Tree scheduling with communication delays

Philippe Chretienne*

P. and M. Curie University, LITP Laboratory, Tour 55-65 E428, 4, place Jussieu, 75252 Paris Cedex 05, France

Received 10 December 1990; revised 18 April 1991

Abstract

This paper considers the problem of scheduling a tree-structured task system in a distributed environment with the goal of minimizing the makespan. Interprocessor communication delays are taken into account and task duplication is allowed. Furthermore, it is assumed that the number of processors is unlimited. It is shown that there is a polynomial-time algorithm for an outtree and that the intree case is NP-hard. In addition, a special case of intree is shown to be polynomially solvable.

1. Introduction

New scheduling problems arise from the efficient processing of task systems over distributed memory multiprocessors. They mainly differ from classical scheduling problems [6] because the precedence constraints (i.e. data transfers) between tasks that are not assigned the same processor involve communication times which have a significant effect upon the overall makespan. So there is a growing interest in knowing the computational complexity of these problems and in developing efficient algorithms to solve them either exactly or with performance guarantees.

Rayward-Smith has shown in [8] that the UET scheduling problem with unit communication times is NP-hard. In [5], the authors have given an heuristic that takes the interprocessor communication delays into account; they have determined an upper bound of the makespan of the generated schedule in terms of the number of processors, the optimal makespan without considering communication delays and the maximum communication delay in one chain. In [7] Papadimitriou and Yannakakis have shown that when duplication is allowed and there is no limitation on the number of processors, the scheduling problem with unit execution times and a common larger communication time is NP-hard; they also have developed an approximate algorithm

*Email: chretienne@litp.ibp.fr.
with performance ratio two for the more general case when the communication times depend on the broadcasting tasks. Chretienne and Colin have proved in [3] that if the communication times are smaller than the processing times, duplication is allowed and there is no limitation on the number of processors, the corresponding scheduling problem can be solved by an efficient polynomial algorithm. When duplication is not allowed and the precedence graph is an intree or an outtree, Chretienne has shown in [1] that if the communication times are smaller than the processing times and there is no limitation on the number of processors, the corresponding scheduling problem is efficiently solved by an $O(n^2)$ dynamic programming algorithm. In [2], it is shown that when duplication is not allowed and there is no limitation on the number of processors, the scheduling problem is NP-hard and special cases such as the FORK graph, the JOIN graph and some extensions of these graphs are shown to be polynomial.

The aim of this paper is to present new complexity results about tree scheduling with interprocessor communication delays and no limitation on the number of processors. It is shown that while scheduling an outtree is a simple polynomial problem if duplication is allowed, scheduling an outtree without duplication is an NP-hard problem as well as scheduling an intree with or without duplication. The analysis of the special case of an outtree we call the HARPOON graph yields those NP-hardness results. In Section 1, the scheduling problem is specified. Section 2 briefly presents the simple solution of the outtree case. Section 3 is devoted to the intree case, it is shown that no gain may be expected by creating duplicates, that the intree case is equivalent to the outtree case without duplication and that the HARPOON problem is NP-hard. Section 4 present a polynomial special case of the HARPOON problem. Some concluding remarks are given in the last section.

2. Problem specification

A generic instance of the scheduling problem is specified in terms of the following four parameters $(I, U, p, v)$, where $I = \{1, \ldots, n\}$ is a finite set of nodes (i.e. generic tasks), $G = (I, U)$ is the precedence graph ($G$ may have transitive directed edges), the natural number $p_i, i \in I$ is the processing time of node $i$ (whichever processor executes it) and the natural number $v_{ij}, (i, j) \in U$ is the communication time of the arc $(i, j)$ (i.e. the time of the data transfer from any copy of task $i$ to any copy of task $j$ if these two copies are not processed by the same processor).

A schedule $S$ of the problem $P = (I, U, p, v)$ is a finite set of assigned copies (where an assigned copy is a member of $I \times N \times N$ whose first component is the generic task from which the copy is issued, whose second component is the processor to which the copy is assigned and whose third component is the time at which the copy is processed) such that

1. for each node $i$ there is at least one copy $(i, \pi, t)$;
2. at any time, a processor executes at most one copy;
3. if $(i, j) \in U$, then any copy $(j, \pi, t)$ must have a supplier $(i, \pi', t')$ such that if $\pi = \pi'$ then $t \geq t' + p_i$ else $t \geq t' + p_i + v_{ij}$.
So, in a schedule, at least one copy of each generic task has to be processed; each copy of task \( i \) has to be supplied by one copy of every immediate predecessor of task \( i \) in \( G \); if one copy and its supplier are not assigned the same processor, the interprocessor communication delay must be taken into account.

