Complexity and approximation for precedence constrained scheduling problems with large communication delays

Rodolphe Giroudeau *, Jean-Claude Konig, Farida Kamila Moulai, Jérôme Palaysi

LIRMM, Université de Montpellier II, UMR 5506 CNRS, 161 rue Ada, 34392 Montpellier Cedex 5, France

A R T I C L E   I N F O

Article history:
Received 30 January 2006
Received in revised form 18 March 2008
Accepted 23 March 2008
Communicated by G. Ausiello

Keywords:
Scheduling with large communication delays
Non-approximability
Approximation

A B S T R A C T

We investigate the problem of minimizing the makespan (resp. the sum of completion time) for the multiprocessor scheduling problem. We show that there is no hope of finding a ρ-approximation with ρ < 1 + 1/(c + 4) for minimization of the makespan (resp. 1 + 1/(2c+5) for total job completion time minimization) (unless \(\mathcal{P} = \mathcal{NP}\)) for the case where all the tasks of the precedence graph have unit execution times, where the multiprocessor is composed of an unrestricted number of machines, and where c denotes the communication delay between two tasks i and j submitted to a precedence constraint and to be processed by two different machines. The problem becomes polynomial whenever the makespan is at most (c + 1). The (c + 2) case is still partially open. Moreover, we both define and study a new scheduling approximation to schedule unitary tasks in the presence of large communication delays. We provide a polynomial-time approximation algorithm with performance ratio $\frac{5c+1}{4}$ with $c \geq 2$.

© 2008 Elsevier B.V. All rights reserved.

1. Introduction

1.1. Problem statement

Makespan scheduling problems are in the mainstream of operations research, industrial engineering, manufacturing systems and computer science. More particularly, scheduling theory is concerned with the optimal allocation of limited resources to activities over time. The theory of algorithm design for scheduling is more recent, but still has a significant history. From a theoretical point of view, the majority of scheduling problems are extremely difficult in the sense that finding optimum scheduling solutions is \(\mathcal{NP}\)-complete in general.

In this article we adopt the classical scheduling delay model, or homogeneous model, in which an instance of a scheduling problem is specified by a set $V = \{1, \ldots, n\}$ of $n$ non-preemptive tasks, a set $E$ of $q$ precedence constraints $(i, k)$ such that $G = (V, E)$ is a directed acyclic graph (dag), processing times $p_i$, $\forall i \in V$, and communication times $c_{ik}$, $\forall (i, k) \in E$.

If task $i$ starts its execution at time $t$ on processor $\pi$, and if task $k$ is a successor of $i$ in the dag, then either $k$ starts its execution after time $t + p_i$ on processor $\pi$, or after time $t + p_i + c_{ik}$ on some other processor. In what follows, we consider the case of $\forall k \in V$, $p_k = 1$ and $\forall (i, k) \in E$, $c_{ik} = c \geq 2$.

In the theory of scheduling, a problem type is categorized by its machine environment, job characteristic and objective function. So, using the three field notation scheme $\alpha/\beta/\gamma$, proposed by Graham et al. [9], the problem is denoted as $P|\text{prec}, \gamma = c \geq 2; p_i = 1|c_{\text{max}}$, i.e. we have an unrestricted number of identical processors in order to schedule a dag such that, each task has the same execution time and each pair of tasks has the same communication time. The aim is to

\* Corresponding author. Tel.: +33 4 67 41 85 40; fax: +33 4 67 41 85 00.
E-mail addresses: rgirou@lirmm.fr (R. Giroudeau), konig@lirmm.fr (J.-C. Konig), moulai@lirmm.fr (F.K. Moulai), palaysi@lirmm.fr (J. Palaysi).

1 Where $\alpha$ designates the environment processors, $\beta$ the characteristics of the job and $\gamma$ the criteria.

0304-3975/$ – see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2008.03.027
In what follows, this model is denoted as Large Communication Time model (LCT model). Note that if \( \forall i \in V \) such that \( p_i = 1 \) and \( \forall (i, k) \in E \) such that \( c_{ik} = 1 \), this model is called Unit Execution Communication Time–Unit Communication Time (UET–UCT model).

This UET–UCT model was first discussed by Rayward-Smith [20]. In this model we have a set of identical processors that are able to communicate in a uniform manner. We want to use these processors in order to process a set of tasks that are subjected to precedence constraints.

The difficulty in scheduling problems where communication delays are an issue, is to find a compromise between the two extreme solutions, namely, executing all tasks sequentially without communication, or trying to use all potential parallelism, but at the cost of increased communication overhead. This model has been extensively studied over recent years from both complexity and (non)-approximability points of view [3].

1.2. Contribution of this paper

The purpose of this article, is to propose two new complexity and approximation results concerning a scheduling problem on an unrestricted number of processors in the presence of large communication delays:

- On the one hand, in order to establish a lower bound for approximation algorithm performance, i.e. a threshold such that there exists no \( \rho \)-approximation with \( \rho < k \in \mathbb{R} \), we develop a polynomial-time reduction from an \( \mathcal{NP} \)-complete problem which is a variant of the problem 3\text{SAT} [6] and we use an Impossibility theorem [5].
- On the other hand, we derive a new approximation algorithm (an approximation algorithm for an optimization problem is an algorithm that provides a feasible solution whose quality does not differ significantly from the quality of an optimal solution) with a non-trivial ratio guarantee.

Moreover, we also wish to study stability concerning the complexity of the passage from an infinite number of processors to a finite number for the UET–UCT model and LCT model.

Thus, in this paper, we answer the following fundamental question: What is the impact of large communication delays on the complexity of the related scheduling problems, from the passage of a scheduling problem with an infinite number of processors to a scheduling problem with a restricted number of processors?

The variation (concerning the \( \mathcal{NP} \)-completeness with minimization of scheduling time as the objective) is two units of time for the UET–UCT model (see detailed results hereafter). Is variation the same in the presence of large communication delays?

1.3. Presentation of the paper

This article is organized as follows: in the next section, we present an overview of complexity and approximation for the scheduling problem with unitary tasks and communication delays. In Section 3, we give a preliminary complexity result concerning a variant of the \( \mathcal{NP} \)-complete 3\text{SAT}. This problem will be used, in Section 4, for the two non-approximability results concerning the scheduling problem with these objective functions: makespan minimization and total job completion time minimization. In Section 5, we propose a polynomial-time algorithm for \( C_{\max} = c + 2 \) with \( c \in \{2, 3\} \). In Section 6, we derive the polynomial-time algorithm with a worst case performance ratio of \( \frac{2(c+1)}{c+1} \) for the \( P_{|\text{prec}|} \); \( c_j = c \geq 2 \); \( p_i = 1|C_{\max} \) problem. The feasible schedule comes from expansion of the makespan of the \( P_{|\text{prec}|} \); \( c_j = 1 \); \( p_i = 1|C_{\max} \) schedule problem. We offer comments on our results in the final section.

2. Related work

In what follows, only the main results, in complexity and approximation, are retained in order to establish state-of-the-art scheduling with unitary tasks and communication delays.

