of processors is small compared to the intrinsic parallelism of the program.

In this paper, we present a computational study of two new methods for lower bound computation. The first method is a dynamic programming approach applied to a decomposition of the task graph. Graph decomposition and dynamic programming have been successfuly used for lower bound computation in many applications. In special, Chrétiènne applied this method in the restricted case of PCSP where the transitive reduction of $\prec$ defines a tree and the number of processors is unbounded [6]. In this restricted case, the lower bound so obtained matches the minimum makespan. In the present paper, this technique is generalized to find lower bounds for arbitrary graphs and a bounded number of processors. An improvement of the Fernández and Bussel's method is used, which leads to tight lower bounds when the number of processors is small.

When the number of processors is relatively large, tighter lower bounds are obtained with the second method proposed in this paper. This one is based on linear relaxation of a new ILP formulation with $O(n^2)$ variables and constraints, which employes some of the ideas of the formulation by Munier and König [13] for the the case where the number of processors is unbounded. We represent a schedule by a chain partition of the poset of precedence constraints, and we use some flow constraints to impose a bounded number of processors. Lower bounds obtained with the first method are used to generate some valid inequalities.

The rest of the paper is organized as follows. The formal statement of the problem is given in Section 2, including a brief discussion on our view of the bounded number of processors constraint. The methods we propose in this paper are the subject of two successive sections. Section 3 is devoted to the dynamic programming method, including its definition and some algorithmic aspects. The ILP formulation is given in Section 4. In Section 5, experimental results are presented and analyzed. Finally, Section 6 contains some concluding remarks.

2 Statements and notation

The PCSP is formally stated in this section. The definitions and discussions are related to an instance $(N, \prec, Q)$. For a more detailed description of PCSP, the reader is refered to [5, 12].

2.1 Task graph

The tasks and the precedence constraints are usually represented by means of a directed acyclic graph $G = (N, \vdash)$, called *task graph*, where $\vdash$ stands for the transitive reduction of $\prec$. The following is the terminology associated with the task graph. If an ordered pair (i, j) is in $\prec$ $(\vdash)$, then write $i \prec j$ $(i \vdash j)$. Otherwise, write $i \not\prec j$ $(i \not\vdash j)$. If $i \vdash j$ then i is an *immediate predecessor* of j

and j itself is an *immediate successor* of i. More generally, tasks i and j are *dependent* if $i \prec j$ or $j \prec i$. Otherwise, i and j are *independent*, which is denoted $i \parallel j$. If $j \not\prec i$ for all $j \in N$, then i is a *minimal task of* $\prec$. If $i \not\prec j$ for all $j \in N$, then i is a *maximal task of* $\prec$. $M^-_\prec$ and $M^+_\prec$ stand for the set of all minimal and maximal tasks of $\prec$, respectively. An *inset* (*outset*) of $\prec$ is a set $N' \subseteq N$ such that $j \in N'$ and $i \prec j$ ($j \prec i$) implies $i \in N'$. Define $N^-_\prec[i]$ ($N^+_\prec[i]$) as the minimal inset (outset) of $\prec$ that contains i. We assume $N^+_\prec[0] = N^-_\prec[n+1] = N$. Similarly, define $N^-_\vdash[i]$ and $N^+_\vdash[i]$ as the minimal inset and outset of $\vdash$, respectively. Finally, define $N^-_\vdash(i) = N^-_\vdash[i] \backslash \{i\}$ and $N^+_\vdash[i] = N^+_\vdash[i] \backslash \{i\}$.

We need to define some more terminology associated with $\prec$ [15]. For $N' \subseteq N$, $\prec_{N'}$ stands for $\prec$ restricted to N', which is also a transitive, irreflexive and anti-symmetric relation. If $\prec_{N'}$ contains no pair of independent tasks, then $\prec_{N'}$ is called *a chain of* $\prec$ *induced by* N', or simply a *chain of* $\prec$. An *ℓ-chain partition* $\mathbf{C}$ *of* $\prec$, is a partition $C_1, C_2, \cdots, C_\ell$ of N such that C_k, for all $k \in \{1, \cdots, \ell\}$, induces a non-empty chain of $\prec$. The *length* of a chain induced by $C \subseteq N$ in $\prec$ is defined as $|C|$. Let an *extension* of $\prec$ be a transitive, irreflexive and anti-symmetric relation $\prec' \supseteq \prec$ on N. Let an *antichain of* $\prec$ be a set of pairwise independent tasks. The *width* of $\prec$, denoted by $\omega(\prec)$, is the maximum cardinality of an antichain of $\prec$.

