A linear time algorithm to schedule trees with communication delays optimally on two machines

Marinus Veldhorst

RUU-CS-93-04
January 1993

Utrecht University

Department of Computer Science

Padualaan 14, P.O. Box 80.089,
3508 TB Utrecht, The Netherlands,
Tel. : ... + 31 - 30 - 531454

A linear time algorithm to schedule trees with communication delays optimally on two machines

Marinus Veldhorst

Technical Report RUU-CS-93-04
January 1993

Department of Computer Science
Utrecht University
P.O.Box 80.089
3508 TB Utrecht
The Netherlands

ISSN: 0924-3275

A linear time algorithm to schedule trees with communication delays optimally on two machines *

Marinus Veldhorst

Department of Computer Science, Utrecht University,
P.O. Box 80.089, 3508 TB Utrecht, The Netherlands,
E-Mail: `marinus@cs.ruu.nl`

Abstract

In this paper we prove that a minimum length schedule on 2 machines can be found in linear time for a set of n unit length tasks with a forest of intrees as precedence relations, and with unit interprocessor communication delays, while duplication of tasks is not allowed.

1 Introduction

For the implementation of a parallel program, considered as a collection of processes, on a parallel computer, one of the major problems is to determine which processor should execute which process and at what time; the purpose is that the moment of time that all processes have been executed (i.e., the execution of the parallel program has finished), occurs as early as possible. Besides the possibility that processes may have different execution lengths (possibly not exactly known in advance), processes might use (intermediate) results computed by other processes. Making these results available may require an additional (to the execution length) amount of time, the communication delay. During this communication delay processors may execute processes that have already obtained all necessary data.
The necessary communication delay is dependent of both the parallel program and the system on which it must be implemented. The parallel program with its input

*This research was done while the author was on sabbatical leave at the Department of Operations Research and Stochastics, Technical University of Eindhoven, Eindhoven, The Netherlands. Moreover it was partially supported by ESPRIT Basic Research Action No. 7141 (project ALCOM II: *Algorithms and Complexity*) and by EC Cooperative Action IC-1000 (project ALTEC: *Algorithms for Future Technologies*).

determines how much data must be sent from one process to the other processes. The parallel system has a different impact on the communication delay. Has it a shared memory or is it a distributed memory system with some interconnection network of processors? How much routing time is used if many data items must be made known simultaneously? Are special routing processors responsible for the routing, or must a processor temporarily uphold the execution of some process in order to help with the routing of data? Does the system allow duplication of tasks? In case of task duplication several processors will execute the same process, and this may lead to less data to be routed, and/or more evenly balanced routing problems. In that case we would expect to see a decrease in routing time, at the cost of an increase in execution time (used for the execution of processes).

In this paper we will deal with only a subclass of the above mentioned problems of assigning tasks (processes) to machines (processors).[1] We assume that task duplication is not allowed. The tasks as well as their execution lengths are given. The precedence relations are given as an directed acyclic graph on the tasks, and if task v needs data from task w, then w has to precede v and the communication delay is given as a positive number. We assume that this communication delay is only imposed if w and v are executed by different machines. We assume that processors to execute tasks are not involved in the routing of data. Hence it seems that we assume either a shared memory parallel system or a system with a complete processor interconnection network. Furthermore, preemption of tasks is not allowed, i.e., a task cannot be split and its pieces executed in nonconsecutive time intervals. Of course we assume that each machine has enough local memory to store all the data that must be kept available for tasks that this machine still has to execute. The number of machines can be finite or infinite.
To state it mathematically: we are given m identical machines and a set $V = \{v_1, \ldots, v_n\}$ of n tasks; with each task $v \in V$ is associated a positive integral number $p(v)$, its length; there is an acyclic graph $G = (V, A)$ and with each arc $(w, v) \in A$ is associated a non-negative integral number $c_{w \to v}$, its communication delay. A schedule S consists for each v in V of a 2-tuple $(t(v), m(v))$ of integers that satisfies the following two conditions:

C.1 $0 \leq t(v)$ and $1 \leq m(v) \leq m$ for each $v \in V$,

C.2 for all v and w in V with $v \neq w$ and $m(v) = m(w)$, the half open intervals $[t(v), t(v) + p(v))$ and $[t(w), t(w) + p(w))$ have an empty intersection.

We say that v is scheduled at time $t(v)$ on machine $m(v)$. S is a feasible schedule if in addition condition **C.3** is satisfied.

C.3 if $(w, v) \in A$ then:

$$t(v) \geq \begin{cases} t(w) + p(w) & \text{if } m(v) = m(w) \\ t(w) + p(w) + c_{w \to v} & \text{otherwise} \end{cases}$$

[1]We will switch here to terminology commonly used in the field of scheduling theory.

For such a feasible schedule S the completion time $C_{\max}(S)$ is defined as

$$C_{\max}(S) = \max_{v \in V}(t(v) + p(v)).$$

The problem is to find for given m, G, task lengths, and communication delays, a feasible schedule of minimum completion time. Such a schedule is called an optimal schedule. In the corresponding decision problem, we are given m, G, task lengths, communication delays, and a deadline D, and are asked to find out whether a feasible schedule S exists such that $C_{\max}(S) \leq D$. We call this problem the multiprocessor scheduling problem with communication delays, for short the MSCD problem.