2.1. Complexity results

2.1.1. Problems with unitary communication delays

First, concerning the problem of scheduling a precedence graph with unitary communication delays and unit execution time (UET–UCT), on an unrestricted number of processors, Hoogeveen et al. [11] proved that the decision problem associated with \( P_{|\text{prec}|} \); \( c_j = 1 \); \( p_i = 1|C_{\max} \) becomes \( \mathcal{NP} \)-complete even for \( C_{\max} \geq 6 \), and that it is polynomial for \( C_{\max} \leq 5 \). Their proof is based on a reduction from the \( \mathcal{NP} \)-complete problem 3\text{SAT} [6]. The \( \mathcal{NP} \)-completeness result for \( C_{\max} = 6 \) implies that there is no polynomial-time approximation algorithm with a better ratio guarantee than 7/6, unless \( \mathcal{P} = \mathcal{NP} \), and consequently there is no hope of finding a \( \mathcal{PTA} \delta \) for this problem.

\footnote{The length of the schedule (resp. the total job completion time) is defined as follows: \( C_{\max} = \max_{i \in V} (t_i + p_i) \) (resp. \( \sum_{j=1}^{n} c_j = \sum_{j=1}^{n} (t_j + p_j) \).}
Remark 1. Note that in the case of an unrestricted number of processors, Hoogeveen et al. [11] establish that an instance of \( P|\text{prec}; c_i = 1; p_i = 1|C_{\text{max}} \) having a schedule of length of four at the most is \( \mathcal{NP} \)-complete (a reduction from the \( \mathcal{NP} \)-complete problem \textit{Clique} is provided), whereas Picouleau [18] develops a polynomial-time algorithm for \( C_{\text{max}} = 3 \). In the same way, the \( \mathcal{NP} \)-completeness result for \( C_{\text{max}} = 4 \) implies that there is no polynomial-time approximation algorithm with a better ratio guarantee than \( 5/4 \), unless \( \mathcal{P} = \mathcal{NP} \) and there is no hope to find a \( \mathcal{PTAS} \) for this problem.

2.2.1. Problems with large communication delays

If we consider the problem of scheduling a precedence graph with large communication delays and unit execution time \((\text{\textit{UET}} - \text{\textit{LCT}})\), on a restricted number of processors, Bampis et al. in [1] proved that the decision problem denoted by \( P|\text{\textit{prec}}; c_i = c \geq 2; p_i = 1|C_{\text{max}} \) for \( C_{\text{max}} = c + 3 \) is an \( \mathcal{NP} \)-complete problem, and for \( C_{\text{max}} = c + 2 \) (for the special case \( c = 2 \)), they develop a polynomial-time algorithm. This algorithm cannot be extended for \( c \geq 3 \). Their proof is based on a reduction from the \( \mathcal{NP} \)-complete problem \textit{Balanced Bipartite Complete Graph}, \( \text{BBCG} \) [6,21]. Thus, Bampis et al. [1] showed that the \( P|\text{\textit{prec}}; c_i = c \geq 2; p_i = 1|C_{\text{max}} \) problem does not possess a polynomial-time approximation algorithm with a ratio guarantee better than \( (1 + \frac{1}{c+3}) \), unless \( \mathcal{P} = \mathcal{NP} \).

Complexity results\(^3\) are summarized in Table 1, and a classification tree for complexity results is shown in Fig. 1.

### Table 1

<table>
<thead>
<tr>
<th>Machines</th>
<th>( c_{ij} )</th>
<th>( C_{\text{max}} )</th>
<th>Complexity</th>
<th>Lower bound</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P )</td>
<td>( c = 1 )</td>
<td>5</td>
<td>Polynomial</td>
<td>[23]</td>
<td></td>
</tr>
<tr>
<td>( P )</td>
<td>( c = 1 )</td>
<td>6</td>
<td>( \mathcal{NP} )-complete</td>
<td>7/6 ( \leq \rho )</td>
<td>[23]</td>
</tr>
<tr>
<td>( P )</td>
<td>( c = 1 )</td>
<td>3</td>
<td>Polynomial</td>
<td>[18]</td>
<td></td>
</tr>
<tr>
<td>( P )</td>
<td>( c = 1 )</td>
<td>4</td>
<td>( \mathcal{NP} )-complete</td>
<td>5/4 ( \leq \rho )</td>
<td>[23]</td>
</tr>
<tr>
<td>( P )</td>
<td>( c \geq 2 )</td>
<td>&gt;c</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>( P )</td>
<td>( c \geq 2 )</td>
<td>c + 1</td>
<td>Polynomial</td>
<td>[1]</td>
<td></td>
</tr>
<tr>
<td>( P )</td>
<td>( c \geq 2 )</td>
<td>c + 3</td>
<td>( \mathcal{NP} )-complete</td>
<td>( 1 + 1/(c + 3) \leq \rho )</td>
<td>[1]</td>
</tr>
</tbody>
</table>

\( \mathcal{NP} \)-complete, weak\[19\] 2.2.1. Problems with unitary communication delays

The best known approximation algorithm for \( P|\text{\textit{prec}}; c_i = 1; p_i = 1|C_{\text{max}} \) is due to Munier and Konig [16]. They presented a \((4/3)\)-approximation algorithm, which is based on an integer linear programming formulation. The algorithm is based on the following procedure: an integrity constraint is relaxed, and the feasible schedule is produced by rounding down.

Munier and Hanen [14] proposed a \((\frac{\rho}{2} - \frac{4}{3m})\)-approximation algorithm for the problem \( P|\text{\textit{prec}}; c_i = 1; p_i = 1|C_{\text{max}} \). They define and study a new list scheduling approximation algorithm based on the solution given for an unrestricted number of

---

\(^3\) \( \text{dup} \) (resp. without \( \text{dup} \)) indicates that the duplication of tasks is allowed (resp. forbidden).
processors. They also introduced the notion of favourite successor in order to define priorities between conflicting successors of a task.

### 2.2.2. Problems with large communication delays

Very little is known about the design of an efficient approximation algorithm for the problem with large communication delays. In contrast to complexity results, as we know, a unique approximation algorithm is given by Rapine [19]. The author gives the lower bound $O(c)$ for list scheduling (one of the most often used general approximation strategies for solving scheduling problems is list scheduling, whereby a priority task list is given, and at each step the first available processor is selected to process the first available task on the list).

In the case of a restricted number of processors, a single (known) constant 2-approximation algorithm is given by Munier [12], for the special case where the precedence graph is of tree form. Approximation results are summarized in Table 2.

Note that if large communication delays are considered, there is no known $\rho$-polynomial-time approximation algorithm, except the trivial bound $(c+1)$, where the first step consists of executing the tasks and the second step consists of initiating communication phases, and so on.

The classification tree for approximation algorithm results is shown in Fig. 2.