2.2 Feasible schedules

A *schedule* σ is defined to be a pair (X, P), being $X = \{x_1, \cdots, x_n\}$, where $x_i \in \mathbb{Z}^+$ is the start time of task i, and $P = \{p_1, \cdots, p_n\}$, where $p_i \in Q$ is the processor on which task i is scheduled. That is, a schedule associates each task with a date when it starts its execution and a processor location, subject to the following constraints for any two distinct tasks i and j in N:

1. If $i \prec j$ then $x_j \geq x_i + 1$. In addition, if $p_i \neq p_j$ then $x_j \geq x_i + 2$.

2. Otherwise, assume that $i \parallel j$. Then, $p_i \neq p_j$ or either $x_j \geq x_i + 1$ or $x_i \geq x_j + 1$.

The makespan of σ is denoted $x_{n+1} = \max_{i \in M^+_\prec}\{x_i + 1\}$.

2.3 Number of processors

We close this introductory section making two observations on the feasible schedules we will consider in the rest of the paper. Such observations are in connection with the number of processors in Q, and describe our view of the bounded number of processors constraint. The first observation establishes a limit on q as a function of the structure of $\prec$.

Observation 1 $q \leq \omega(\prec)$.

In order to state the second observation, let us take a schedule σ. We say that a processor k is *used* in σ if there exists some task scheduled on k in σ. Considering Observation 1 and the schedule σ, we can note that either all processors are used or there exists at least two independent tasks, say i and j, scheduled on a single processor, say k. In the latter case, suppose that k' is an unused processor. Then, it can be shown that there exists two independent tasks i' and j' scheduled on k such that j' and all other tasks scheduled on k after j' can be moved to k', increasing the number of processors used in σ. The second observation is a consequence of this fact.

Observation 2 *There exists an optimum schedule which uses exactly q processors.*

3 Chain decompositions and their algorithmic aspects

This section comprises a definition of a decomposition of G into networks, and a general method for computing a lower bound on the minimum makespan of $(N, \prec, Q)$. Given $i \in N \cup \{0\}$ and $j \in N \cup \{n+1\}$, define the *network* G_{ij} as the subgraph of G induced by $N_{ij} = N^+_{\prec}[i] \cap N^-_{\prec}[j]$. For the sake of simplicity, we write $\prec_{ij}$ and $\vdash_{ij}$ to indicate $\prec_{N_{ij}}$ and $\vdash_{N_{ij}}$, respectively.

3.1 Chain cutsets and chain decompositions of G

Let a *chain cutset of* G_{ij} be a cutset $C \subset N_{ij}$ such that C defines a chain in $\vdash_{ij}$. We call a *piece of* G_{ij}*, relative to* C, the network of G_{ij} induced by the tasks of a component B of $G_{ij} - C$ and a special task t chosen from the chain cutset C. If the tasks in C succeed those in B, then t is the minimal task of C. Otherwise, t is the maximal task of C.

Let $\mathcal{C}$ be a set of disjoint chains of N. We write $\mathcal{C}_{ij}$ for the maximal subset of $\mathcal{C}$ containing only chain cutsets of G_{ij}. A piece of G_{ij}, *relative to* $\mathcal{C}_{ij}$, is a piece of G_{ij}, relative to some $C \in \mathcal{C}_{ij}$. The *chain decomposition of* G*, with respect to* $\mathcal{C}$, is the directed graph $D = (V, E)$, where V is a set of pairs $[i, j]$ representing the pieces G_{ij} of the chain decomposition. The set V is formed by $[0, n+1]$ and all pairs $[k, \ell]$, for $k, \ell \in N$ and $k \neq \ell$, such that there exists a pair $[i, j]$ of tasks having one of the following properties:

1. $\mathcal{C}_{ij} \neq \emptyset$ and $G_{k\ell}$ is a piece of G_{ij}, or
2. $\mathcal{C}_{ij} = \emptyset$ and either $(i = k$ and $\ell \vdash j)$ or $(j = \ell$ and $i \vdash k)$.

In both cases, if such a pair $[i, j]$ exists, then $([i, j], [k, \ell]) \in E$. Some of our algorithmic developments are based on the following property of D, whose proof is omitted.

Observation 3 *D is acyclic.*

3.2 Finding a chain decomposition

A key point in the chain decomposition method is the determination of a chain cutset in a piece of G. The following result is helpful in this context.

