RAIRO. OPERATIONS RESEARCH

FRÉDÉRIC GUINAND
DENIS TRYSTMAN

Scheduling UET trees with communication delays on two processors

RAIRO. Operations Research, tome 34, n° 2 (2000), p. 131-144
<http://www.numdam.org/item?id=RO_2000__34_2_131_0>

RAIRO Operations Research
RAIRO Oper. Res. **34** (2000) 131-144

SCHEDULING UET TREES WITH COMMUNICATION DELAYS ON TWO PROCESSORS (*)

by Frédéric GUINAND ([1]) and Denis TRYSTMAN ([2])

Communicated by Philippe CHRÉTIENNE

Abstract. – *In this paper, we present a new linear time algorithm for scheduling* UECT *(Unit Execution and Communication Time) trees on two identical processors. The chosen criterion is the makespan. The used strategy is based on clustering of tasks. We show that this algorithm builds optimal schedules. Some extensions are discussed for non* UECT *tasks.*

Keywords: Parallel processing, scheduling, intrees, UECT, communication Delays.

1. INTRODUCTION

A parallel application is commonly modeled by a precedence task graph $G = (T, C)$. The computational parts of the application are represented by a set of tasks (T, the vertices of G) exchanging data. These data transfers entail precedence relations between involved tasks, and correspond to communications (C, the arcs of G). G is the graphical representation of $\prec$ the partial order over T. In this context, $T_i \prec T_j$ means T_i precedes T_j. This is generally the case when data computed by T_i are required by T_j.

We consider a restriction of the Papadimitriou and Yannakakis model [PY90], without duplication and with a fixed number of processors. This mainly corresponds to the model of parallel distributed-memory machines used by Rayward-Smith [RS87]. Within this model, a processor is expected to compute no more than one task at a time, and each task requires only one processor in order to be executed. Both execution and communication times are supposed to be unitary. This is called the UECT *assumption*. About the communications:

(*) Received December 1993.
([1]) LIH Université du Havre, BP. 540, 76058 Le Havre Cedex, France.
([2]) LMC-IMAG Domaine Universitaire, BP. 53, 38041 Grenoble Cedex, France.

Recherche opérationnelle/Operations Research, 0399-0559/00/02/$ 7.00

- The communication delays between tasks executed on the same processor are negligible and therefore not taken into account. This assumption is also known as the *locality assumption.*
- The ability of a processor to perform both computations and communications simultaneously is called the *overlap* assumption. For the remaining part of the paper we suppose that the *overlap of the communications by computations* is possible.

In this work we are mainly interested in the scheduling of UECT trees under the previously defined set of hypotheses. The goal of this work is to minimize the date of the end of the execution, the *makespan* denoted $C_{\max}$. While this problem was proved NP-Complete when the number of processors is not fixed [LVV93], much works have been done with a fixed number of processors [GT93, Law93, Pic93, Vel93]. When this number is fixed (m), Varvarigou et al. provide a pseudo-polynomial time algorithm ($O(n^{2(m-1)})$) which produces optimal schedules for the problem. They also propose a linear-time algorithm which produces optimal schedules on a two-processor system and near-optimal schedules on a fixed number of processors [VRKL96].

In the following sections, previous algorithms are analysed, and a new one solving the same problem is described. This new algorithm builds optimal schedules for two-processor machines. Finally, we discuss some extensions for non-UET intrees.

2. PREVIOUS WORK

Every algorithm that solves the problem aims at avoiding idle time due to communication delays, exploiting the overlap assumption. They differ mainly in their strategies. Some are based on a list strategy [Law93, Vel93], while the others are based on a clustering strategy [GT93, Pic93].

2.1. List-scheduling algorithms

2.1.1. *Lawler's algorithm*

Lawler's algorithm is based on the remark that, due to the presence of communication delays, any task has at most one successor executed in the following time slot. In this way, a task with several children has a *favored child* when just one of them is assigned a date earlier than the dates assigned to the other children. For an intree, this idea remains unchanged except that the children are the parents, and the successors are the predecessors. The

principle is simple. Every task is associated to an integer value computed as follows:

- $h(T) = 1$ if T is a leaf,
- $h(T) = \max\{1 + \max(h(j)), 2 + \max_2(h(j))\}$ if T is not a leaf

where (T, j) is an arc, and where $\max_2$ denotes the second largest value of $h(j)$ among all parents of T. $1 + \max(h(j))$ corresponds to the date of the completion of the last executed parent of T. Due to the locality assumption, this date could be the starting date of T if and only if no other parent of T ends simultaneously with j. If another parent ends simultaneously with j, $\max(h(j)) = \max_2(h(j))$, and one unit of time is added because of the impossible overlapping. The favored parent of a task T corresponds to the task with the largest integer. If several tasks have the same integer, the favored parent will be arbitrarily chosen. The labeled tree of the next Figure 1 illustrates the favored parent of each task.