In case $c_{w \to v} = 0$ for all $(w, v) \in A$, we obtain the classical multiprocessor scheduling problem, which is NP-complete (cf. [10]). For several special cases of the multiprocessor scheduling problem polynomial algorithms have been designed: G is a forest of intrees, $p(v) = 1$ for all v (cf. [6]); G is the complement of a chordal graph, $p(v) = 1$ for all v (cf. [7]); $m = 2$, $p(v) = 1$ for all v (cf. [4]). Other special cases have been proven NP-complete: G is a forest of intrees and outtrees, $p(v) = 1$ for all v (cf. [5]).

Obviously, the MSCD-problem is NP-complete; but compared with the multiprocessor scheduling problem, much less is known about the complexity of special cases. NP-complete or NP-hard are:

G arbitrary $p(v) = 1$ $c_{w \to v} = 1$ m unbounded $D = 6$	G is tree of depth 2 $p(v)$ arbitrary $c_{w \to v}$ arbitrary m unbounded	G bipartite $p(v) = 1$ $c_{w \to v} = 1$ m arbitrary $D = 4$
cf. [12]	cf. [3]	cf. [8]

Polynomial algorithms exist for:

G arbitrary $p(v) = 1$ $c_{w \to v} = 1$ m unbounded $D = 5$	G a forest of intrees $p(u)$ arbitrary $c_{w \to v}$ arbitrary m unbounded $p(u) \geq c_{w \to v}$	G arbitrary $p(v) = 1$ $c_{w \to v} = 1$ $D = 3$	G a forest of intrees $p(v) = 1$ $c_{w \to v} = 1$ m is fixed
cf. [12]	Time $O(n)$ (cf. [2])	cf. [8]	Time $O(n^{2m})$ (cf. [11]) $O(n^2)$ if $m = 2$ (cf. [9])

For a more extensive overview we refer to [13].

In this paper we will improve on the last special case that G is a forest of intrees, $p(v) = 1$ for each $v \in V$, $c_{w \to v} = 1$ for each $(w, v) \in A$, and $m = 2$. We will design a linear time algorithm for it, which is obviously optimal. Our approach differs fundamentally from the approach in [9] that leads to a quadratic time algorithm.

While in the latter, subtrees of tasks were scheduled based on their size, we use a list-scheduling approach. Unfortunately our algorithm cannot easily be adjusted to give optimal schedules in case $m = 3$ machines are available.

The remainder of this paper is organized as follows. First we will review some terminology. In section 2 we will use the general approach of list scheduling that yields feasible schedules, and prove that under certain conditions the algorithm runs in linear time. Then in section 3 we will concentrate on the creation of the list, such that the list-scheduling algorithm of section 2 yields an optimal schedule and runs in linear time.

Terminology Given two lists $H = (h_1, h_2, \ldots, h_k)$ and $E = (e_1, \ldots, e_p)$. h_1 is at the front of H, h_i occurs before (after) h_j in H if $i < j$ $(i > j)$. E is a sublist of H if there are indices $i_1 < i_2 < \cdots < i_p \leq k$ such that $h_{i_j} = e_j$ for each j $(1 \leq j \leq p)$.

Let $G = (N, B)$ be a directed graph with a set N of vertices and a set B of arcs. The indegree (outdegree) of a vertex v is the size of the set $\{w \in N : (w, v) \in B\}$ (the size of the set $\{w \in N : (v, w) \in B\}$). A path in G of length k $(k \geq 0)$ from vertex v_0 to vertex v_k is a sequence $(v_0, v_1, \ldots, v_k)$ of vertices such that $(v_{i-1}, v_i) \in B$ for all i $(1 \leq i \leq k)$. A cycle of length k $(k \geq 1)$ is a path of length k from a vertex v_0 to v_0. A directed graph $G' = (N', B')$ is a subgraph of G if $N' \subseteq N$ and $B' \subseteq B$. G' is the subgraph of G induced by N' (denoted by $G' = G(N')$), if G' is a subgraph of G, and $(v, w) \in B$ for $v, w \in N'$ implies $(v, w) \in B'$.
In an acyclic directed graph G, a vertex w is a predecessor if G contains a path from w to v (v is a predecessor of itself). v is then a successor of w. $Pred(v)$ denotes the set of predecessors of v. w is a direct predecessor of v if $(w, v) \in B$. Two vertices v and w are unrelated if w is neither a predecessor nor a successor of v. A topological order of G is a list L of vertices of G such that each vertex of G occurs in L after its predecessors.
A directed graph T is an intree if it does not contain any cycle, there is one vertex (the root) with outdegree 0, and all other vertices have outdegree 1. A directed graph $F = (V, A)$ is a forest of intrees if it does not contain any cycle and each vertex has outdegree at most 1. For each vertex r in F with outdegree 0, the subgraph induced by all vertices for which a path to r exists, forms an intree. A leaf in F is a vertex with indegree 0. In a forest a direct predecessor w of v is also called a son, and v is then the father of w. Two vertices are brothers if they have the same father. In this paper we consider roots to be brothers (they have the same imaginary superroot).