Proposition 1 *A chain C is a chain cutset of G_{ij} if and only if, for all $k \in C$, if $\ell \in N$ and $\ell \parallel k$, then $\ell \notin N_{ij}$.*

Proof. As defined in the proposition, consider a piece G_{ij} of G, a chain C of G_{ij}, and a task $\ell \in N_{ij}$. First, suppose that C is a chain cutset of G_{ij}. By contradiction, suppose that

$$(\ell \in N) \text{ and } (\ell \parallel k) \Rightarrow (\ell \notin N_{ij}), \quad \forall k \in C$$

does not hold. Since $\ell \in N_{ij}$, we have $\ell \parallel k$, for all $k \in C$. There exists a path P from i to j that includes ℓ since G_{ij} is a piece. This means that $P \cap C = \emptyset$ because, otherwise, ℓ would not be independent on some task in C. Consequently, P also belongs to $G_{ij} - C$, which contradicts the fact that $G_{ij} - C$ is disconnected.

Conversely, suppose that $\ell \parallel k$ does not hold for some $k \in C$. Then, either $\ell \prec k$ or $k \prec \ell$. It turns out that C is a subset of every path from i to j containing ℓ. Hence, removing C disconnects ℓ from either i or j. It follows that C is indeed a cutset. $\square$

A straightforward consequence of this fact is the following.

Corollary 1 *A task $k \in N$ is in some chain cutset if and only if there exists a piece G_{ij}, $i, j \in N$, such that $|N_{ij}^+(k) \cup N_{ij}^-(k)| = |N_{ij}| - 1$.*

This corollary suggests an algorithm to determine the maximal chain cutsets of G_{ij}. Using this algorithm, there exists an algorithm to find, simultaneously, a set of disjoint maximal chains $\mathcal{C}$ and the chain decomposition $D = (V, E)$ of G, relative to $\mathcal{C}$.

3.3 Lower bounds using a chain decomposition

In this subsection, we describe how divide-and-conquer can be used to compute lower bounds on the minimum makespan of the schedules of $(N, \prec, Q)$ based on a chain decomposition $D = (V, E)$ of G. Our technique uses an ordering $\{v_1 = [i_1, j_1], \cdots, v_\nu = [i_\nu, j_\nu]\}$ of the vertices of D obtained by reversing the order given by any topological order of V. By Observation 3, such an order exists and, consequently, a depth-first search in D can be used to efficiently determine it. Once the previous order is determined, a lower bound LB_{ij}, for each $[i, j]$ in V, is computed depending on one of the following two cases. In both of them, LB_{ij} is computed as a function of the outneighbors of $[i, j]$ in D. Notice that the outneighbors of $[i, j]$ in D appear before $[i, j]$ in the order of the vertices of D.

Consider initially, as the first case, that $\mathcal{C}$ contains some chain cutset of G_{ij}. In other words, consider that $\mathcal{C}_{ij} \neq \emptyset$. The next result follows directly from the fact that $|C_{t'}| - 1$ is a tight lower bound for the minimum makespan of $(C_{t'}, \prec_{C_{t'}}, Q)$.

Proposition 2 *Let $[i_1, j_1], \cdots, [i_r, j_r]$, $r \geq 1$, be the pieces of G_{ij}, relative to $\mathcal{C}_{ij} = \{C_1, \cdots, C_{r'}\}$, where r' is either $r - 1$, r or $r + 1$. Then,*

$$LB_{ij} = \sum_{t=1}^{r} LB_{i_t j_t} + \sum_{t'=1}^{r'} |C_{t'}| - r'$$

is a lower bound on the minimum makespan of $(N_{ij}, \prec_{ij}, Q)$. Moreover, if $LB_{i_t j_t}$ is tight for all $t \in \{1, \cdots, r\}$, then LB_{ij} is tight.

Now, for the second case, consider that G_{ij} does not contain any chain cutset in $\mathcal{C}$, *i.e.* $\mathcal{C}_{ij} = \emptyset$.