During the next step, the algorithm builds another tree called the *shortest delay-free tree*. On one hand, every task marked *favored* in the original tree has its favored parent and its brothers as predecessors in the *shortest delay-free* tree. On the other hand, every task not marked has only its favored parent as its predecessor.

The last step consists in applying any level-by-level algorithm (for instance Hu's well-known algorithm [Hu61]) with respect to the precedence constraints.

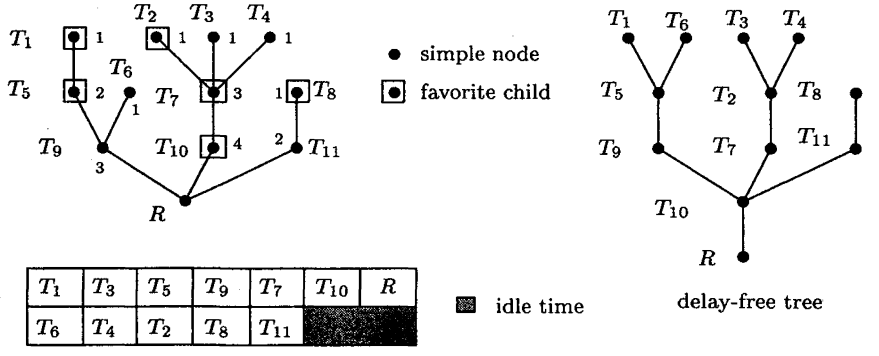


Figure 1. – An optimal schedule obtained by Lawler.

For a fixed number of machines m, the produced schedules do not exceed an optimal solution by more than $m/2$ time units [GRT97]. The optimality could be obtained for a good choice of the favorite child. An optimal choice

is given by the algorithm based on dynamic programming and described in [VRKL96].

2.1.2. *Veldhorst's algorithm*

Within Veldhorst's algorithm a list of the tasks is built according to their distance to the root [Vel93]. The definition of the distance is slightly different from the common one which considers only the number of arcs between the task and the root. The algorithm assigns a *scheddist-label* (the distance) to each task. For each task T, the scheddist-label is computed according to the *earliest processing time* of T ($ept(T)$):

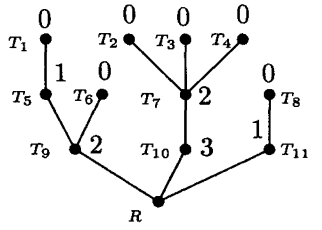
- $ept(T) = 0$ if T is a leaf,
- $ept(T) = 1 + ept(T')$ if any predecessor w of T verifies $ept(w) < ept(T')$,
- $ept(T) = 2 + ept(T')$ if T has at least two predecessors, w_1 and w_2, such that any other predecessor w verifies $ept(w) \leq ept(w_1) = ept(w_2)$.

Figure 2a represents an example of a tree labeled with *ept-values*. Next, the scheddist-labels are computed from the *ept-values*:

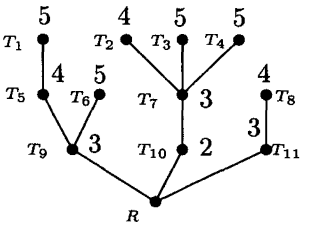
- $scheddist(T) = 0$ if T is the root;
- $scheddist(T) = 1 + scheddist(T')$ if T is the predecessor of T' with the largest *ept-value* (among the predecessors of T'). If two tasks have the maximum ept value, only one can be assigned $1 + scheddist(T')$ as the scheddist-label;
- $scheddist(T) = 2 + scheddist(T')$ for the remaining predecessors of T'.

Figure 2b part of the same figure represents an example of a tree labeled with the *scheddist-labels*.