For a given acyclic directed graph $G = (N, B)$ a schedule S consists of a 2-tuple $(t(v), m(v))$ for each $v \in N$ that satisfies the conditions **C.1** and **C.2**. A partial schedule consists of the assignment of 2-tuples satisfying **C.1** and **C.2**, to a subset of the tasks. A partial schedule is feasible if the assigned 2-tuples satisfy **C.3**, and for each task with a 2-tuple each of its predecessors has also been assigned a 2-tuple. A task v is ready with respect to a (partial) schedule if at time t the following two conditions hold:

C.4 for all direct predecessors w of v we have $t(w) + p(w) \leq t$,

C.5 for at least $indegree(v) - 1$ direct predecessors w of v, we have $t(w) + p(w) + c_{w \to v} \leq t$.

Condition **C.4** states that all predecessors of v must have been executed at time t, and **C.5** states that all data that v needs can be available at some machine.

2 The list-scheduling approach

In this and the following section we consider n tasks in a forest $F = (V, A)$ of intrees, where $p(v) = 1$ for all $v \in V$ and $c_{w \to v} = 1$ for all $(w, v) \in A$. We develop a linear (i.e., $O(n)$) time algorithm to find an optimal schedule for F on $m = 2$ machines, but we will try to formulate and prove theorems to hold for arbitrary m. The conditions **C.4** and **C.5** reduce for these weights and communication delays to:

C.4′ for all direct predecessors w of v we have $t(w) + 1 \leq t$,

C.5′ for at least $indegree(v) - 1$ direct predecessors w of v, we have $t(w) + 2 \leq t$.

The list-scheduling approach to find good schedules consists of two stages. In the first stage one creates a list L of tasks. In the second stage one scans for increasing time steps t the list L (starting at the front) to find as many (but not more than m) ready (with regard to the partial schedule obtained so far) tasks and tries to schedule them on the available machines at time t. We will modify the list-scheduling approach slightly in order to deal with the communication delays; and moreover prove that stage 2 can be implemented to run in $O(mn)$ time, provided that L is a topological order of F. Then in the next section we will show how to create in linear time a specific topological order that leads to an optimal schedule in case $m = 2$.
We modify stage 2 of the list-scheduling approach in such a way that condition **C.4′** implies **C.5′**. This can be done by preventing two brothers to be scheduled at the same time while all their brothers have been scheduled earlier. In figure 1 the time assignment of the modified stage 2 is given, and in figure 2 an example is given which shows that the modification seems to be really necessary.

Without proof we state:

PROPOSITION 1 *Suppose each vertex v has been assigned a time $t(v)$ by algorithm Tassign. If condition* **C.4′** *holds for v with $t = t(v)$, then* **C.5′** *holds also for v and t.*

DEFINITION 1 *If during the execution of algorithm Tassign a task v is assigned to the variable x_i for some i at some time t, then v is an i^{th}-*choice task.

```
(1)  $time := 0$;
(2)  while $L \neq \emptyset$
(3)  do    for $i := 1$ to $m$
(4)        do  Let $x_i$ be the first ready task in $L$ such that
(5)                $indegree(father(x_i)) \neq 1$ or
(6)                $(x_0, x_1, \ldots, x_{i-1})$ does not contain a brother of $x_i$;
(7)            if    such an $x_i$ exists
(8)            then  $t(x_i) := time$;  delete $x_i$ from $L$;
(9)                  $indegree(father(x_i)) - := 1$
(10)           endif ;
(11)       enddo ; $time := time + 1$
(12) enddo ; $C_{\max} := time$
```

co The statements using $father(x_i)$ must be properly adjusted in case x_i is a root. **co**

Figure 1: Algorithm *Tassign* for the assignment of schedule times to tasks.

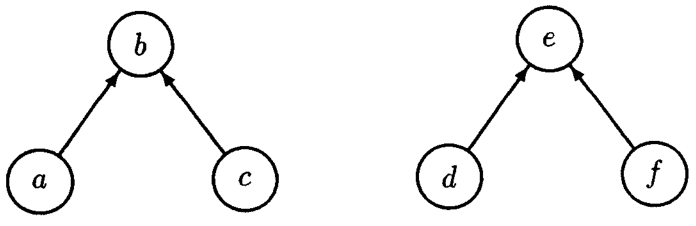
Actually algorithm *Tassign* assigns only time units to tasks, but not machines to tasks.

PROPOSITION 2 *Suppose each vertex v has been assigned a time $t(v)$ by algorithm Tassign. Then in $O(n)$ time each task v can be assigned a machine $m(v)$ such that the set of 2-tuples $(t(v), m(v))$ constitute a feasible schedule.*

Proof. First we will present the algorithm for the machine assignment. The i^{th}-choice task at time $t = 0$ is assigned to machine i. Suppose all tasks scheduled before time t are assigned to a machine. A task v scheduled at time t with a direct predecessor w at time $t - 1$, is assigned to machine $m(w)$. The other tasks scheduled at time t are assigned to the remaining machines.
Obviously this algorithm runs in linear time, provided the right information is stored with the tasks during the execution of algorithm *Tassign*. No two tasks can be assigned to the same machine at time t because in an intree no two tasks have predecessors in common. Moreover no task v scheduled at time t can be assigned to two machines, because by the algorithm *Tassign* v has at most one direct predecessor at time $t - 1$.
It is obvious that the schedule obtained is feasible. Q.E.D.