Proposition 3 *Let M_{ij} and V_{ij} be defined as follows:*

$$M_{ij} = \max\{LB_{k\ell} \mid ([i,j],[k,\ell]) \in E\} \text{ and } V_{ij} = \{[k,\ell] \mid ([i,j],[k,\ell]) \in E \text{ and } LB_{k\ell} = M_{ij}\}.$$

Then, $LB_{ij} = \begin{cases} M_{ij} + 1, & |V_{ij}| = 1 \\ M_{ij} + 2, & |V_{ij}| > 1 \end{cases}$ *is a lower bound on the minimum makespan of $(N_{ij}, \prec_{ij}, Q)$.*

Proof. To show that LB_{ij} is a lower bound, let us consider a schedule of G_{kj}, for $([i,j],[k,j]) \in E$. If its makespan is greater than M_{ij}, then every schedule of G_{ij} obtainable from it has makespan greater than $M_{ij} + 1$, and, consequently, at least LB_{ij}. Otherwise, if the makespan is exactly M_{ij}, then there are two cases. The first one occurs when there are at least two immediate successors k_1 and k_2 of i in N_{ij} such that the makespan of both $G_{k_1 j}$ and $G_{k_2 j}$ are M_{ij}. Otherwise, we have the second case. The first case yields that k_1 and k_2 are scheduled on distinct processors. So, a communication is inccurred between i and either k_1 or k_2, and the proposition follows. The second case is trivial. □

3.4 Improving the lower bounds

In this subsection, we propose an improvement on the Fernández and Bussel's method. Let us first briefly review their original idea [7]. Consider a time subinterval $[\theta_1, \theta_2) \subseteq [0, LB_{ij}+1)$, where both θ_1 and θ_2 are integers. Define the set of tasks that must be executed within the interval $[\theta_1, \theta_2)$ as

$$R_{ij}(\theta_1, \theta_2) = \{k \in N_{ij} \mid [LB_{ik}, LB_{ij} - LB_{kj}] \subseteq [\theta_1, \theta_2)\}.$$

The computation of $R_{ij}(\theta_1, \theta_2)$ is performed considering all the possibilities to start the tasks in order to minimize their impact in the interval $[\theta_1, \theta_2)$ without yielding a makespan greater than LB_{ij}. The key property of any task k in $R_{ij}(\theta_1, \theta_2)$ is that if $x_k^{(ij)} \geq \theta_2$ in a schedule σ_{ij}, then $|\sigma_{ij}| > LB_{ij}$. In addition, observe that $q(\theta_2 - \theta_1)$ is the maximum number of tasks that can be executed during the interval $[\theta_1, \theta_2)$. An improvement on LB_{ij} can be established when $S_{ij}(\theta_1, \theta_2) = |R_{ij}(\theta_1, \theta_2)| - q(\theta_2 - \theta_1) > 0$, as follows.

Theorem 1 (Fernández and Bussell, 1973 [7]) *Define*

$$FB_{ij} = \max_{[\theta_1, \theta_2) \in [0, LB_{ij}+1)} \left\{ \left\lceil \frac{S_{ij}(\theta_1, \theta_2)}{q} \right\rceil \right\}.$$

Then, $\max\{LB_{ij}, LB_{ij} + FB_{ij}\}$ *is a lower bound on the minimum makespan of a schedule of* $(N_{ij}, \prec_{ij}, Q)$.

This result can be improved if we observe the minimum amount of time that a task in $R_{ij}(\theta_1, \theta_2)$ must be delayed.

Proposition 4 *Let* $[\theta_1, \theta_2) \in [0, LB_{ij} + 1)$ *such that* $S_{ij}(\theta_1, \theta_2) > 0$. *Still, let* $s^{(ij)}_{\theta_1, \theta_2}$ *denote the* $S_{ij}(\theta_1, \theta_2)$*-th task in a non-increasing order of the tasks* ℓ *in* $R_{ij}(\theta_1, \theta_2)$ *according to* $LB_{\ell j}$. *Then,*

$$IFB_{ij} = \max_{[\theta_1, \theta_2) \in [0, LB_{ij}+1)} \left\{ \theta_2 + \left\lfloor \frac{S_{ij}(\theta_1, \theta_2)}{q} \right\rfloor - (LB_{ij} - LB_{s^{(ij)}_{\theta_1, \theta_2} j}) \right\}$$

and $LB_{ij} + IFB_{ij}$ *is a lower bound on the minimum makespan of a schedule of* $(N_{ij}, \prec_{ij}, Q)$.