An ordered list beginning with the root is created. The predecessors of a task in this list are its predecessor with the minimum scheddist-value and its brothers. They are positioned in the front of the list (see Fig. 2c). The tasks are finally scheduled according to their availability in the list. But, if W, a direct predecessor of T, is scheduled exactly one unit of time before T, then both tasks are assigned the same processor. A complete scheduling is represented in Figure 2d.



a) earliest processing time labelling



b) scheddist-labelling

$L = \{T_4, T_3, T_1, T_6, T_2, T_8, T_5, T_7, T_{11}, T_9, T_{10}, R\}$

c) List creation

T_4	T_1	T_2	T_7	T_{11}	T_{10}	R
T_3	T_6	T_8	T_5	T_9		

d) scheduling

Figure 2. – The different steps of Veldhorst's algorithm leading to an optimal schedule.

This algorithm and the previous one are both based on a level-by-level strategy, and they have some common steps. For instance, the labeling in each case leads to the equivalence between the notion of *favorite child* and the predecessor with the lowest *scheddist-label.*

2.2. Clustering algorithms

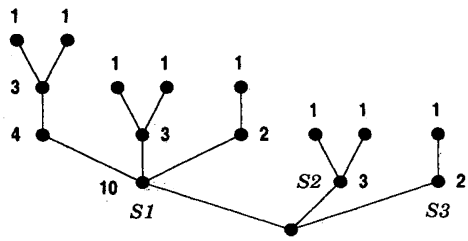
2.2.1. *Principle*

These strategies explicitly gather tasks into clusters. A cluster is simply defined as a set of tasks. A task belongs to one and only one cluster, and a cluster contains at least one task. All the tasks belonging to the same cluster are executed on the same processor. These strategies are well-suited to scheduling trees. Indeed, subtrees constitute evident clusters. The main problem resides in their choice.

2.2.2. *Picouleau's algorithm*

This algorithm builds clusters according to the number of nodes of each subtree belonging to the same level. Subtrees are said to be of the same level if they are rooted in nodes of the same level. For an intree having n different subtrees, the author numbers the subtrees from S_1 to S_n. The number of nodes of S_j is greater than or equal to the number of nodes of S_k (denoted respectively by $N(S_j)$ and $N(S_k)$) if and only if $j < k$ (see Fig. 3). Consider now the following inequality:

$$N(S_1) \leq \left\lfloor \frac{1}{2} \sum_{i=1}^{n} N(S_i) \right\rfloor. \tag{1}$$



$$N(S_1) > \left\lfloor \frac{1}{2} \sum_{i=2}^{i=3} N(S_i) \right\rfloor$$

$\Longrightarrow$ recursive search of optimal scheduling for subtree S_1.

Figure 3. – Labeling step of Picouleau's algorithm.

If (1) is verified, the algorithm performs the following steps:

- determination of k such that S_k is the first subtree for which:

$$\sum_{i=1}^{k} N(S_i) > \sum_{i=k+1}^{n} N(S_i)$$

- allocation of the $k-1$ first subtrees and part of the k^{th} subtree to one processor. The tasks belonging to the k^{th} subtree are scheduled before the tasks belonging to the $k-1$ first allocated subtrees. The remaining tasks of the k^{th} subtree as well as the other non allocated ones are scheduled on the second processor (Fig. 4).

$$N(S_{11}) \leq \left\lfloor \frac{1}{2} \sum_{i=2}^{i=3} N(S_{1i}) \right\rfloor \qquad \sum_{i=1}^{i=2} N(S_{1i}) > \sum_{i=3}^{i=3} N(S_{1i})$$

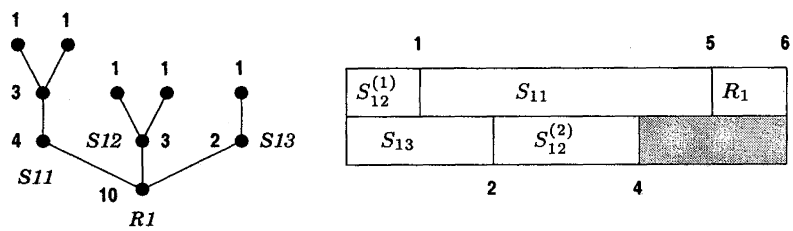


Figure 4. – An optimal schedule for the largest subtree (S_1).