Assuming no limitation on the number of processors, the objective is to determine a schedule whose makespan (i.e., the largest completion time of a copy) is minimal.

3. The outtree case

If the precedence graph \( G = (I, U) \) is an outtree, the solution of the scheduling problem is made particularly simple by the ability to create duplicates.

Let us first observe that if \( \gamma_i \) is the set of tasks on the path from the root to node \( i \), then \( P_i = (\sum_{k \in \gamma_i} p_k) - p_i \) is a lower bound of the starting time of any copy of task \( i \). We then define a schedule \( S^* \) any copy of which is processed at its lower bound. The schedule \( S^* \) uses as many processors as there are leaves in the outtree. Assume task \( i \) is a leaf and \( \pi \) is the corresponding processor, one copy of every task \( k \) of \( \gamma_i \) is assigned to \( \pi \) and processed at time \( P_k \) on that processor. The supplier of any copy (distinct from the root) assigned to processor \( \pi \) is its immediate predecessor on \( \pi \). The process is then repeated for each leaf (see Fig. 1) and yields the schedule \( S^* \). It is clear that \( S^* \) is an earliest schedule, that task \( i \) has as many duplicates as there are leaves in the sub-outtree whose root is \( i \) and that every copy of task \( i \) is scheduled at time \( P_i \). Notice that no matter how large the communication delays are, the minimal makespan is equal to \( \max_{i \in I} \{ P_i \} \).

4. The intree case

In this section we show that when the precedence graph is an intree the corresponding scheduling problem is NP-hard. In fact, we shall prove a stronger result by...
showing that one special case of an intree whose depth is two is itself an NP-hard problem. We shall get this result through the following three stages: the first stage shows that no gain may be brought by task duplication and transforms the intree instance into an equivalent outtree instance without duplication. The second step considers the special case when one broadcasting task sends data to the heading tasks of \( n \) distinct two-task paths and shows that the schedules of a dominant subset are in one-to-one correspondence with the 0/1 integer solutions of a linear inequality system. The last step proves that solving the KNAPSACK decision problem polynomially reduces to deciding whether the inequality system has a 0/1 integer solution.

4.1. No gain from duplication and the equivalent outtree instance

Let \( P = (I, U, p, v) \) be an instance of the scheduling problem where \( G = (I, U) \) is an intree. The following lemma shows that no gain may be brought by creating duplicates.

**Lemma 4.1.** The subset of schedules with no duplicates is a dominant subset.

**Proof.** Assume \( S \) is a schedule of \( P \) (with makespan \( M(S) \)) that does duplicate some tasks and let \( i \) be one duplicated task all of whose immediate successors are not duplicated (notice that at least the root satisfies this condition). If task \( i \) is the root, then one may select in \( S \) one copy of the root with the smallest completion time and remove from \( S \) all the other copies of the root. If task \( i \) is not the root, let \( j \) be the immediate successor of task \( i \) in \( G \) and \( c \) be the unique assigned copy of \( j \) in \( S \). The copy \( c \) has a \( i \)-supplier \( c' \) in \( S \) and all the other copies of task \( i \) may be removed from \( S \). In either case the makespan of the new schedule is at most \( M(S) \). The process is then repeated until we get a schedule without duplicates. \( \square \)

It is now straightforward to transform an instance \( P = (I, U, p, v) \) of the intree scheduling problem into an “equivalent” instance \( P' = (I, U', p, v') \) where \( G' = (I, U') \) is an outtree and duplication is not allowed. For this purpose, we define \( G' \) as the reverse graph of \( G \) and the communication time \( v_{ij} \) as \( v_{ji} \). \( G' \) is an outtree and any schedule with no duplicates \( S \) of \( P \) yields a schedule \( S' \) of \( P' \) with the same makespan simply by reading backwards the time diagram of \( S \) (i.e. keeping the same processor allocation and defining the starting time \( t_i(S') \) as \( M(S) - C_i(S) \), where \( C_i(S) \) is the completion time of task \( i \) in \( S \)). The converse transformation can be made from any schedule of \( P' \). This one-to-one correspondence between the two sets of schedules and Lemma 4.1 allows us to assert that the intree scheduling problem and the outtree case with no duplication are equivalent problems.