### 3. Preliminary results

In this section, we define a variant of the SAT problem [6], denoted subsequently by $\Pi_1$. The $\mathcal{NP}$-completeness of the scheduling problem $P|\text{prec} ; c_q = c \geq 2 ; p_i = 1|C_{\text{max}}$ (see Section 3), is based on a reduction of this problem.

**The problem** $\Pi_1$ is a variant of the well-known SAT problem [6]. We will call this variant the One-in-(2, 3)SAT(2, 1) problem.

Let $\pi$ be an instance of $\Pi_1$. We use $\mathcal{V}$ to denote the set of $n$ variables. Let $n$ be a multiple of 3 and let $\mathcal{C}$ be a set of clauses of cardinality 2 or 3. There are $n$ clauses of cardinality 2 and $n/3$ clauses of cardinality 3 such that:

- Each clause of cardinality 2 is equal to $(x \lor \bar{y})$ for some $x, y \in \mathcal{V}$ with $x \neq y$.
- Each of the $n$ literals $x$ (resp. of the literals $\bar{x}$) for $x \in \mathcal{V}$ belongs to one of the $n$ clauses of cardinality 2, thus to only one of them.
- Each of the $n$ (positive) literals $x$ belongs to one of the $n/3$ clauses of cardinality 3, thus to only one of them.
- Whenever $(x \lor \bar{y})$ is a clause of cardinality 2 for some $x, y \in \mathcal{V}$, then $x$ and $y$ belong to different clauses of cardinality 3.

**Question:** Is there a truth assignment $I : \mathcal{V} \rightarrow \{0, 1\}$ whereby each clause in $\mathcal{C}$ has exactly a one true literal?
For all clauses with a length of two denoted by \( \bar{C} \). Let us first assume that there is a schedule with length of \( \bar{C} \) (explains reduction):

\[
(x_0 \lor x_1 \lor x_2) \land (x_3 \lor x_4 \lor x_5) \land (x_6 \lor x_1) \land (x_4 \lor x_2) \land (x_5 \lor x_4) \land (x_5 \lor x_1).
\]

The answer to \( \Pi_I_1 \) is yes. It is sufficient to choose \( x_0 = 1 \) (for true), \( x_3 = 1 \) and \( x_6 = 0 \) (for false) for \( i = 1, 2, 4, 5 \). This yields a truth assignment that satisfies the formula, and there is exactly one true literal for each clause. For proof of the \( \mathcal{NP} \)-completeness see [7].

4. Computational complexity

For the problem \( P|\text{prec}|; c_j = c \geq 2; p_i = 1|C_{\text{max}} \) the challenge, is to determine a lower bound on the performance of approximation algorithm. In this section, we first show that the problem denoted by \( P|\text{prec}|; c_j = c \geq 2; p_i = 1|C_{\text{max}} \) cannot be approximated by a polynomial-time approximation algorithm with a ratio guarantee better than \((1 + \frac{\ln c}{c - 2})\) for minimization of the makespan (resp. \((1 + \frac{1}{2c - 5})\) minimization of total job completion time). We also prove the \( \mathcal{NP} \)-completeness for the special case where \( c = 2 \) and \( C_{\text{max}} = 6 \) in Section 4.2.

4.1. Makespan minimization

Theorem 2. The problem of deciding whether an instance of \( P|\text{prec}|; c_j = c; p_i = 1|C_{\text{max}} \) has a schedule of length at most \((c + 4)\) is \( \mathcal{NP} \)-complete with \( c \geq 3 \).

Proof. It is easy to see that \( P|\text{prec}|; c_j = c; p_i = 1|C_{\text{max}} = c + 4 \in \mathcal{NP} \).

Our proof is based on a reduction from \( \Pi_1 \). Given an instance \( \pi^{*} \) of \( \Pi_1 \), we construct an instance of the problem \( P|\text{prec}|; c_j = c; p_i = 1|C_{\text{max}} = c + 4 \), in the following way (Fig. 3 explains reduction):

Remark 3. \( n \) denotes the number of variables of \( \pi^{*} \).

1. For all \( l \in \mathcal{V} \), we introduce \((c + 6)\) variable-tasks: \( \alpha_{l+1}, \alpha_{l+2}, \beta_{l+1}, \beta_{l+2}, \alpha_{l+3}, \beta_{l+3} \) with \( j \in \{1, 2, \ldots, c + 2\} \). We add precedence constraints: \( \alpha_{l+1} \rightarrow \alpha_{l+2}, \beta_{l+1} \rightarrow \beta_{l+2} \) with \( j \in \{1, 2, \ldots, c + 1\} \).

2. For all clauses with a length of three denoted by \( C_i \) = \((\lor \lor z \lor t)\), we introduce \((2 + c)\) clause-tasks \( A_{l+1} \) and \( A_{l+2} \), \( j \in \{1, 2, \ldots, c + 2\} \), with precedence constraints: \( C_{l+1} \rightarrow A_{l+1} \) and \( A_{l+1} \rightarrow A_{l+2} \) with \( j \in \{1, 2, \ldots, c + 1\} \). We add the constraints \( C_{l+1} \rightarrow l \) with \( l \in \{y, y', z, z', t\} \) and \( l \rightarrow A_{l+2} \) with \( l \in \{y, y', z, z', t\} \).

3. For all clauses with a length of two denoted by \( C_i = (x \lor y) \), we introduce \((c + 3)\) clause-tasks \( D_{l+1} \) and \( D_{l+2} \), \( j \in \{1, 2, \ldots, c + 2\} \) with precedence constraints: \( D_{l+1} \rightarrow D_{l+2} \) with \( j \in \{1, 2, \ldots, c + 2\} \) and \( l \rightarrow D_{l+2} \) with \( l \in \{x, y\} \).

The above construction is illustrated in Fig. 3. This transformation can be computed clearly in polynomial time.

Remark 4. \( \tilde{t} \) is in the clause \( C' \) of length two, associated with path \( D_{l+1} \rightarrow D_{l+2} \rightarrow \cdots D_{l+2} \rightarrow D_{l+3} \).

• Let us first assume that there is a schedule with length of \((c + 4)\) at most. In the following, we will prove that there is a truth assignment \( l : \mathcal{V} \rightarrow \{0, 1\} \) such that each clause in \( C \) has exactly one true literal (i.e., one literal equal to 1).

First we can note that if \( c \geq 3 \) then \( 2c + 2 > c + 4 \) and so, each path \( A_{l+1} \), \( A_{l+2} \), \( B_{l+1} \) with \( j \in \{1, 2, \ldots, c + 2\} \) and \( f \in \{1, 2, \ldots, c + 3\} \) must be executed on the same processor. Furthermore, two of these paths cannot be allotted on the same processor.

Notation: In the following, use \( P_{A} \) (resp. \( P_{B} \)) to denote the set of the \( \frac{1}{2} \) processors that execute a path \( A_{l+1} \) (resp. a path \( C_{l+1} \)). Note that from the definition of problem \( \Pi_1 \), that an instance admits \( \frac{1}{2} \) clauses of length three where \( n \) denotes the number of variables. In the same way, we use \( P_{A} \) (resp. \( P_{B} \)) to denote the set of \( n \) processors executing a path \( B_{l+1} \) (resp. a path \( A_{l+1} \)).
Lemma 5. For $c_{\text{max}} = c + 4$: the decision to assign the true value to the variable $x$, iff the variable-task $x'$ is executed on a processor on which scheduled path $P_{c}$, leads to a correct solution.