If (1) is not verified (example case), the algorithm is applied recursively:

- determination of an optimal schedule of S_1 on two processors,
- release of idle times in order to add the remaining subtrees.

Idle times are released by replacing all the tasks (suppose n) allocated to the less loaded processor, with the $n-1$ tasks scheduled on the other processor.

These moves are performed until the total number of idle times available on the less loaded processor is at least equal to the number of tasks of the remaining subtrees (Figs. 5 and 6).

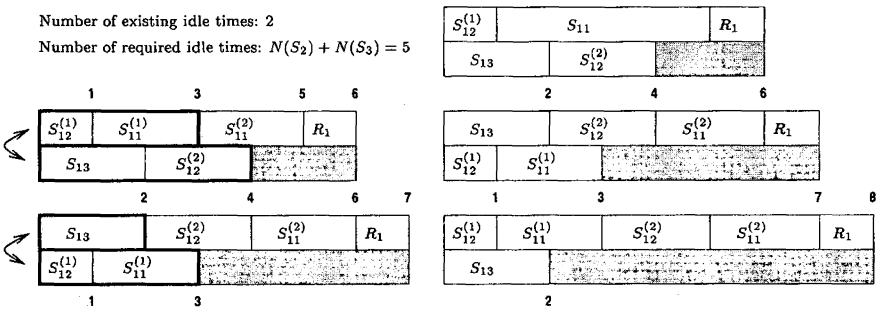


Figure 5. – Idle times creation process.

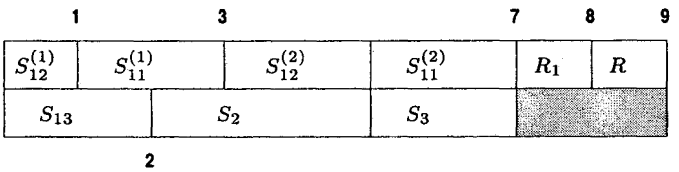


Figure 6. – Scheduling of the remaining part of the tree.

3. A NEW SCHEDULING ALGORITHM

3.1. Notations

The *distance* d_{ij} is the number of arcs between T_i and T_j. If R denotes the root, $l_i = d_{iR} + 1$ is the *level* of T_i. n represents the total number of tasks of the tree, and P_1 and P_2 the processors. In the following, the root is supposed to be executed on P_1.

3.2. Principle of the algorithm

From the execution model hypothesis, a communication between two tasks executed on different processors is much more expensive than the same communication between tasks executed on the same processor. The goal of our clustering strategy is to avoid interprocessor communication. In the most degenerated cases, the best clustering would be to gather all the tasks in one cluster such that all interprocessor communications would be avoided. However, a load-balancing criterion is involved in the value

of the *makespan*. The core of the problem is to find the best trade-off between the minimization of the communications and the minimization of the difference in load between the processors. The algorithm focuses on the allocation of clusters to processor P_2. The clusters are subtrees. The clusters are determined according to a load-balancing criterion. However, unless all tasks are executed on the same processor, some communications must occur. To minimize the impact of these delays on the makespan, the algorithm allocates tasks in such a way that the involved communications can be overlapped by computations.

3.3. Description

In the first step the tasks are labeled with their weight (as in Picouleau's algorithm). Each node is examined once. Then, a level-by-level analysis is performed in order to choose which subtrees (also called clusters) will be allocated to P_2. Finally, subtrees are scheduled on processors.

3.3.1. *Labeling*

The labeling step is straigthforward and needs $O(n)$ operations. The label of T_i corresponds to the number of tasks, including itself, contained within the subtree rooted in T_i. The whole set of tasks of this subtree is denoted $Cluster(T_i)$. The level of a cluster is equal to the level of its root (level of $Cluster(T_i) = l_i$).

3.3.2. *Cluster determination*

The choice of the clusters allocated to P_2 constitutes the second stage of the algorithm. Once a cluster is allocated, all its tasks are *marked*. In order to perform these choices, we compute the maximum number of tasks that could be allocated to P_2 without creating any idle time on P_1. This number is updated after a cluster allocation, or when only one task is left not marked at the current level. It is represented by R in the algorithm, and for the remainder of the paper.