4.2. The Harpoon problem

It has been shown in [2] that scheduling the FORK graph (see Fig. 2) with no duplication is a polynomial problem. This section considers the HARPOON problem involving one root broadcasting data to the heading tasks of \( n \) distinct two-task paths. Fig. 2 illustrates a generic instance of the HARPOON problem.
The FORK graph

The HARPOON graph

for all i, the communication delay from the root to task i is larger than $\Sigma a_i + \Sigma b_i$.

Fig. 2. The FORK graph and the HARPOON graph.

It is further assumed that the communication times $v_{ij}$ from the root to the $n$ heading tasks are larger than the sum of all the processing times so that no gain may be brought by processing some heading tasks in parallel. So we restrict the search problem to the schedules such that all the heading tasks are assigned the same processor (by convention processor 0) as the root. We shall also ignore the root since in any schedule the root is the only task to be processed during the first $p_0$ time units.

We use the following convenient notation: $P = \{1, \ldots, n\}$ is the set of paths, $a_i$ (resp. $b_i$) is the communication time from task $i_1$ to task $i_2$, $A$ (resp. $B$) is the sum $\sum_{i=1}^{n} a_i$ (resp. $\sum_{i=1}^{n} b_i$) and $\Omega$ is the ordered list $(1, \ldots, n)$. If $S$ is a schedule, then $P_0(S)$ (resp. $P_0(S)$) is the set of paths whose tail tasks are (resp. are not) processed by processor 0. We finally assume that the path names are such that $b_1 + C_1 \geq b_2 + C_2 \geq \cdots \geq b_n + C_n$.

Thus, an instance $H$ of the HARPOON problem will be specified in terms of the parameters $(n, a, b, C)$, where

1. $n$ is a positive natural number,
2. $a = (a_1, \ldots, a_n)$, $b = (b_1, \ldots, b_n)$ and $C = (C_1, \ldots, C_n)$ are vectors whose components are natural numbers,
3. $b_1 + C_1 \geq b_2 + C_2 \geq \cdots \geq b_n + C_n$.

The next lemma defines a dominant subset.

**Lemma 4.2.** Let $H$ be an instance of HARPOON and $J$ be a subset of $P$. The schedule $S^*(J)$ specified by

(a) processor 0 executes successively the tasks $i_j, i \in J$ according to their order in $\Omega$, the tasks $i_j, i \in P - J$ in any order, and finally the tasks $i_j, i \in P - J$ in any order,
(b) the tasks \( i_2, i \in J \) are processed at their earliest starting times on distinct processors, is optimal within the subset of schedules such that \( P_1(S) = J \).

**Proof.** Let \( S \) be a schedule of \( H \) such that \( P_1(S) = J \). We first observe that the schedule \( S' \) we get from \( S \) by making processor 0 execute successively the tasks \( i_1, i \in J \) in the same order as in \( S \), the tasks \( i_1, i \in P - J \) in any order and finally the tasks \( i_2, i \in P - J \) in any order is at least as good as \( S \).

Assume now that in the schedule \( S' \) task \( i_2, i \in J \) is not the first task to be processed by processor \( k \) \( (k \neq 0) \). Then task \( i_2 \) may as well be processed on an unassigned processor at the same starting time or may be earlier. It follows that the schedule \( S'' \) we get from \( S' \) by processing the tasks \( i_2, i \in J \) at their earliest starting times on distinct processors is at least as good as \( S' \).

Suppose now that in the schedule \( S'' \) processor 0 executes successively the two tasks \( i_1 \) and \( j_1 \) \( (i, j \in J) \) and that \( i > j \). Consider the schedule \( S''' \) we get by exchanging task \( i_1 \) and task \( j_1 \) on processor 0 and processing the corresponding second tasks \( i_2 \) and \( j_2 \) at their earliest starting times (see Fig. 3).

From the definition of the list \( \Omega \), we have

\[
C_{i_2}(S''') \leq C_{j_2}(S''),
\]

(1)

\[C_{i_2}(S''') = C_{j_2}(S'') + b_i + c_i \leq C_{j_1}(S'') + b_j + c_j = C_{j_2}(S'').\]

(2)

It follows that \( S''' \) is at least as good as \( S'' \) and that the schedule \( S^*(J) \) is the best schedule such that \( P_1(S) = J \). \( \Box \)

Fig. 4 illustrates the structure of a dominant schedule \( S^*(J) \).