Proof. In order to respect the feasible schedule of length $(c + 4)$, we can first derive following the polynomial-time transformation: the starting time of variable-tasks $I$, $I'$ and $\hat{I}$, and that the processors on which these tasks must be executed, are given by the following remarks:

- Each variable-task $I$ is executed on a processor of $P_{c}$ at slot 3 or on a processor of $P_{b}$ at slot $(c + 2)$ or $(c + 3)$.
- Each variable-task $I'$ is processed on a processor of $P_{b}$ at slot 3 or on a processor of $P_{b}$ at slot $(c + 2)$ or $(c + 3)$.
- Each variable-task $\hat{I}$ is allotted on a processor of $P_{b}$ at slot 2 or 3, or on a processor of $P_{a}$ at slot $(c + 2)$ or $(c + 3)$.
- The variable-tasks $I'$ and $\hat{I}$ cannot be processed together on a processor of $P_{b}$ (these tasks assume a common predecessor).

Notation and property: We can associate the three tasks $I$, $I'$, $\hat{I}$ for each $l \in \mathcal{V}$. We denote three sets of tasks by $X = \{l | l \in \mathcal{V} \}$, $\hat{X} = \{l | l \in \mathcal{V} \}$ and $\hat{X} = \{l | l \in \mathcal{V} \}$. For each subset $A$ of $X$ (resp. $\hat{X}$), we can associate a subset $B$ of $X$ as follows:

<table>
<thead>
<tr>
<th>$P_{c}$</th>
<th>$P_{b}$</th>
<th>$P_{a}$</th>
<th>$P_{0}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x'$</td>
<td>$x_{1}$</td>
<td>$x_{2}$</td>
<td></td>
</tr>
<tr>
<td>$x'$</td>
<td>$x_{3}$</td>
<td>$x_{4}$</td>
<td></td>
</tr>
<tr>
<td>$x'$</td>
<td>$x_{5}$</td>
<td>$x_{6}$</td>
<td></td>
</tr>
</tbody>
</table>

Using the variable $x_{i}$ from the previous above table, we obtain the following inequations system:

1. $x_{1} + x_{2} = n$
2. $x_{3} + x_{4} = n$
3. $x_{5} + x_{6} = n$
4. $x_{1} \leq \frac{n}{2}$
5. $x_{6} \leq \frac{2n}{3}$
6. $x_{3} + x_{5} \leq n$
7. $x_{2} + x_{4} \leq n$.

Here, we precise some details about the above system:

- For the Eqs. (1)–(3): all the tasks of sets in $X$, $\hat{X}$ and $\hat{X}$ must be executed.
- For Eq. (4), on the processor which executes the path $C_{l}$ of the clause $C_{l} = (y \lor z \lor t)$, at most one of the three variable-tasks $y'$, $z'$, $t'$ can be processed. Indeed, all variable-tasks $I$ assume a successor which is executed on a processor of $P_{b}$. If it is allotted on the processor which scheduled the tasks from path $P_{c}$, it cannot be scheduled before slot 3, and so the variable-task $\alpha_{c}$ must be executed on the same processor which becomes saturated. Thus, we obtain $|X| < |P_{c}|$.
- For Eq. (5), each processor of paths $P_{a}$ has two free slots and $|P_{a}| = \frac{2}{3}$.
- For Eq. (6), all the variable-tasks $I'$ or $\hat{I}$ that are executed on a processor of the path $P_{b}$ must be finished before slot 3 (it has a successor scheduled on another processor). The variable-task $\alpha_{c}$ must be executed on the same processor which becomes saturated. Therefore, only one of variable-tasks $I'$ and $\hat{I}$ can be processed on a processor of the path $P_{b}$ and therefore $|X_{3}| + |X_{5}| \leq |P_{b}|$.
- For Eq. (7), it is clear that $|P_{b}| = n$, and that is one free slot at most on each processor of $P_{a}$.

On the one hand, we obtain $x_{3} + x_{5} = n$ (indeed, we have $x_{3} + x_{4} + x_{5} + x_{6} = 2n$ and $x_{6} \leq \frac{2n}{3}$, $x_{4} \leq \frac{n}{2}$, so $x_{3} + x_{5} \geq n$).

On the other hand, $\forall l \in \mathcal{V}$, only one of the variable-tasks $I'$ and $\hat{I}$ can be executed on a processor of $P_{b}$, thus we obtain $X_{3} \cap X_{5} = \emptyset$. Consequently, we have $X_{3} \cup X_{5} = X$. As the set $X_{4}$ (resp. $X_{5}$) is complementary to the set $X_{3}$ (resp. $X_{5}$), we obtain $X_{4} \cup X_{6} = X$.

Moreover, if $I'$ is executed on a processor of $P_{c}$, then the variable-task $\alpha_{c}$ is allotted on the same processor. Thus, the variable-task $\hat{X}$ cannot be processed before slot $(c + 2)$, thus it is executed on a processor of $P_{b}$. We can deduce that $X_{4} = X$ (the two sets are the same of cardinality).
Lastly, we obtain $X_1 \cup X_2 = X, X_3 \cup X_4 = X, X_5 \cup X_6 = X, X_4 \cup X_6 = X, X_3 \cup X_5 = X, X_1 = X_4$, and therefore $X_1 = X_4 = X_5$ and $X_2 = X_3 = X_6$.

We can deduce from above equations that:

$$x_1 = x_4 = x_5 = \frac{n}{3}$$

and

$$x_2 = x_3 = x_6 = \frac{2n}{3}.$$ 

If we assign the value “true” to the variable $l$ iff the variable-task $l'$ is executed on a processor of $P_c$. It can be seen clearly that in the clause of length 3 we have one and only one literal equal to “true”.

Consider $C = (x \lor y)$, a clause of length 2.

- If $x' \in X_1 \Rightarrow y' \in X_4 \Rightarrow y' \in X_1$. The first implication (resp. the second) is due to the fact that each processor of path $P_0$ must be saturated ($x_2 + x_4 = n$) (resp. $X_1 = X_4$). Only the literal $x$ is “true” between the literals $x$ and $y$.
- If $x' \in X_2 \Rightarrow y' \in X_3 \Rightarrow y' \in X_2$. The first (resp. the second) implication is due to the fact that there is only one free slot on each processor executing path $P_0$ (resp. $X_3 = X_2$). Of literals $x$ and $y$, only the literal $\bar{y}$ is “true”.

In conclusion, there is only one true literal per clause.

This concludes proof of Lemma 5. □

Conversely, we suppose that there is a truth assignment $l : \forall \rightarrow \{0, 1\}$, such that each clause in $C$ has exactly one true literal.