Proof. Take θ_1 and θ_2 as in the statement of the proposition. Clearly, at least $s^{(ij)}_{\theta_1, \theta_2}$ tasks in $R_{ij}(\theta_1, \theta_2)$ must be executed after θ_2 in any schedule whose makespan is at most LB_{ij}. Let us determine a lower bound on the makespan of any schedule σ_{ij} in which a task $k \in R_{ij}(\theta_1, \theta_2)$ is scheduled at starting time $x_k^{(ij)} \geq \theta_2$. We get this lower bound if we observe the minimum amount of time needed to schedule k and its successors. This amount is LB_{kj}, which gives $\theta_2 + LB_{kj}$ as a lower bound on $|\sigma_{ij}|$. This bound corresponds to an improvement of $\theta_2 - (LB_{ij} - LB_{kj})$ on LB_{ij}. The proposition follows from the fact that task $s^{(ij)}_{\theta_1, \theta_2}$ corresponds to the $S_{ij}(\theta_1, \theta_2)$-th of such improvements. $\square$

4 Lower bounding via linear relaxation

In this subsection, we describe an ILP formulation for $(N, \prec, Q)$ based on [3].

4.1 Chain partition constraints

In order to describe our ILP formulation, we observe that every feasible schedule (X, P) corresponds to a chain partition of an extension $\prec'$ of $\prec$, defined as the transitive closure of

$$\prec \cup \{(i,j) \mid i,j \in N, p_i = p_j, x_i < x_j\},$$

while each chain of the chain partition is given by the set of tasks scheduled on a single processor in (X, P). It stems from this fact and Observation 2 that the set of feasible schedules corresponds to all q-chain partitions of q-extensions of $\prec$.

First, let us see how to describe the set of all q-chain partitions of $\prec$. In order to represent such a chain partition, define the binary variables w_{ij}, for all $i,j \in N$, $i \prec j$, with the following interpretation: $w_{ij} = 1$ means that i and j lie in a single chain in a chain partition of $\prec'$, and $w_{ij} = 0$ otherwise. The binary variables w_{0j}, for all $j \in N$, indicate whether j is minimal or not in some chain. Similarly, the binary variables $w_{i,n+1}$ indicate whether i is maximal or not in some chain of the chain partition.

The constraints are as follows. First, observe that every task must be in some chain, which means that it contains exactly one successor and one predecessor in this chain [13]. If a task is minimal or maximal in a chain, then it is preceeded by the dummy task 0 or succeeded by the dummy task $n+1$, respectively. Constraints (1) and (2) reflect these necessary conditions. Further, constraint (3) forces the number of chains to be q.

(1)	$w_{0j} + \sum_{i \prec j} w_{ij} = 1,$	$j \in N$
(2)	$w_{i,n+1} + \sum_{i \prec j} w_{ij} = 1,$	$i \in N$
(3)	$\sum_{i \in N} w_{i,n+1} = q$	
(4)	$w_{0j}, w_{ij}, w_{i,n+1} \in \{0,1\},$	$i,j \in N, i \prec j$

4.2 Handling independent tasks

Now consider the schedules containing at least one pair of independent tasks scheduled on a single processor. Besides the chain partition constraints, a schedule must satisfies the Constraints 1 and 2 established in Section 2.2. In order to handle such a situation, we include the binary variables w_{ij}, for all tasks i, j in the following set:

$$U = \{(i,j) \mid i,j \in N \text{ and the starting times of } i \text{ and } j \text{ are unrelated}\},$$

where the term "unrelated" means that the Constraint 1 does not establish any relation between the starting times of i and j in any optimum schedule. In this sense, it is clear if $i \prec j$ or $j \prec i$, then $(i,j) \notin U$ and, consequently, U is a subset of $\|$. Less clear though, there are situations where i and j are independent and, in addition, are considered related. Such a situation occurs when the precedence relations applied to the predecessors and successors of both i and j yield a relationship between the starting times of i and j. Barskiy proposed a simple way to determine some of these situations, as follows. Again, we denote by LB_{ij} the lower bound associated to a network $[i,j] \in V$.

Proposition 5 (Barskiy [2]) *Let $i,j \in N$ and σ be some feasible schedule of $(N, \prec, Q)$ whose makespan is $|\sigma|$. If*

$$LB_{0i} > |\sigma| - LB_{j,n+1}, \tag{5}$$

then the inequality $x_j < x_i$ holds.