The preceding lemma shows that the problem complexity comes from selecting the path subset \( P_0(S) \). It also allows us to solve an instance \( H \) of the HARPOON problem by considering only those schedules whose structure satisfies the conditions (a) and (b). We denote by \( D(H) \) this dominant subset.

Let us now specify an instance \( H \) of the HARPOON decision problem by \( (n, a, b, C, M) \), where \( M \) is a natural number not less than \( \sum_{i=1}^{n} a_i \). We associate with
An instance of the HARPOON problem.
(ignoring the root)

\[ J = (1,4) \]
\[ \Omega = (1,2,3,4) \]

The best schedule \( S^*((1,4)) \).

Fig. 4. The structure of a dominant schedule.

such an instance \( H \) the linear inequality system \( LP01(H) \) defined as follows:

\[ x_i \in \{0, 1\}, \ i \in \{1, \ldots, n\}, \quad (3) \]
\[ A + \sum_{i=1}^{n} (1 - x_i)b_i \leq M, \quad (4) \]
\[ \sum_{j=1}^{k} a_j x_j + (b_k + C_k) x_k \leq M, \quad k \in \{1, \ldots, n\}. \quad (5) \]

Lemma 4.3 shows that the answer to the instance \( H \) is yes if and only if \( LP01(H) \) has a solution.

**Lemma 4.3.** Let \( H = (n, a, b, C, M) \) be an instance of the HARPOON decision problem, there is a one-to-one mapping between the solutions of \( LP01(H) \) and the schedules of \( D(H) \) whose makespan is at most \( M \).

**Proof.** Let \( S \) be a schedule of \( D(H) \) whose makespan is at most \( M \). The corresponding values \((\tilde{x}_1, \ldots, \tilde{x}_n)\) of the \( x_i \) variables are defined as follows: if \( i \in P_0(S) \) then \( \tilde{x}_i = 1 \) else \( \tilde{x}_i = 0 \). Let us show that \( \tilde{X} = (\tilde{x}_1, \ldots, \tilde{x}_n) \) is a solution of \( LP01(H) \). First it is clear from the definition of \( D(H) \) that \( A + \sum_{i=1}^{n} (1 - \tilde{x}_i)b_i \) is the time during which processor 0 is busy, so we have \( A + \sum_{i=1}^{n} (1 - \tilde{x}_i)b_i \leq M \).
Assume now that path $k$ is in $P_1(S)$. Then, from the structure of any schedule in $D(H)$ we know that the tasks $i_1, i \in P_1(S)$ that are processed by processor $0$ before task $k_1$ are precisely the tasks $i_1$ such that $i \in P_1(S)$ and $i < k$. It follows that $\sum_{j=1}^{k} a_j \bar{x}_j + (b_k + C_k) \bar{x}_k$ is the completion time of the task $k_2$ on its own processor. Thus, we have $\sum_{j=1}^{k} a_j \bar{x}_j + (b_k + C_k) \bar{x}_k \leq M$.

Suppose now that path $k$ is in $P_0(S)$; then $\bar{x}_k = 0$ and $\sum_{j=1}^{k} a_j \bar{x}_j$ is the time at which processor $0$ has completed the processing of the tasks $i_1$ such that $i \in P_1(S)$ and $i < k$, so we have

$$\sum_{j=1}^{k} a_j \bar{x}_j + (b_k + C_k) \bar{x}_k \leq M.$$

Conversely, let $\bar{X} = (\bar{x}_1, \ldots, \bar{x}_k)$ be a solution of $LP01(H)$. We define $J$ as $\{i | \bar{x}_i = 1\}$ and $S$ as one schedule of $D(H)$ such that $P_1(S) = J$. It is clear from inequality (2) that processor $0$ is idle after time $M$ and from the inequalities (3) that for every $k$ in $J$, the processor which executes the task $k_2$ is free after time $M$. It follows that $S$ is a schedule of $D(H)$ whose makespan is at most $M$.