Suppose that the true literal in clause $C_i = (y \lor z \lor t)$ is $t$. Therefore, the variable-task $t'$ (resp. $y'$ and $z'$) is processed at slot 2 (resp. at slot $(c + 2)$) on the same processor as path $P_{c_i}$ (resp. as paths $P_{D}$ and $P_{D'}$, where $D$ and $D'$ indicate clauses of length two where the variables $y$ and $z$ occurred). The $\frac{c}{2}$ other variable-tasks $y'$, not yet scheduled, are executed at slot 3 on processor $P_0$ as variable-task $\alpha_{y,y'}$. Variable-task $\bar{t}'$ (resp. $\bar{y}'$ and $\bar{z}'$) is executed at slot 2 (resp. $c + 2$ and $c + 3$) on a processor of path $P_D$ (resp. $P_D'$).

This concludes proof of Theorem 2. □

4.2. Special case $c = 2$

In this section, we consider the following special case: $\forall (i, j) \in E, \text{if } \pi' = \pi = \pi''$ then $t_j \geq t_i + p_i \text{ else } t_j \geq t_i + p_i + 2$. We study the complexity of this problem, and then we will prove $\mathcal{NP}$-completeness of the $\tilde{P}_{P_{\text{prec}}}$: $c_\tilde{y} = 2; p_i = 1|\text{C}_{\text{max}} = 6$. Remember that (see [18,23]), for the classic UET–UCT problem ($\tilde{P}_{\text{prec}}$: $c_\tilde{y} = 1; p_i = 1|\text{C}_{\text{max}} = 6$), the problem becomes (resp. is) $\mathcal{NP}$-complete (resp. polynomial) for $|\text{C}_{\text{max}} = 6$ (resp. $|\text{C}_{\text{max}} = 5$).

Based on the problem $\Pi_1$, construction of an instance of the scheduling problem $\tilde{P}_{\text{prec}}$: $c_\tilde{y} = 2; p_i = 1|\text{C}_{\text{max}}$ is similar to the previous case.

Theorem 6. The problem of deciding whether an instance of $\tilde{P}_{\text{prec}}$: $c_\tilde{y} = 2; p_i = 1|\text{C}_{\text{max}}$ has a schedule with length at most of six is $\mathcal{NP}$-complete.

Proof. Before developing a polynomial-time transformation, it is important to explain why the polynomial-time transformation suggested for demonstrating of $\mathcal{NP}$-completeness of the scheduling problem cannot be applied to the particular case $c = 2$.

Let us suppose that we apply reduction with $c = 2$. Consider that $C_{ij} = (y \lor z \lor t)$ is a clause of length three. We suppose that the variable-task $y$ is executed at $t = 2$ on processor $\pi$. For all clauses of length three, by construction, a path of length four is created (here, the following $C_i^1 \rightarrow C_i^2 \rightarrow C_i^3 \rightarrow C_i^4$ is created). The variable-task $y$ associated with task $y$ assumes two predecessors. We suppose that the clause-task $C_i^1$ is executed at $t = 0$. Then, the variable-task $\alpha_{y,y'}$ is scheduled at $t = 1$. Then, the remaining clause-tasks of the path $C_2^1 \rightarrow C_2^2 \rightarrow C_2^3 \rightarrow C_2^4$ may be executed on another processor than $\pi$. On processor $\pi$, variable-tasks $z$ and $\bar{x}$ and the clause-task $\bar{C}_{ij}$, where $C_{ij} = (z \lor x)$, may be scheduled. By repeating this assignment for all clauses of length three, the number of tasks being executed on $P_0$ is $n$ (remember that $P_0$ is the set of $\frac{n}{2}$ processors that execute a path $C_j$ with $j \in \{1, \ldots, 4\}$ for a clause of length three $C_j$). In this preceding proof, we certify that for the suggested construction, the number of tasks being scheduled is $n/3$, which is a contradiction.

Let us return to the demonstration of the theorem.

It is clear to see that $\tilde{P}_{\text{prec}}$: $c_\tilde{y} = 2; p_i = 1|\text{C}_{\text{max}} = 6$ is in $\mathcal{NP}$.

Our proof is based on a reduction from $\Pi_1$.

Given an instance $\Pi_1$ of $\Pi_1$, we construct an instance $\pi$ of the problem $\tilde{P}_{\text{prec}}$: $c_\tilde{y} = 2; p_i = 1|\text{C}_{\text{max}} = 6$ as follows:

1. For all $l \in \forall$, we introduce five variable-tasks: $\alpha_{\pi}, \bar{t}', \bar{t}, t, \beta_{\pi}$. We add precedence constraints: $\alpha_{\pi} \rightarrow \bar{t}', \alpha_{\pi} \rightarrow \bar{t}, \beta_{\pi} \rightarrow \bar{t}, \beta_{\pi} \rightarrow \bar{t}$.
2. For all clauses of length three denoted by $C_i = (y \lor z \lor t)$, we introduce two clause-tasks $C_i$ and $A'$. We add the following precedence constraints: $C_i \rightarrow I$ with $l \in \{y', \bar{z}', \bar{t}'\}$ and $I \rightarrow A'$ with $l \in \{\bar{y}', z', t\}$.
(3) For all clauses of length two denoted by $C_i = (x \lor \bar{y})$, we introduce five clause-tasks $D^i_j, j \in \{1, 2, 3, 4, 5\}$ with precedence constraints: $D^i_j \rightarrow D^i_{j+1}$ with $j \in \{1, 2, 3, 4\}$ and $I \rightarrow D^i_5$ with $I \in \{x, \bar{y}\}$.

The above construction is illustrated in Fig. 4. This transformation can be computed clearly in polynomial time.

Let us first assume that there is a schedule with length at most of six. In what follows, we will prove that there is a truth assignment $I : \mathcal{V} \rightarrow \{0, 1\}$ such that each clause in $C$ has exactly one true literal.

First, we can note that if a task is executed at slot 3 (resp. at slot 4) or before (resp. after), all these predecessors (resp. successors) are allotted on the same processor. Since communication is allowed on a path of length five so that $\forall i$, all the clause-tasks $D^i_j$ are executed on the same processor.

**Lemma 7.** For $c_{\text{max}} = 6$: the decision to assign the true value to the variable $x$ iff the variable-task $x'$ is executed at slot 3 leads to a correct solution.

**Proof.** $\forall I \in \mathcal{V}$, by construction, it is clear that:

- Variable-task $l$ is executed at slot 3 on the processor that processed variable-task $\alpha_l$ and clause-task $C^i$ where variable $l \in C_i$, or after slot 4 on the processor that scheduled $D^i_l$ such that $l \in C_i$.
- Variable-task $l$ is processed at slot 3 on the processor that executed tasks $\alpha_l$ and $\beta_l$ or after slot 4 on the processor which executed $D^i_l$ such that $l \in C_i$.
- Variable-task $l$ is allotted at slot 2 or 3 on the processor that executed $\beta_l$ or after slot 4 on the processor that processed $A^i$ such that $x \in C_i$.