Observe in Proposition 5 that if (5) holds, then it remains valid for all schedules whose makespan is at most σ and, in particular, for any optimum schedule. Based on this fact, we rewrite U as

$$\{(i,j) \mid i,j \in N, i \parallel j \text{ and } (LB_{0i} \leq |\sigma| - LB_{j,n+1} \text{ and } LB_{0j} \leq |\sigma| - LB_{i,n+1})\},$$

for some schedule σ. Thus, the formulation $\Pi(N, \prec, Q)$ is given as the minimization of x_{n+1}, subject to (3)-(4) and

(6)	$w_{0j} + \sum_{i \prec j} w_{ij} + \sum_{(i,j)\in U} w_{ij} = 1,$	$j \in N$
(7)	$w_{i,n+1} + \sum_{i \prec j} w_{ij} + \sum_{(i,j)\in U} w_{ij} = 1,$	$i \in N$
(8)	$x_j \geq x_i + 2 - w_{ij},$	$i,j \in N, i \vdash j$
(9)	$x_j \geq x_i - \alpha_{ij}\overline{w}_{ij} + 1,$	$i,j \in N, (i,j) \in U$
(10)	$x_{n+1} \geq x_i + 1,$	$i \in M^+_{\prec}$
(11)	$x_i \geq 0,$	$i \in M^-_{\prec}$

where $\overline{w}_{ij}$ stands for $1 - w_{ij}$ and constraint (9) expresses the disjunctive condition indicated in Constraint 2 of the definition of feasible schedules (Section 2.2). Observe that all pairs $(i,j) \in U$ such that $w_{ij} = 1$ define a q-extension $\prec'$ of $\prec$. Consequently, each constant α_{ij} must be sufficiently large to make constraint (9) redundant in at least one optimum feasible schedule whenever $(i,j) \in U$ and $\bar{w}_{ij} = 1$ [14].

4.3 Valid inequalities

Some valid inequalities for the formulation just presented can be written in function of the decomposition method of the previous section. They are described in the following straightforward proposition, whose proof is omitted.

Proposition 6 *The following are valid inequalities for* $\Pi(N, \prec, Q)$*:*

(12) $\quad x_j - x_i \geq LB_{ij} + IFB_{ij}, \quad [i,j] \in V$

5 Computational experiments

In this section, we present the comparison results and analysis of the two different methods for lower bound computation desribed in the previous sections.

5.1 Implementation details

Both methods presented in this paper have been implemented on a Pentium III 750 MHz microcomputer using CPLEX version 7.1 as the linear programming solver. The following is a brief description of some of its main aspects.

The improvement presented in Proposition 4 can only be effective if the width of G_{ij} is greater that the number of processors $(\omega(\prec_{ij}) > q)$. We evaluate $\omega(\prec_{ij})$ by solving a linear program. Actually, if the $(\omega(\prec_{ij}) > q)$ then the widths of all subgraphs containing G_{ij} do not need to be calculated. If $IFB_{ij} > 0$, we add contraint (12) to the ILP formulation.

Besides providing a lower bound we also calculate an upper bound for PCSP. We implement a list heuristic. At each iteration, a task whose predecessors have been scheduled is chosen. It is scheduled at a processor giving the possible minimum starting time.

5.2 The testbench

In our computational experiments, we have used a large variety of small to moderately sized task graphs in order to determine the robustness of the methods. Besides some randomly generated instances, we have also used as testbench some instances extracted from practical applications and some specially structured graphs. We briefly describe the task graphs in the following.

`ran`x These are randomly generated graphs. Each graph is generated by first choosing its number of tasks. Then, for each task i, choose the number of its immediate successors and elect them from the set of tasks that are candidates to be successors of i, both at random and using a uniform distribution. Prevent each elected task j to become a predecessor of i and of all of predecessors of i. The character x is a number used to differentiate among instances.

`bin`h, `di`n, `irr`41 These are full binary trees of height h, diamond graphs on n tasks and an irregular graph on 41 tasks, respectively.

`cho`ω, `fft`x, `gau`n, `mol` These graphs represent Cholesky Factorization, Fast Fourier Transform, Gaussian Elimination and molecular physics programs, respectively. ω stands for the width of each graph and n stands the number of tasks. The character x is a number used to differentiate among instances.

5.3 Experimental results

The results obtained with our experiments are summarized in Tables 1. Each task graph was tested in two situations, corresponding respectively to $q = \lfloor\omega(\prec)/2\rfloor$ and $\lfloor\omega(\prec)/4\rfloor$. The following information are reported:

Size: These are information about the size of the task graph, namely the number of tasks (n), the width and the length of a critical path.

UB: Stands for the upper bounds obtained with a list heuristic in the (width/2)/(width/4) cases.