Now we will work onto the main result of this section that the time assignment runs in $O(nm)$ time in case L is a topological order of F. Therefore, from now on we assume that L satisfies this property. With $L^{(t)}$ we denote the list L as it is in algorithm *Tassign* at the moment that the value t is assigned to the variable *time*. Hence $L^{(0)} = L$.

F :



$L : a\ c\ d\ f\ b\ e$

Scheduling:

Machine 1:	a	d	b	e
Machine 2:	c	f		

Unmodified stage 2.

Machine 1:	a	c	b
Machine 2:	d	f	e

With modified stage 2.

Figure 2: Example of modified stage 2 of list-scheduling algorithm.

LEMMA 3 *At each time t, the first-choice task is at the front of $L^{(t)}$.*

Proof. Let x be at the front of $L^{(t)}$. Because L is a topological order, all predecessors of x occur in L before x and must have been deleted from L at some time before t. Hence x satisfies **C.4′**, and by Proposition 1 also **C.5′**. By definition x is ready at time t. Q.E.D.

As a consequence we have that the sequence of first-choice tasks (for increasing time) forms a sublist of L, and hence the searches for the first-choice tasks together take only $O(n)$ time.

LEMMA 4 *Suppose $(x_1, \ldots, x_p)$ $(p \leq m)$ are unrelated tasks such that x_j $(1 \leq j \leq p)$ is a j^{th}-choice task scheduled at time t. Let u be an i^{th}-choice task scheduled before time t. Then u occurs in L before x_i.*

Proof. For $i = 1$, the lemma holds by Lemma 3. Because the obtained schedule S is feasible, we have that at each time $t' < t$ a predecessor y_j of x_j is ready. With $x_1, \ldots, x_p$ unrelated, the tasks $y_1, \ldots, y_p$ are also unrelated, and $y_i = x_i$ or occurs in L before x_i. Moreover, x_j occurs in L after x_{j-1} $(2 \leq j \leq p)$, and hence, there are at least i ready tasks in $L^{(t')}$ before x_i (x_i was not chosen at time t'). Thus the scan for the i^{th}-choice task in $L^{(t')}$ encountered a task z_i occurring before x_i. Q.E.D.

COROLLARY 5 *The sequence of i^{th}-choice tasks is a sublist of L.*

COROLLARY 6 *If at time $t > 0$ p tasks are scheduled, then at least p tasks are scheduled at time $t - 1$.*

THEOREM 7 *If L is a topological order of the tasks in the forest of intrees, then algorithm $Tassign$ can be implemented to run in $O(nm)$ time.*

Proof. With the previous lemma, it is enough in algorithm $Tassign$ to keep for each i $(1 \leq i \leq m)$ a pointer for the scan for the i^{th}-choice task. The scan for choice i at time t proceeds from where the scan for choice i at time $t-1$ finished. Hence the scans for the i^{th}-choice tasks together take $O(n)$ time for each i. The theorem follows. Q.E.D.

3 Finding the right topological order

In this section we want to design an efficient algorithm that creates a topological order L such that the application of algorithm $Tassign$ to L yields an optimal schedule. Unfortunately we were only able to prove optimality for $m = 2$, and we found a counter example for the case of $m = 3$ machines.

Like Hu (cf. [6]) did for the case of no communication delay, we create a list in which tasks are ordered according to their distance to the root. In case of no communication delays, this distance is just the number of arcs. It gives a sharp lower bound to the difference between the moments of time that a task and its root successor are scheduled. Incorporating communication delays the concept of distance should be adjusted not only to comprise the number of intermediate tasks, but also the amount of time required for communication; and it should give the precise schedule times in case of an unbounded number of machines.

Several researchers (cf. [1], [2]) have published algorithms to find an optimal schedule of a forest when an unbounded number of machines is available. Most of them schedule each task also as early as possible, and hence compute for each task v the value $ept(v)$ defined as:

DEFINITION 2 *Let F be a forest of intrees, and v a task of F. Then the* earliest processing time *$ept(v)$ of v is the time that v is scheduled in an optimal schedule of $F(Pred(v))$ in case of $m = \infty$ machines.*

The values $ept(v)$ can be computed in $O(n)$ time according to the following recurrence relation

$$ept(v) = \begin{cases} 0 & \text{if } v \text{ is a leaf} \\ 1 + ept(w_0) & \text{if } v \text{ has exactly one son } w_0 \text{ such that } ept(w) \leq \\ & ept(w_0) \text{ for all sons } w \text{ of } v \\ 2 + ept(w_0) & \text{if } v \text{ has at least to different sons } w_0 \text{ and } w_1 \text{ such} \\ & \text{that } ept(w) \leq ept(w_0) = ept(w_1) \text{ for all sons } w \text{ of} \\ & v \end{cases}$$

Next we assign to each task a so-called *scheddist*-label. It comprises a distance concept between a task and an imaginary root that has all roots of F as sons. Obviously this imaginary root has *scheddist*-label 0. If a non-leaf task v has *scheddist*-label d then exactly one son w_0 of v has $scheddist(w_0) = 1 + scheddist(v)$ and moreover $ept(w_0) \geq ept(w)$ for all sons w of v. Observe that the *scheddist*-labeling is not unique.

DEFINITION 3 *Let v be a non-leaf in F. Task w is the* eldest son *of v if w is a son of v and $scheddist(w) = 1 + scheddist(v)$. Task u is the* eldest brother *of u' if u is the eldest son of u'.*
(Roots are considered to have a common imaginary father.)