4.3. Deciding whether $LP01(H)$ has a solution is NP-complete

It will be convenient in this section to denote $D. LP01(H)$ the decision problem that asks if the linear inequality system $LP01(H)$ has a solution. In this last step, we show that $D. LP01(H)$ is an NP-complete problem by polynomially reducing the KNAPSACK decision problem (which is known to be NP-complete [4]) to it.

Let us briefly recall that an instance of the KNAPSACK decision problem is specified by the following parameters $(m, \alpha, \beta, U, V)$, where $m$ is a positive integer, $\alpha = (\alpha_1, \ldots, \alpha_m)$ and $\beta = (\beta_1, \ldots, \beta_m)$ are two vectors whose components are nonnegative integers, $U$ is a nonnegative integer less than $\sum_{i=1}^{m} \alpha_i$, and $V$ is a nonnegative integer less than $\sum_{i=1}^{m} \beta_i$. The question is: does there exist a 0/1 integer solution satisfying the two following inequalities:

$$\sum_{i=1}^{m} \alpha_i x_i \leq U, \quad \sum_{i=1}^{m} \beta_i x_i \geq V.$$

Before deriving the polynomial transformation, let us observe that the answer to $D. LP01(H)$ is no if $M < A$ and give $LP01(H)$ the following convenient form where $h (h = M - A)$ is a nonnegative integer:

1. $(a_1 + b_1 + C_1)x_1 \leq A + h,$
2. $a_1 x_1 + (a_2 + b_2 + C_2)x_2 \leq A + h,$
3. $\ldots \leq \ldots,$
4. $(n - 1) a_1 x_1 + a_2 x_2 + \ldots + (a_{n-1} + b_{n-1} + C_{n-1})x_{n-1} \leq A + h,$
5. $(n) a_1 x_1 + a_2 x_2 + \ldots + (a_n + b_n + C_n)x_n \leq A + h,$
6. $(n + 1) b_1 x_1 + b_2 x_2 + \ldots + b_n x_n \geq B - h.$
The first \( n \) inequalities correspond to the inequalities (3) of \( LPO1(H) \) and the last equality \( (n + 1) \) corresponds to the inequality (2) of \( LPO1(H) \). We may now state the theorem.

**Theorem 4.4.** The decision problem \( D. LPO1(H) \) is NP-complete.

**Proof.** Let \((m, x, \beta, U, V)\) be an instance of the KNAPSACK decision problem, the corresponding instance of \( D. LPO1(H) \) is as follows:

1. \( n = m + 1; \)
2. \( a_i = x_i, \) \( i \in \{1, \ldots, n-1\} \) and \( b_i = \beta_i, \) \( i \in \{1, \ldots, n-1\}; \)
3. \( C_n = \sum_{i=1}^{n-1} x_i - U - 1; \)
4. \( h = \sum_{i=1}^{n-1} b_i - V, \) \( b_n = \sum_{i=1}^{n-1} b_i - V + 1; \) (notice that \( h = b_n - 1); \)
5. \( C_1 = \max \{ \sum_{i=1}^{n-1} b_i - \beta_1, C_n + b_n - \beta_1 \}; \)
6. \( C_i = C_{i-1} + \beta_{i-1} - \beta_i, \) \( i \in \{2, \ldots, n-1\}; \)
7. \( a_i = \max \{0, \beta_1 + C_1 - h\}; \)

where \( \sum_{i=1}^{n-1} x_i \) and \( \sum_{i=1}^{n-1} \beta_i \). Let us first show that all these values are nonnegative integers. This is clearly true for \( n, a_i, i \in \{1, \ldots, n-1\}, b_i, i \in \{1, \ldots, n-1\}, \)

\( h, b_n, C_1, a_n, \) and \( C_n. \) From (6), we get \( \forall i \in \{2, \ldots, n-1\}, C_i = C_1 - (\beta_i - \beta_{i-1}) \) and from (5), we have \( C_1 \geq \beta_i - \beta_{i-1}, \) \( i \in \{1, \ldots, n-1\} \), so \( C_i, i \in \{2, \ldots, n-1\} \) is a nonnegative integer. It remains to show that \( b_1 + C_1 \geq b_2 + C_2 \geq \cdots \geq b_n + C_n. \) From (6), we get

\[
b_1 + C_1 = b_2 + C_2 = \cdots = b_n + C_n-1
\]

and from (5) we have \( C_i \geq C_n + b_n - \beta_1 \) or equivalently \( C_n + b_n \leq C_1 + \beta_1 = C_n + \beta_n-1. \) So the preceding values constitute a valid instance of \( D. LPO1(H); \) moreover, they may be computed from the KNAPSACK instance in polynomial time.