Processors: Lists the results obtained with both methods, divided according to the two possible values of q. In each case, besides the values of q, the lower bounds and the execution times (in seconds) of the dynamic programming (column "DP") and ILP methods are presented.

There are a few things to observe in the results. The first one is that the dynamic programming approach runs faster than the linear relaxation of the ILP formulation. The main reason for that is the size of the linear program. On the other hand, the results of the linear relaxation are, in many cases where $q = \lfloor\omega(\prec)/2\rfloor$, better than the result of the dynamic programming approach. Note that, as a consequence of the valid inequalities describe in Subsection 4.3, the lower bound given by the linear relaxation is at least as large as the one given by the dynamic programming approach. The performance of both approaches when $q = \lfloor\omega(\prec)\rfloor$ indicates that the method described in Subsection 3.4 works very well if the number of processors available is significantly smaller than the intrinsic parallelism of the program. Finally, we note that the results obtained with both methods are significantly better than the lower bounds given by the length of a critical path and n/q.

6 Concluding remarks

We have in this paper introduced two novel approaches for computing the lower bound of unit cost precedence-constrained scheduling problems. The first approach is a dynamic programming

Graph	Size			UB	Processors									
					$\lfloor$width/2$\rfloor$					$\lfloor$width/4$\rfloor$				
	n	width	CP		q	DP	Time	ILP	Time	q	DP	Time	ILP	Time
ran1	140	42	11	19/24	21	13	4	14	14	10	16	5	16	17
ran2	80	29	10	17/19	14	12	0.5	13	2.5	7	12	1	13	4
ran3	153	48	11	18/26	24	13	8	14	26	12	13	10	14	31
ran4	108	30	11	21/27	15	14	2	15	8	7	16	2	16	11
ran5	170	94	8	14/16	47	9	3	9	17	23	9	6	9	19
ran6	223	110	8	13/19	55	8	10	9	33	27	10	13	10	40
ran7	298	119	9	16/21	59	10	20	11	78	29	12	25	12	114
ran8	256	118	9	13/18	59	11	15	11	45	29	11	21	11	85
ran9	124	65	6	10/13	32	7	2	8	7	16	9	1	9	9
ran10	286	83	14	23/31	41	17	38	17	131	20	17	55	17	174
ran11	510	153	12	22/25	76	15	241	17	794	38	17	264	18	814
ran12	310	99	11	20/24	48	15	219	16	788	24	17	215	17	821
ran13	357	167	8	14/19	82	10	414	10	1525	41	10	387	10	1679
ran14	364	121	9	16/21	60	13	193	14	544	30	15	224	15	588
ran15	152	47	11	20/24	22	15	15	15	60	11	16	20	16	69
ran16	186	76	9	14/20	38	10	5	11	23	19	10	9	11	30
ran17	150	37	6	17/26	18	11	53	11	158	9	11	61	11	201
ran18	100	38	6	14/21	19	10	6	11	20	9	10	8	11	30
ran19	250	66	10	22/26	33	18	1134	19	4455	16	19	1291	19	4641
ran20	50	20	4	10/21	10	7	0	7	2	5	7	1	7	2
ran21	200	42	11	25/31	21	20	251	20	764	10	20	264	20	781
bin3	7	4	3	9/13	3	6	0	7	0	1	7	-	7	-
bin4	15	8	4	11/16	4	7	0	7	1	2	9	0	9	1
bin5	31	16	5	13/18	8	9	0	9	2	4	9	1	9	3
bin6	63	32	6	15/20	16	11	1	11	4	8	11	2	11	7
bin7	127	64	7	17/21	32	13	8	13	30	16	13	6	13	24
bin8	255	128	8	19/24	64	15	39	15	90	32	15	33	15	110
cho19	209	19	38	53/76	9	38	82	43	180	4	56	55	56	104
cho29	464	29	58	81/107	14	58	4908	65	9060	7	73	3100	73	6302
di16	16	4	7	12/-	2	10	0	10	0	1	16	-	16	-
di25	25	5	9	17/-	2	15	0	15	1	1	25	-	25	-
di36	36	6	11	22/-	3	16	0	16	1	1	36	-	36	-
di64	64	8	15	26/38	4	22	3	22	10	2	34	1	34	4
di100	100	10	19	34/60	5	28	15	28	44	2	52	11	52	33
di144	144	12	23	44/57	6	34	103	34	236	3	52	36	52	72
di225	225	15	29	56/66	7	43	489	43	1813	3	52	110	52	433
di256	256	16	31	60/80	8	46	1110	46	3530	4	52	214	52	837
fft4	95	16	9	26/37	8	17	3	17	7	4	24	3.5	24	7.5
fft5	223	32	11	40/61	16	21	33	21	75	8	28	68	28	141
gau18	18	5	8	14/-	2	11	0	11	1	1	18	-	18	-
gau33	33	7	12	19/-	3	13	0	15	0	1	33	-	33	-
gau228	228	20	38	66/84	10	39	99	43	318	5	47	45	47	188
gau348	348	25	48	84/108	12	49	431	54	1723	6	60	321	60	1147
irr41	41	11	10	18/25	5	15	0.3	16	1.7	2	23	0	23	1
mol	40	11	9	19/26	5	14	0.4	15	1.6	2	22	0	22	1