The basic idea is more or less to schedule tasks in decreasing order of *scheddist*-label, and in this way an eldest son is scheduled later than its brothers. For a correct proof that the schedule is optimal, we use a list L with an additional property.

> Let v and v' be tasks. Let u be the eldest brother of v if v is not the eldest son of its father; in the other case let u be the father of v. u' is defined similarly with respect to v'. If v occurs before v' in L, then u occurs before u' in L.

In figure 3 we give the algorithms for the creation of the list L. It is left as an exercise to the reader to verify that the list L as created has indeed the above property.
The equivalent of the longest path (in case of no communication delays) is formed by a longest sequence of tasks that starts with a leaf and then proceeds with the eldest brother or the father.

DEFINITION 4 *A sequence $(x_k, x_{k-1}, \ldots, x_0)$ of different tasks is a* brother path *of length k from x_k to x_0 if for each i $(1 \leq i \leq k)$ x_{i-1} is the father or the eldest brother of x_i, and moreover $scheddist(x_i) = 1 + scheddist(x_{i-1})$.*

LEMMA 8 *Let x be a task such that $ept(x) = k$. Then there is a brother path P to x with length k. If $k > 1$, then the one but last task in P is the eldest son of x.*

Proof. We prove this by induction on k. The lemma obviously holds for $k = 0$. Now let $k > 0$, and suppose the lemma holds for all tasks y with $ept(y) < k$. Let x be a task with $ept(x) = k$. Let x_1 be the eldest son of x. Then $scheddist(x_1) = 1 + scheddist(x)$.
Case 1. $ept(x_1) = ept(x) - 1$. Then $ept(x_1) = k - 1$ and by the induction hypothesis applied to x_1 there is a brother path $(x_k, x_{k-1}, \ldots, x_1)$ to x_1. Extending this sequence with x yields a required brother path.
Case 2. $ept(x_1) = ept(x) - 2$. Then x_1 has a brother x_2 with $ept(x_2) = ept(x_1)$ and $scheddist(x_2) = 2 + scheddist(x)$. Apply the induction hypothesis to x_2, and extend the brother path ending at x_2 with x_1 and x. This yields a brother path to x of length k. Q.E.D.

Let $v_1, v_2, \ldots, v_n$ be a topological order of $F = (V, A)$. Assume that for each task v $ept(v)$ and the indegree $indegree(v)$ have already been computed.

```
co Assign to each task a $scheddist$-label co

Let $x_0$ be a root with largest $ept$-value;  $scheddist(x_0) := 1$;
for all roots $x$ with $x \neq x_0$   do   $scheddist(x) := 2$  enddo ;
for $i := n$ downto 1
do  if     $v_i$ has an unlabeled son
    then  Let among all sons of $v_i$ son $w_0$ have a maximum $ept$-value.
          $scheddist(w_0) := 1 + scheddist(v_i)$;
          for all $w$ with $w \neq w_0$ and $(w, v_i) \in A$
          do     $scheddist(w) := 2 + scheddist(v_i)$   enddo
    endif
enddo

co Creation of list $L$ co

Let $Broth(x_0) = \{x \in F \ : \ x$ is a root and $x \neq x_0\}$.
$L := \emptyset$;  Insert $x_0$ in $L$;  Let $ptr$ point to the last task in $L$;
while $ptr \neq nil$
do     Suppose $ptr$ points to task $y$;
       Insert all tasks of $Broth(y)$ at the front of $L$;
       if      $y$ has a son in $F$
       then  Let $z_0$ be a son of $y$ such that $scheddist(z_0) = 1 + scheddist(y)$;
             Insert $z_0$ at the front of $L$;
             $Broth(z_0) := \{z \in F \ : \ (z, y) \in A$ and $z \neq z_0\}$
       endif ;
       if      $y$ has a left neighbor $y'$ in $L$
       then  set $ptr$ to $y'$   else  set $ptr$ to $nil$
       endif
enddo
```

Figure 3: The algorithm *CreateList* to create the list L.

PROPOSITION 9 *A brother path of F is a sublist of L.*

Proof. Left to the reader. Q.E.D.

Now we come to a fundamental lemma, necessary for the proof that the obtained schedule is optimal in case of $m = 2$ machines.

LEMMA 10 *Let S be a schedule of a forest F, obtained by applying the algorithms CreateList and Tassign to F. Suppose there is a time interval $[t-k, t]$ of k time*

units ($k > 1$) in which $a_k, a_{k-1}, \dots, a_1$ and $b_k, \dots, b_1$ are the first-choice and second-choice tasks, respectively. Suppose that for each i ($2 \le i \le k$) b_i occurs in L after a_{i-1}.
Then $(a_k, a_{k-1}, \dots, a_1)$ is a brother path and moreover $(a_{k-1}, a_{k-2}, \dots, a_1)$ is a path in F.