Let us now show that the answers to the two corresponding instances are the same.

Suppose first that \( \bar{X} = \{\bar{x}_1, \ldots, \bar{x}_n\} \) is a solution of \( LPO1(H). \) We know from (4) that \( h < b_n, \) so we have \( \bar{x}_n = 1 \) since otherwise we would have

\[
\sum_{i=1}^{n} b_i \bar{x}_i = \sum_{i=1}^{n-1} b_i \bar{x}_i \leq \sum_{i=1}^{n-1} b_i = B - b_n < B - h
\]

and \( \bar{X} \) would not be a solution. The inequality \( (n) \) is satisfied by the solution \( \bar{X}, \) so we get from (3) and (4).

\[
\sum_{i=1}^{n} a_i \bar{x}_i \leq A + h - (a_n + b_n + C_n) - \sum_{i=1}^{n-1} b_i = U.
\]

The inequality \( (n + 1) \) is satisfied by the solution \( \bar{X}, \) so we get from (4):

\[
\sum_{i=1}^{n-1} b_i \bar{x}_i \geq B - h - b_n = V.
\]

It follows that \( (\bar{x}_1, \ldots, \bar{x}_{n-1}) \) is a solution of the KNAPSACK instance.

Let us now consider a solution \( \bar{X} = (\bar{x}_1, \ldots, \bar{x}_m) \) of the KNAPSACK instance. We show that the vector \( \bar{Y} = (\bar{x}_1, \ldots, \bar{x}_{m+1}), \) where \( \bar{x}_{m+1} = 1 \) is a solution of the \( LP01(H) \) instance.
From (6) and (7), we have for any $i \in \{1, \ldots, n-1\}$
\[ a_n + h \geq \beta_1 + C_1 = \beta_i + C_i. \]
It follows that
\[ A + h \geq (a_1 + \cdots + a_i) + (a_n + h) \geq (a_1 + \cdots + a_i) + (b_i + C_i); \]
from which we derive that the inequality $(i), i \in \{1, \ldots, n-1\}$ is satisfied by any vector of $\{0, 1\}^m$ and thus by $\bar{Y}$. Consider now the inequality $(n)$; since $\bar{X}$ is a solution of the KNAPSACK instance, we have
\[ \sum_{i=1}^{m} a_i \bar{x}_i \leq U \]
from which we derive using (3) and (4) that
\[ \sum_{i=1}^{n-1} a_i \bar{x}_i + (a_n + b_n + C_n) \bar{x}_n \leq U + (a_n + b_n + C_n) = A + h. \]
Thus, the inequality $(n)$ is satisfied. Consider finally the inequality $(n + 1)$; since $\bar{X}$ is a solution of the KNAPSACK instance, we have
\[ \sum_{i=1}^{m} b_i \bar{x}_i \geq V \]
from which we derive, using (4), that
\[ \sum_{i=1}^{n-1} b_i \bar{x}_i + b_n \bar{x}_n \geq V + b_n = B - h. \]
Thus, the inequality $(n + 1)$ is also satisfied and $\bar{Y}$ is a solution of the $D.LP01(H)$ instance.

Since verifying that a vector of $\{0, 1\}^n$ is a solution of an instance of $D.LP01(H)$ can obviously be done in polynomial time, it follows from the above polynomial reduction that $D.LP01(H)$ is an NP-complete problem. \(\square\)

5. A polynomial special case of HARPOON

In this section we consider the special case of HARPOON where all the $b_i$ values are equal. We denote by $\beta$ this common value and $(n, a, \beta, C)$ an instance of this special case. The proof that this scheduling problem is polynomial relies on the computation of a shortest $M$-feasible path with $q$ arcs in polynomial time.