Table 1: Results for random and benchmark graphs

method, based on a decomposition of the task graph. The second approach consists of the linear relaxation of a novel ILP formulation. We have presented experimental results on some random generated and benchmark task graphs, and compared the performance of both approaches. Our results indicate generally competitive performance in small to moderately sized task graphs.

Despite these positive results, there is room, at least in principle, for improvements and generalizations to be attempted. For example, the execution times may be significantly reduced if we use some efficient heuristic to select a small number of networks to compute the width. In addition, a decomposition method, such as Dantzig-Wolf method, may be employed to reduce the execution time of the linear relaxation solution. Finally, the generalization of both approaches for task graphs with arbitrary computation and communication costs seems to be straightfoward.

References

[1] I. Ahmad and Y. Kwok. Optimal and near-optimal allocation of precedence-constrained task to parallel processors: defying the high complexity using effective search technique. In *Proc. 1998 Int'l Conf. Parallel Processing*, pages 424–431, 1998.

[2] A. Barskiy. Minimizing the number of computing devices needed to realize a computational process within a specified time. *Eng. Cybern. (USSR)*, 6:59–63, 1968.

[3] M. Campêlo, R. Corrêa, N. Maculan and F. Protti. ILP formulations for scheduling ordered tasks on a bounded number of processors. In *Brazilian Symposium on Graphs, Algorithms and Combinatorics*, em Eletronic Notes in Discrete Mathematics, Vol. 7, 2001.

[4] P. Chang and Y. Jiang. A state-space search approach for parallel processor scheduling problems with arbitrary precedence relations. European J. on Op. Res., 77:208–223, 1994.

[5] T. Cheng. A state-of-the-art review of parallel-machine scheduling research. European J. on Op. Res., 47:271–292, 1990.

[6] P. Chrétienne. A polynomial algorithm to optimally schedule tasks on a virtual distributed system under tree-like precedence constraints. European J. on Op. Res., 43:225–230, 1989.

[7] E. Fernández and B. Bussell. Bounds on the number of processors and for multiprocessor optimal schedules. *IEEE Trans. on Comp.*, C-22(8):745–751, 1973.

[8] S. Fujita and M. Yamashita. Two approaches for improving Fernández and Bussel's lower bound. In *Workshop on Scheduling Algorithms for Par./Dist. Comp., ICS'99*, 28–21, 1999.

[9] K. Jain and V. Rajaraman. Lower and upper bounds on time multiprocessor optimal schedules. *IEEE Trans. on Par. and Dist. Syst.*, 5(8):879–886, 1994.

[10] K. Jain and V. Rajaraman. Improved lower bounds on time and processors for scheduling precedence graphs on multicomputer systems. *J. of Par. and Dist. Comp.*, 28(1):101–108, 1995.

[11] H. Kasahara and S. Narita. Practical multiprocessor scheduling algorithms for efficient parallel processing. *IEEE Trans. on Comp.*, C-33(11):1023–1029, 1984.

[12] Y.-K. Kwok and I. Ahmad. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Comp. Surveys*, 31(4):406–471, 1999.

[13] A. Munier and J.-C. König. A heuristic for a scheduling problem with communication delays. *INFORMS Op. Res.*, 45(1):145–147, 1997.

[14] G. Nemhauser and L. Wolsey. *Integer and Combinatorial Optimization.* Wiley, 1988.

[15] W. Trotter. Partially ordered sets. In R. Graham, M. Grötschel, and L. Lovász, editors, *Handbook of Combinatorics.* The MIT Press and North-Holland, 1995.

[16] J. Ullman. NP-Complete scheduling problems. *J. Comput. Systems Sci.*, 10:384–393, 1975.