Proof. We will assume that either $t - k = 0$ or the time interval $[t-k-1, t]$ does not satisfy the requirements of the lemma. (If the latter would occur, we could prove the lemma for $[t-k-1,t]$ and the subsequence $(a_k, \dots, a_0)$ is then a path in F with consecutive decreasing *scheddist*-values, and hence satisfies the lemma.) Let, in case $t > k$, a_{k+1} and b_{k+1} be the first-choice and second-choice tasks at time $t-k-1$, respectively. By taking k as large as possible, b_{k+1} occurs in L before a_k. Because the first-choice tasks as well as the second choice tasks form a sublist of L (lemma 4), we have that a_{k+1} and a_k are consecutive in $L = L^{(0)}$. Similarly, a_i and a_{i-1} are consecutive at the front of $L^{(t-i)}$ for $2 \le i \le k$. Hence, during the scan of $L^{(t-i)}$ in algorithm *Tassign*, a_{i-1} is checked whether it can be used as second-choice task, and rejected. a_{i-1} must be the father or last remaining brother of a_i. We know even more.

Claim 1. a_{i-1} is the father or the eldest brother of a_i.
Proof of Claim. Suppose that a_{i-1} is a brother of a_i. Let v be the eldest brother of a_i. v must be unequal to a_i because otherwise the list of first-choice tasks would not be a sublist of L. Suppose $v \ne a_{i-1}$. a_{i-1} is the last remaining brother of a_i in L at time $t-i+1$. Hence, v is scheduled at some time $t' \le t-i$. a_i and a_{i-1} are unrelated tasks, and therefore at time t' two unrelated predecessors of a_i and a_{i-1} are ready. These predecessors occur in L before v. Thus v would not be chosen at time t'. With contradiction we have proved the claim.
End of proof of claim.

Claim 2. If a_i is the father of a_{i+1} then $ept(a_i) \ge k-i$, otherwise $ept(a_i) \ge k-i-1$.
Proof of Claim. By induction over i. Obviously the claim holds for $i = k$ and $i = k-1$. Assume that the claim holds for $k, k-1, \dots, i+1$ ($1 \le i \le k-2$).
<u>Case 1.</u> a_i is the father of a_{i+1}. If a_{i+1} is the father of a_{i+2}, then $ept(a_i) \ge 1 + ept(a_{i+1}) \ge k-i$ by induction applied to a_{i+1}. If a_{i+1} is the eldest brother of a_{i+2}, then a_{i+2} is the father of a_{i+3} or $k = i+2$. By application of the induction hypothesis to a_{i+2} we get $ept(a_{i+2}) \ge k-i-2$. If $ept(a_{i+1}) = ept(a_{i+2})$ then $ept(a_i) \ge 2 + ept(a_{i+2}) \ge k-i$; otherwise $ept(a_i) \ge 1 + ept(a_{i+1}) \ge 2 + ept(a_{i+2}) \ge k-i$.
<u>Case 2.</u> a_i is the eldest brother of a_{i+1}. Then a_{i+1} is the father of a_{i+2} and by induction applied to a_{i+1} we have $ept(a_i) \ge 1 + ept(a_{i+1}) \ge k-i$.
End of proof of claim.

Claim 3. $(a_{k-1}, \dots, a_1)$ is a path in F.
Proof of Claim. Assume the claim does not hold. We will prove the claim by contradiction. Let k_0 ($2 \le k_0 \le k-1$) be the largest index such that a_{k_0} is not a son

of a_{k_0-1}. Such a k_0 exists by assumption. By claim 1 the eldest brother v of a_{k_0} must be task a_{k_0-1}, and a_{k_0} must be the father of a_{k_0+1}. By claim 2 we have that $ept(a_{k_0}) \geq k - k_0$ and thus $ept(v) \geq k - k_0$. By lemma 8 there is a brother path $(v_{k-1}, v_{k-2}, \ldots, v_{k_0}, v)$ in which v_{k_0} is the eldest son of v. By construction of L, v_{k_0} occurs in L between a_{k_0+1} and a_{k_0}, and in general v_i ($k_0 \leq i \leq k-1$) occurs in L after a_{i+1}. Because a_k and a_{k-1} are consecutive in $L = L^{(0)}$, v_{k-1} occurs after a_{k-1}, and all tasks $v_{k-1}, \ldots, v_{k_0}$ are scheduled not before time $t - k$. v_{k_0} must be unequal to b_{k_0} (by the conditions of the lemma), and with lemma 2, v_{k_0} must have been scheduled before time $t - k_0$. Hence, the tasks $v_{k-1}, \ldots, v_{k_0}$ must have been scheduled as second-choice tasks at times $t-k, t-k+1, \ldots, t-k_0-1$, that is in $k - k_0$ time slots. Thus $b_k = v_{k-1}, b_{k-1} = v_{k-2}$, etc. With v_{k-1} occurring in L after a_{k-1} and v_{k_0} occurring in L between a_{k_0+1} and a_{k_0}, and by the pigeon-hole principle, there must be an $i_0 \geq k_0$ such that v_{i_0+1} and v_{i_0} occur in L both between a_{i_0+1} and a_{i_0}. As a consequence a_{i_0+1} is not the eldest son of a_{i_0} and it must have an eldest brother $w \neq a_{i_0+1}$. However, no room is left for the scheduling of w. It cannot have been scheduled before time $t - k$ because it occurs in L after a_k. With $i_0 \geq k_0$, w must be unequal to v but it is a predecessor of v. Hence we have to conclude that the schedule produced by our algorithm is not feasible, contradicting lemma 2.
End of proof of claim.

In order to conclude that $(a_k, \ldots, a_1)$ is a brother path, we only have to show that they have consecutive *scheddist*-values.