Let $G = (V, E)$ be a directed acyclic graph whose edges have integer-valued (possibly negative) lengths. The length of the edge $(i, j)$ is denoted $l_{ij}$. The vertex $s$ is a source from which any other vertex is reachable, $q$ is a natural number less than $|V|$ and $M$ is a natural number. The path $e = (c_0, \ldots, c_q)$ is said to be $M$-feasible if for every $k$ in $\{0, \ldots, q\}$, the length of the subpath $(c_0, \ldots, c_k)$ is at most $M$. Fig. 5 illustrates a very simple example of this problem.
Lemma 4.5, which is a simple variant of the Bellman–Kalaba algorithm, allows to compute in polynomial time the length \( a_{ij}^{(q)}(M) \) of a shortest \( M \)-feasible path with \( q \) arcs from \( s \) to \( i \). The initial values for this lemma are as follows: \( a_{ij}^{(0)}(M) = 0 \) if \( i = s \) and \( +\infty \) otherwise.

Lemma 4.5. For any \( k \in \{1, \ldots, n-1\} \) and any vertex \( j \), the value \( a_{ij}^{(q)}(M) \) either equals \( +\infty \) if \( j = s \) or equals the minimum value of \( a_{ij}^{(k-1)}(M) + l_{ij} \) where the vertex \( i \) runs through the subset of the immediate predecessors of \( j \) in \( G \) such that \( a_{ij}^{(k-1)}(M) + l_{ij} \leq M \).

Proof. The proof is straightforward by induction on \( k \).

Fig. 6 illustrates such computations for a simple graph.

We now present the overall algorithm which solves an instance \((n, a, \beta, C)\) in polynomial time.

Theorem 4.6. The special case of HARPOON with equal \( b_i \)'s is polynomial.

Proof. We first fix the number \( q \) of paths in the subset \( J \) and show that the best dominant schedule \( S^*(J) \) such that \(|J| = q\) may be computed in polynomial time. We associate with the instance \( H = (n, a, \beta, C, M) \) the directed acyclic graph \( G(H) \) defined as follows (see Fig. 7): one node for each path and two extra nodes \( s \) and \( t \); for each path-node \( i \), there is an arc \((s, i)\) whose length is \( a_i + C_i \) and an arc \((i, t)\) whose length is \( \beta \); finally for each \( i,j \) such that \( 1 \leq i < j \leq n \) there is an arc \((i, j)\) whose length is \( a_j + C_j - C_i \). It is clear from the definitions of \( G(H) \) that there is a one-to-one mapping between the paths of \( G(H) \) with \( q + 1 \) arcs from \( s \) to \( t \) and the set of dominant schedules \( S^*(J) \) such that \(|J| = q\). If \((s, k^{(1)}, \ldots, k^{(q)}, t)\) is such a path, the corresponding subset \( J \) is \( \{k^{(1)}, \ldots, k^{(q)}\} \) and for any \( p \in \{1, \ldots, q\} \) the value of the subpath \((s, k^{(1)}, \ldots, k^{(p)})\) is \( a_{(p)}^{(1)} + \cdots + a_{(p)}^{(p)} + C_{(p)}^{(p)} \), which is precisely the time at which the processor executing the task \( k^{(p)} \) is free in \( S^*(J) \) (recall that the root is ignored). So there exists a schedule \( S^*(J)(|J| = q) \) of \( H \) whose makespan is at most \( M \) if and only if

(a) \( A + (n - q)\beta \leq M \),
(b) there is an \( M \)-feasible path with \( q \) arcs from \( s \) to \( t \) in \( G(H) \).
It follows from Lemma 4.3 that deciding whether such an $M$-feasible exists may be done in polynomial time.

Now the overall algorithm for an instance of HARPOON $H = (n, a, \beta, C)$ is as follows: we use binary search on $M$ with $M^- = A$ as the initial lower bound and $M^+ = A + nf$ as the initial upper bound. For each value of $M$, we only consider the $q$ values such that (a) is true and for each such value we examine whether there exists an $M = \text{feasible path with } q \text{ arcs from } s \text{ to } t \text{ in } G(H)$. This yields the minimum $M$ such that there exists a dominant schedule $S^*(J)$ with makespan $M$ in polynomial time. 

6. Concluding remarks

The scheduling problem specified by a general precedence graph, interprocessor communication delays and no limitation on the number of processors was already known not to become an easy problem when duplication is allowed except when the communication times are smaller than the processing times. This paper mainly shows that the special case of this problem when the precedence graph is an intree (and even an intree whose depth is at most two) is an NP-hard problem. Conversely, it appears that duplication is extremely efficient in case the precedence graph is an outtree since creating duplicates here cancels all the effects induced by the communication delays.
Acknowledgement

The author is grateful to the anonymous referees for their helpful remarks and comments.

References