Using the same notation as previously, we consider

- $X_1 = \{I' \cap t_v = 3\}$ where $t_v$ designates the starting of task $j$
- $X_2 = \{I' \cap t_v \geq 4\}$
- $X_3 = \{I' \cap t_v = 3\}$
- $X_4 = \{I' \cap t_v \geq 4\}$
- $X_5 = \{I' \cap t_v = 2 \lor t_v = 3\}$
- $X_6 = \{I' \cap t_v \geq 4\}$.

Consider $x_i = |X_i|$ for $i \in \{1, \ldots, 6\}$. From the construction, we derive:

- $x_1 \leq \frac{n}{2}$ (there are only $\frac{n}{2}$ clause-tasks $C_i$)
- $x_2 + x_4 \leq n$ (there are only $n$ processors that executed a clause-task $D^i_5$)
- $x_3 + x_5 \leq n (\forall x, \bar{x}$ and $\bar{x}$ cannot be scheduled together at slot 3 in the same processor (they have common predecessor))
- $x_6 \leq \frac{3n}{5}$ (there are at most $\frac{3n}{5}$ processors that executed a clause-task $A^i$ and each processor has only slots 4 and 5 to process variable-tasks $x'$).

Thus, we can deduce from $\sum_{i=1}^{6} x_i \leq 3n$. As $\sum_{i=1}^{6} x_i = 3n$ (all variable-tasks must be executed), the inequalities are all equalities.

And lastly, we have $x_1 = x_4 = x_5 = \frac{n}{2}$ and $x_2 = x_3 = x_6 = \frac{3n}{5}$.

If the variable-task $l$ (resp. the variable-task $l$) is executed at slot 3, the variable-task $\bar{l}$ (resp. $l'$) must be scheduled at, or after, slot 4. Therefore, $x_1 \leq x_4$ and $x_3 \leq x_2$ and we obtain $x_1 = x_4$ and $x_3 = x_2$.

Thus if we assign the value “true” to the variable $l$ if and only if the variable-task $l$ is executed at slot 3, we can clearly see that in the clause of length 3, we have one and only one literal equal to “true”.

Consider $C = (x \lor \bar{y})$, a clause of length two. As in proof for the Theorem 2, we obtain:

- If $x' \in X_1 \implies y' \in X_4 \implies y' \in X_1$. Of $x$ and $y$, only the literal $x$ is “true”.
- If $x' \in X_2 \implies y' \in X_3 \implies y' \in X_2$. Of $x$ and $\bar{y}$, only the literal $\bar{y}$ is “true”.

In conclusion, there is only one true literal per clause. \qed
• Conversely, we suppose that there is a truth assignment \( I : \mathcal{V} \rightarrow \{0, 1\} \), such that each clause in \( C \) has exactly one true literal.

Suppose that the true literal in clause \( C_i = (y \lor z \lor v) \) is \( r \). Therefore, the variable-task \( r' \) is processed at slot 3 on the same processor that executed clause-task \( C_i' \) and task \( r' \). Task \( r' \) (resp. \( r' \), \( z' \)) is allotted on the processor that is executed the clause-task \( D_i' \) corresponding to the clauses of length two in which \( r \) occurs (resp. \( y \), \( z \)). The clause-tasks \( D_i' \), \( j = 1, 2, 3, 4 \), are executed at slot \( j \) on the same processor. The \( 2^k \) other variable-tasks \( y' \), not yet scheduled, are processed at slot 3 on the processor that executed \( r' \) and \( r' \). The variable-task \( r' \) (resp. \( r' \), \( z' \)) is allotted at slot 2 (resp. 4 and 5) on the processor executing \( r' \) (resp. \( r' \)). We can observe that this scheduling is valid.

This concludes the proof of Theorem 6. □

We can deduce the following classical corollary:

Corollary 8. There is no polynomial-time algorithm for the problem \( \overline{P} \mid \text{prec} \); \( c_i = c \geq 2; p_i = 1 \) |\( C_{\text{max}} \) with a performance bound smaller than \( 1 + \frac{1}{2s+5} \) unless \( \mathcal{P} \neq \mathcal{NP} \).

Proof. The proof of Corollary 8 is an immediate consequence of the Impossibility Theorem, (see [5,6]). □

4.3. Total job completion time minimization

In this section, we will show that there is no polynomial-time approximation algorithm for the problem \( \overline{P} \mid \text{prec} \); \( c_i = c \geq 2; p_i = 1 \) |\( C_{\text{max}} \) with a performance guarantee bound less than \( 1 + \frac{1}{2s+5} \) unless \( \mathcal{P} \neq \mathcal{NP} \). This result is obtained by the polynomial-time transformation used to prove Theorem 2, and the gap technique (see [10]).

Theorem 9. There is no polynomial-time algorithm for the problem \( \overline{P} \mid \text{prec} \); \( c_i = c \geq 2; p_i = 1 \) |\( C_{\text{max}} \) obtained by reduction (see Theorem 2).

Proof. We suppose that there is a polynomial-time approximation algorithm denoted by \( A \) with a performance guarantee bound less than \( 1 + \frac{1}{2s+5} \). Let \( I \) be the instance of the problem \( \overline{P} \mid \text{prec} \); \( c_i = c \geq 2; p_i = 1 \) |\( C_{\text{max}} \) obtained by reduction (see Theorem 2).

Let \( I' \) be the instance of the problem \( \overline{P} \mid \text{prec} \); \( c_i = c \geq 2; p_i = 1 \) |\( C_{\text{max}} \) by adding \( x \) new tasks from an initial instance \( I \). In the precedence constraints, each group of \( x \) (with \( x > \frac{2c+6x+(c+4)n}{2s+6x+(c+4)n} \)) new tasks is a successor of old tasks (old tasks are from the polynomial-time transformation used in the proof of Theorem 2). We obtain a complete directed graph from old tasks to new tasks.

Let \( A(I') \) (resp. \( A^*(I') \)) be the result given by \( A \) (resp. an optimal result) on an instance \( I' \).

1. If \( A(I') < (2c+5)x(c+4)n \), then \( A^*(I') < (2c+5)x(c+4)n \). We can therefore decide that there exists a scheduling of an instance \( I' \) with \( C_{\text{max}} \leq c+4 \). Indeed, if no such schedule exists, it follows that, for every schedule, a task of an instance \( I' \) is executed after time \( (c+5) \). But, perhaps \( x \) tasks are executed after \( (2c+6) \) and so \( A(I') > (2c+6)(x-c) \) is impossible, indeed if \( (2c+6)(x-c) \geq (2c+5)x-c \), then \( x \leq \frac{(2c+6)(x-c)}{2s+6x+(c+4)n} \). This is in contradiction with the hypothesis, a contradiction with \( x > \frac{(2c+6)(x-c)}{2s+6x+(c+4)n} \). Thus, a schedule exists of length \( c+4 \).