Claim 4. For each i ($1 \leq i < k$) the equality $scheddist(a_{i+1}) = 1 + scheddist(a_i)$ holds.
Proof of claim. The proof of this claim is similar to the proof of claim 3. Hence, assume that the claim does not hold, and let k_0 ($1 \leq k_0 < k$) be the largest index such that $scheddist(a_{k_0+1}) = 2 + scheddist(a_{k_0})$. Then a_{k_0+1} is a son of a_{k_0} but not the eldest son. Let v be the eldest brother of a_{k_0+1}. With claim 2 $ept(v) \geq ept(a_{k_0+1}) \geq k - k_0 - 1$ and there is a brother path $P = (v_{k-k_0-1}, \ldots, v_1, v_0 = v)$ in which v_1 is the eldest son of v. All tasks of P are scheduled not before time $t - k$. On the other hand v must be scheduled before a_{k_0}, and it cannot be scheduled at the same time as its brother a_{k_0+1} (by communication requirements). Hence the $k - k_0$ tasks of P must have been scheduled as second-choice tasks at $k - k_0 - 1$ slots at times $t-k, \ldots, t-k_0-2$, which is impossible. Contradiction.
End of proof of claim.
This completes the proof of the lemma. Q.E.D.

LEMMA 11 *Suppose the schedule S satisfies the conditions of lemma 10; and let k be such that the second-choice task at time $t-k-1$ occurs in L before a_k. Then all tasks scheduled before time $t-k$ are predecessors of the father of a_k.*

Proof. Suppose the lemma does not hold. Let x be a task scheduled before time $t - k$. Let $(x = x_p, \ldots, x_1)$ be the brother path from x to a task x_1 such that x_1 is a

root in F or $x_1 = a_i$ for some a_i a proper successor of the father of a_k, and x_2 is not a proper successor of a_k. There must be a q such that x_q and x_{q-1} occur before and after a_k in L, respectively. Then, by construction of L, x_{q-1} occurs in L between a_k and a_{k-1}. This contradicts the fact that a_k and a_{k-1} are consecutive in $L = L^{(0)}$.
Q.E.D.

With these two lemmas it seems that under the conditions of lemma 10 the subgraph of F induced by $a_1, \ldots, a_k$ and all predecessors of the father of a_k is scheduled optimally.

THEOREM 12 *Application of the algorithms CreateList and Tassign together gives an optimal schedule for each forest F of intrees on $m = 2$ machines.*

Proof. Suppose the theorem does not hold, and F is a forest for which algorithms *CreateList* and *Tassign* together yields a suboptimal schedule S. Let $\lambda = C_{\max}(S)$. An optimal schedule for F has completion time $\lambda_{\text{opt}} < C_{\max}(S)$ and hence, S has $s \geq 2$ empty slots. By lemma 4 these empty time slots occur at times $\lambda - s, \ldots, \lambda - 1$. In these time slots there are no second-choice tasks.
Consider these empty slots as empty tasks that are added at the end of L. Let a_i and b_i be the first-choice and second-choice tasks, respectively, scheduled in S at time $\lambda - i$. b_1 and b_2 are empty tasks. Let k be the largest number such that b_k occurs in L after a_{k-1}. Obviously $k \geq s \geq 2$.

Claim. $k > 2$.
Proof of claim. Suppose $k = 2$. Then the optimal schedule for F contains no idle time slots at all. Hence, F consists of at least 2 trees. Obviously x_1 is a root, but also x_2 is a root, because otherwise the sublist of second-choice tasks (except for the empty tasks) would end with a root that would occur in L after x_2, and k would have been chosen larger. So with x_2 and x_1 both roots, x_1 must occur in L after x_2, and therefore, x_1 is considered to become the second-choice task at time $\lambda - 2$. It would not have been rejected. Contradiction.
End of proof of claim.
With lemma 10 we know that $(a_k, \ldots, a_1)$ is a brother path, and moreover that $(a_{k-1}, \ldots, a_1)$ is a path in F, and by lemma 11 all tasks a_i and b_i $(k + 1 \leq i \leq \lambda)$ are predecessors of the father of a_k. Let V' be defined as

$$V' = \{a_i \ : \ 1 \leq i \leq k - 2\} \cup \mathit{Pred}(\mathit{father}(x_k))$$

and let $F' = F(V')$. We will show that each schedule of F' on $m = 2$ machines has completion time λ, and hence each schedule for F has completion time λ, and as a consequence S is an optimal schedule.
Case 1. x_{k-1} is the father of x_k. All predecessors of a_{k-1} (except a_k and a_{k-1} are scheduled before time $\lambda - k$. By lemma 11 no other tasks are scheduled before $\lambda - k$. Because b_k is not a predecessor of a_{k-1}, a_{k-1} has exactly $2(\lambda - k) + 1$

predecessors (itself not counted). Hence scheduling $F(Pred(a_{k-1}))$ requires at least $\lceil 2(\lambda - k) + 1 \rceil + 1$ units of time. Scheduling $a_{k-2}, \cdots, a_1$ requires an additional $k - 2$ time units because these tasks form a path in F. Hence

$$\lambda_{\text{opt}} \geq \lambda - k + 2 + k - 2 = \lambda$$

Thus S is optimal in this case.
Case 2. a_{k-1} is the eldest brother of a_k. Then a_{k-2} exists (by the claim) and is the father of a_k. The only predecessors of a_{k-2} could be: all tasks scheduled before $t - k$, and a_k, a_{k-1}, and a_{k-2}. b_k cannot be a predecessor of a_{k-2}, because it would then occur in L before the eldest son a_{k-1}. Hence $Pred(a_{k-2})$ has a size of $2(\lambda - k) + 3$. In an optimal schedule of $F(Pred(a_{k-2}))$ the task a_{k-2} is scheduled in the last time unit, and in one time unit earlier only one direct predecessor of a_{k-2} can be scheduled because of communication delays. Knowing that $(a_{k-2}, \cdots, a_1)$ is a path in F' we get that

$$\lambda_{\text{opt}} \geq \lceil \frac{2(\lambda - k) + 1}{2} \rceil + 1 + k - 2 = \lambda - k + 2 + k - 2 = \lambda.$$

Thus S is optimal in this case. Q.E.D.

In case of $m = 3$ machines the combination of *CreateList* and *Tassign* does not necessarily yield an optimal schedule. Consider the graph of figure 4. $ept(b) = ept(g) = 3$. If b is chosen to have $scheddist(b) = 2$, then the created list becomes

$$m\ f\ d\ l\ k\ i\ j\ e\ h\ c\ g\ b\ a$$

and *Tassign* will find a schedule with completion time 7.
If on the other hand g is chosen to have $scheddist(g) = 2$, then the list will be

$$f\ d\ m\ e\ l\ h\ i\ j\ c\ k\ b\ g\ a$$

and *Tassign* will find a schedule with completion time 6.

References

[1] F. D. Anger, J.-J. Hwang, and Y.-C. Chow. Scheduling with sufficient loosely coupled processors. *Jrnl. Parallel and Distributed Computing*, 9:87–92, 1990.

[2] P. Chrétienne. A polynomial algorithm to optimally schedule tasks on a virtual distributed system under tree-like precedence constraints. *Europ. Jrnl. Operat. Res.*, 43:225–230, 1989.

[3] P. Chrétienne and C. Picouleau. The basic scheduling problem with interprocessor communication delays. In *Proc. Summerschool on Scheduling Theory and its Applications, Bonas, France*, pages 81–100. INRIA, 1992.

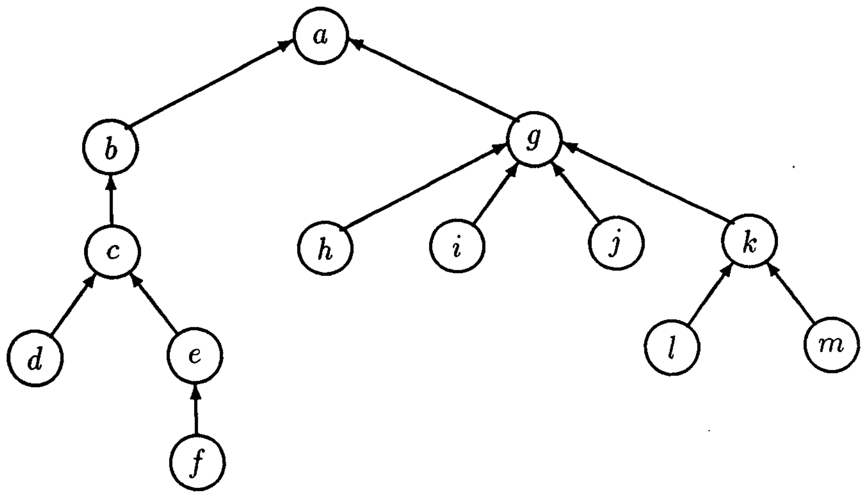


Figure 4: A counter example for $m = 3$ machines.

[4] E. G. Coffman, Jr. and R. L. Graham. Optimal scheduling for two-processor systems. *Acta Inf.*, 1:200–213, 1972.

[5] M. R. Garey, D. S. Johnson, R. E. Tarjan, and M. Yannakakis. Scheduling opposing forests. *SIAM J. Alg. Disc. Meth.*, 4:72–93, 1983.

[6] T. C. Hu. Parallel sequencing and assembly line problems. *Operat. Res.*, 9:841–848, 1961.

[7] C. H. Papadimitriou and M. Yannakakis. Scheduling interval-ordered tasks. *SIAM J. Comput.*, 8:405–409, 1979.

[8] C. Picouleau. Ordonnancement de tâches de durée unitaire avec des délais de communication unitaires sur m processeurs. Technical Report RP 91/64, Laboratoire Méthol. Architecture Systèmes Informatiques, Univers. Pierre et Marie Curie, Paris, France, Dec. 1991.

[9] C. Picouleau. *Etude de problèmes systèmes distribués.* PhD thesis, Univers. Pierre et Marie Curie, Paris, France, 1992.

[10] J. D. Ullmann. NP-complete scheduling problems. *J. Comput. Syst. Sci.*, 10:384–393, 1975.

[11] T. A. Varvarigou, V. P. Roychowdhury, and T. Kailath, 1992. Private communication.

[12] B. Veltman, J. A. Hoogeveen, and J. K. Lenstra, 1992. Personal communication.

[13] B. Veltman, B. J. Lageweg, and J. K. Lenstra. Multiprocessor scheduling with communication delays. *Parallel Computing*, 16:173–182, 1990.