2. We suppose that \( A(I') \geq (2c+5)x(c+4)n \). So, \( A(I') \geq (2c+5)x(c+4)n \) because an algorithm \( A \) is a polynomial-time approximation algorithm with a performance guarantee bound less than \( \rho < \frac{2c+6}{2s+5} \). There is no algorithm to decide whether the tasks from an instance \( I \) allow a schedule of length less than \( c+4 \).

Indeed, if such an algorithm exists, by executing the \( x \) tasks at time \( t = 4+2c \), we obtain a schedule with a completion time strictly less than \( (5+2c)x+(4+c)n \) (there is at least one task that is executed before time \( t = c+4 \)). This is a contradiction, since \( A^*(I') \geq (2c+5)x+(4+c)n \).

Therefore, if there is a polynomial-time approximation algorithm with performance guarantee strictly bounded by \( 1 + \frac{1}{2s+5} \), it can be used to distinguish, in polynomial time, the positive instances from negative instances to the problem \( \overline{P} \mid \text{prec} \); \( c_i = c \geq 2; p_i = 1 \) |\( C_{\text{max}} \) = \( c+4 \), thus providing a polynomial-time algorithm for a \( \mathcal{NP} \)-hard problem. Consequently, problem \( \overline{P} \mid \text{prec} \); \( c_i = c \geq 2; p_i = 1 \) |\( C_{\text{max}} \) does not possess an \( \rho \)-approximation, with \( \rho < 1 + \frac{1}{2s+5} \).

This concludes proof of Theorem 9. □

5. A polynomial time for \( C_{\text{max}} = c+2 \) with \( c \in \{2, 3\} \)

Theorem 10. The problem of deciding whether an instance of \( \overline{P} \mid \text{prec} \); \( c_i = c; p_i = 1 \) |\( C_{\text{max}} \) with \( c \in \{2, 3\} \) has a schedule with length of at most \( c+2 \), is solvable in polynomial time.

Proof. We will prove that, if there exists a schedule in \( (c+2) \) units of time, it will be easy to determine. W.l.o.g., we suppose that graph \( G = (V, E) \) is connected and \( |V| \geq 5 \).

For all \( x \in V \) such that \( |\Gamma^-(x)| > 0 \) and \( |\Gamma^+(x)| > 0 \), where \( \Gamma^-(x) \) (resp. \( \Gamma^+(x) \)) denotes the set of predecessors (resp. successors), we define the set \( M(x) = \Gamma^+(x) \cup \Gamma^-(x) \cup \{x\} \).
Note that all the tasks from an $M(x)$ must be executed on the same processor. If any $M(x)$ and $M(y)$ satisfies $M(x) \cap M(y) \neq \emptyset$, then the set $M(x) \cup M(y)$ will be considered.

For $c = 2$, if $|M(x)| = 3$, then, w.l.o.g., the source (resp. the sink) is scheduled at $t = 0$ (resp. at $t = 3$). If $|M(x)| = 4$, at most one source (resp. one sink) can have a successor (resp. a predecessor). This source (resp. sink) is executed at $t = 0$ (resp. at $t = 3$).

For $c = 3$, if $|M(x)| = 3$, the schedule is the same as that mentioned earlier. If $|M(x)| = 4$, and if there is only one source and one sink, we execute the source at $t = 0$ and the sink at $t = 4$. If two sources exist (the case of two sinks is symmetrical), then there is only one sink which will be executed at $t = 4$. At least one of these two sources, denoted by $s$, allows either no successor or only one isolated successor $z$ (with $|I^{-}(z)| = 1$). Thus, $s$ is scheduled at $t = 1$, the other source at $t = 0$, and the task $z$ (if it exists) at $t = 3$. If $|M(x)| = 5$, one source and one sink at most, accept neighbours in another $M(y)$, $y \neq x$, this source (resp. this sink) is scheduled at $t = 0$ (resp. $t = 4$). The other tasks must be executed as soon as possible. □

The previous strategy cannot be extended to the general case. Indeed, with values above $c = 4$, the combinatorial becomes very large. The challenge is to provide a general polynomial-time algorithm for $c = 2, 3$, etc. This concludes the proof of Theorem 10.

6. The approximation algorithm

After giving a threshold for any approximation algorithm, the challenge is to develop a polynomial-time approximation algorithm with non-trivial ratio. And to improve, in the presence of a restricted number of processors, the bound given by Rapine [19].

In this section, we will propose a new polynomial-time approximation algorithm with a non-trivial performance guarantee for $\{\text{prec}; c_{ij} = c \geq 2; p_{i} = 1|c_{\text{max}}\}$. Analysis of the worst-case behaviour yields a $2(c+1)^{3}$-approximation algorithm.

6.1. Introduction, notation and description of the method

**Notation:** We use $\sigma^{\infty}$ to denote the UET–UCT schedule, and $\sigma_{c}^{\infty}$ the UET-LCT schedule. Moreover, we use $t_{i}$ to denote (resp. $t'_{i}$) the starting time of task $i$ in the schedule $\sigma^{\infty}$ (resp. in the schedule $\sigma_{c}^{\infty}$).

**Principle:** We keep an assignment for the tasks given by a “good” feasible schedule on an unrestricted number of processors $\sigma^{\infty}$. We proceed with an expansion of the makespan, while preserving communication delays ($t'_{i} \geq t_{i} + 1 + c$) for two tasks, $i$ and $j$ with $(i, j) \in E$, processing on two different processors.

Let $G = (V, E)$ be a precedence graph. We consider the solution from a feasible schedule $\sigma^{\infty}$ provided by a $(4/3)$-approximation algorithm proposed by Munier and Konig [16]. A couple $\forall i \in V$. $(t_{i}, \pi)$ is provided where $t_{i}$ designates the starting time of the task $i$ and $\pi$ the processor on which the task $i$ is processed at $t_{i}$.

Now, we can determine a couple $\forall i \in V$. $(t'_{i}, \pi')$ on schedule $\sigma_{c}^{\infty}$. The starting time $t'_{i}$ is determined from the starting time $t_{i}$ as follows: $t'_{i} = d \times t_{i} = \frac{(c+1)}{2}t_{i}$ and, $\pi = \pi'$. The justification for the expansion coefficient is given below. An illustration of the expansion is given in Fig. 5.

6.2. Analysis of the method

**Lemma 11.** The coefficient of an expansion is $d = \frac{(c+1)}{2}$.

**Proof.** Consider two tasks $i$ and $j$ such that $(i, j) \in E$, which are processed on two different processors in the feasible schedule $\sigma^{\infty}$. The aim is to find a coefficient $d$ such that $t'_{i} = d \times t_{i}$ and $t'_{j} = d \times t_{j}$. Thus, after expansion, in order to respect the precedence constraints and communication delays, we must have $t'_{i} \geq t'_{i} + 1 + c$, therefore $d \times t_{i} - d \times t_{j} \geq c + 1$, $d \geq \frac{c+1}{t_{i} - t_{j}}$, $d \geq \frac{c+1}{t_{j}}$. It is sufficient to choose $d = \frac{(c+1)}{2}$.
Lemma 12. An expansion algorithm provides a feasible schedule for the problem denoted by \( \bar{P} \text{prec} \); \( c_j = c \geq 2 \); \( p_i = 1 \mid C_{\text{max}} \).

**Proof.** Obviously, it is sufficient to check that the solution given by an expansion algorithm produces a feasible schedule for the model UET–LCT. Consider two tasks \( i \) and \( j \) such that \((i, j) \in E\). We denote by \( \pi_i \) (resp. \( \pi_j \)) the processor on which the task \( i \) (resp. the task \( j \)) is executed in the schedule \( \sigma^\infty \). Moreover, we denote by \( \pi'_i \) (resp. \( \pi'_j \)) the processor on which the task \( i \) (resp. the task \( j \)) is executed in the schedule \( \sigma^\infty \). Thus,

- If \( \pi_i = \pi_j \) then \( \pi'_i = \pi'_j \). Since the solution given by Munier and König [16] gives a feasible schedule on the model UET–UCT, then we have \( t_i + 1 \leq t_j + \frac{2}{c+1} t'_i + 1 \leq \frac{2}{c+1} t'_j \); \( t_i + 1 \leq t'_i + \frac{c}{c+1} \leq t'_j \).
- If \( \pi_i \neq \pi_j \), \( \pi'_i \neq \pi'_j \). We have \( t_i \leq t_j + \frac{2}{c+1} t'_i + 2 \leq \frac{2}{c+1} t'_j \); \( t_i + (c+1) \leq t'_j \). \( \square \)

**Theorem 13.** An expansion algorithm gives a \( \frac{2(c+1)}{3} \)-approximation algorithm for the problem \( \bar{P} \text{prec} \); \( c_j = c \geq 2 \); \( p_i = 1 \mid C_{\text{max}} \).

**Proof.** We use \( C_{\text{max}} \) (resp. \( C_{\text{opt}} \)) to denote the makespan of the schedule computed by Munier and König (resp. the optimal value of a schedule \( \sigma^\infty \)). Similarly, we use \( C_{\text{max}} \) (resp. \( C_{\text{opt}} \)) to denote the makespan of the schedule computed by our algorithm (resp. the optimal value of a schedule \( \sigma^\infty \)).

We know that \( C_{\text{max}} \leq \frac{4}{3} C_{\text{opt}} \). Thus, we obtain \( \frac{C_{\text{max}}}{C_{\text{opt}}} = \frac{(c+1)}{3} \leq \frac{(c+1)}{3} \leq \frac{(c+1)}{3} \leq \frac{(c+1)}{3} \leq \frac{2(c+1)}{3} \). \( \square \)

**Remark 14.** This expansion method can be used for other scheduling problems.

### 7. Analysis of our results

We complete Table 1 (resp. 2) in order to create Table 3 (resp. 4).

We can now respond with a negative answer to the question put forth at the beginning of this article. Although we still do not know for the moment, the complexity for \( c_{\text{max}} = c + 2 \) and for \( c_{\text{max}} = c + 4 \) with \( c \geq 4 \), we can state that the variation, in the presence of significant communication delays, is of more than one unit of time. Thus, for large communication delays, scheduling is almost as difficult on an infinite number of processors as on a finite number. The difficulty comes from the type of communication and not from the type of processors.

Note that for the case \( c = 2 \), we showed that problem \( \bar{P} \text{prec} \); \( c_j = c \geq 2 \); \( p_i = 1 \mid C_{\text{max}} \) is \( \mathcal{NP} \)-complete. It is the same complexity result as the classic UET–UCT problem. Remember that a \( \frac{2}{3} \)-approximation algorithm exists for this problem. Using an expansion algorithm for \( c = 2 \), we obtain a 2-approximation algorithm. This raises an interesting question: *Can the ILP given for the UET–UCT problem in the presence of an unrestricted number of processors, can be extended to problem with \( c = 2 \) on unrestricted number of processors?*

Does considering the time shift from \( c = 1 \) to \( c = 2 \) lead to an additional way to find a good approximate solution, including a non-trivial performance guarantee?

### 8. Conclusion

**Figs. 1 and 2** are completed by the Figs. 6 and 7.

In this paper, we first proved that the problem of deciding whether an instance of \( \bar{P} \text{prec} \); \( c_j = c \geq 2 \); \( p_i = 1 \mid C_{\text{max}} \) has a schedule of length equal or inferior to \((c+4)\) is \( \mathcal{NP} \)-complete. This result is to be compared with the result of [11,1], which states that \( \bar{P} \text{prec} \); \( c_j = 1 \); \( p_i = 1 \mid C_{\text{max}} = 6 \) (resp. \( \bar{P} \text{prec} \); \( c_j = c \geq 3 \); \( p_i = 1 \mid C_{\text{max}} = c + 3 \)) is \( \mathcal{NP} \)-complete. Our results imply that there is no \( \rho \)-approximation algorithm with \( \rho < 1 + \frac{1}{2c+5} \), for the problem of minimizing the sum of completion times. Secondly, we establish that the problem of deciding whether an instance of \( \bar{P} \text{prec} \); \( c_j = c \); \( p_i = 1 \mid C_{\text{max}} \) with \( c \in [2, 3] \) has a schedule of length less than \((c+2)\) is solvable in polynomial time. We also propose a \( \frac{2(c+1)}{3} \)-approximation algorithm based on the notion of expansion.

**Conjecture 15.** We conjecture that the problem of deciding whether an instance of \( \bar{P} \text{prec} \); \( c_j = c \); \( p_i = 1 \mid C_{\text{max}} \) with \( c \geq 2 \) has a schedule of length at most \((c+3)\) is solvable in polynomial time.
Fig. 6. Main complexity results for unitary tasks for the minimization of the length of the schedule for homogeneous model [2,4,9,14–17,19,22].

Fig. 7. Main results for approximation algorithms for unitary tasks for the minimization of the length of the schedule for homogeneous model [2,4,9,14–17,19,22].

Table 4
New approximation results

<table>
<thead>
<tr>
<th>Machines</th>
<th>$c_i$</th>
<th>Approximation</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\bar{P}$</td>
<td>$c = 1$</td>
<td>$\rho \leq 4/3$</td>
<td>[14]</td>
</tr>
<tr>
<td>$\bar{P}$</td>
<td>$c = 1$</td>
<td>$\rho \leq 7/3$</td>
<td>[13]</td>
</tr>
<tr>
<td>$\bar{P}$</td>
<td>$c \geq 2$</td>
<td>2 for tree</td>
<td>[12]</td>
</tr>
<tr>
<td>$\bar{P}$</td>
<td>$c \geq 1$</td>
<td>$\theta(c)$</td>
<td>Theorem 13</td>
</tr>
<tr>
<td>$\bar{P}$</td>
<td>$c \geq 2$</td>
<td>$O(c)$</td>
<td>[19]</td>
</tr>
</tbody>
</table>

For Further reading

[8].

Acknowledgements

Thanks to our reviewers for the thorough review of this paper and many helpful comments and suggestions. Thanks to Michael Modjeska for helping us in improving the readability of this paper.

References