3.3.3. *The central role of* R

The core of the algorithm CD is the setting up and updating of R. R represents an invariant: the maximum number of tasks that could be allocated to P_2 without creating any idle time on P_1. In the beginning, without any prior knowledge concerning the structure of the tree, we suppose a perfect load balancing of the tasks (near $\frac{n}{2}$). However, from the structure of any tree, while one processor executes the root, the other cannot execute any task. So, the root has to be removed from the set of tasks we want to share between the

```
R ← ⌊½(n − 2)⌋
currentLevel ← 2
WHILE (R > 0) DO
        L = {T | T ∈ currentLevel sorted in decreasing order by weight}
        IF (|L| > 1) THEN
                Let T_i the first task ∈ L such that weight(T_i) ≤ R
                IF (such a T_i exists) THEN
                        Cluster (T_i) is allocated to P_2
                        Cluster (T_i) is marked
                        R ← R − weight (T_i)
                        T_i is removed from L
                ELSE /* each T ∈ L verifies weight(T) > R */
                        currentLevel ← currentLevel + 1
                ENDIF
        ELSE
                Let T_alone the only task of L
                R ← min (R, ⌊½(weight(T_alone) − 2)⌋)
                currentLevel ← currentLevel +1
        ENDIF
ENDWHILE
```

Algorithm CD (*Cluster Determination*)

two processors. Moreover, we do not want any idle period to appear on P_1, so, one task has to be set aside for overlapping the communication from the last executed task on P_2 to P_1. These considerations lead to the initialization value of R: $\lfloor \frac{n-2}{2} \rfloor$. This value is then updated as soon as a cluster is allocated to P_2, or when all the tasks but one belonging to the current level are marked. In all cases, the updating of R follows the same mechanism.

3.3.4. *Scheduling of the clusters*

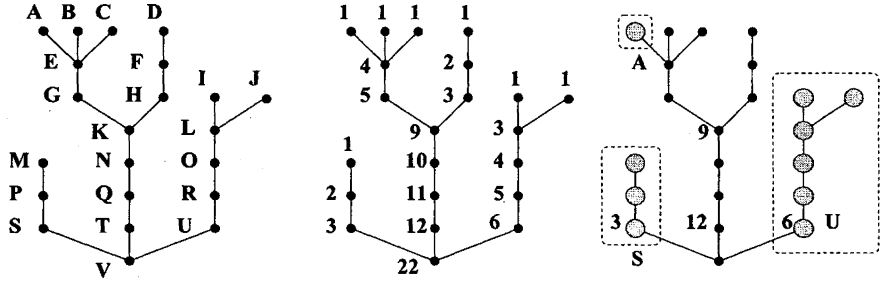
Clusters are allocated to P_2 by algorithm CD. They are scheduled according to the level of their root. The first executed cluster is the one with the highest level. The execution order of clusters with the same level is arbitrary. Thus, a good and simple scheduling of the clusters on P_2 is the reverse order of their allocation. Tasks within a cluster can be scheduled by any level-by-level strategy (Hu's algorithm for example [Hu61]). The same level-based strategy can be used for scheduling the tasks on P_1.

3.4. A complete example

- Number of tasks: 22.
- R $= \lfloor \frac{20}{2} \rfloor = 10$.
- Ordered list at level 2: $L = \{T, U, S\}$.

Level	list L	Chosen cluster	R
2	$\{T, U, S\}$	U	$10 - 6 = 4$
2	$\{T, S\}$	S	$4 - 3 = 1$
2	$\{T\}$	none	$\min\left(1, \left\lfloor \frac{12-2}{2} \right\rfloor\right) = 1$
3	$\{Q\}$	none	$\min\left(1, \left\lfloor \frac{11-2}{2} \right\rfloor\right) = 1$
...		none	$\min(1, ...) = 1$
6	$\{G, H\}$	none	1
7	$\{E, F\}$	none	1
8	$\{A, B, C, D\}$	A	$1 - 1 = 0$

The contents of L (the list) and R are listed in the table for a step-by-step execution of the CD algorithm. At level 6, more than one subtrees remain, and they all have a weight greater than R. In such a situation, the value of R is not updated, and as soon as a level containing subtrees with a weight less than or equal to R is reached, the allocation proceeds (this occurs at level 8).



P_1	B	C	D	E	F	G	H	K	N	Q	T	V
P_2	A	M	P	S	I	J	L	O	R	U		

4. THEORETICAL ANALYSIS

4.1. Some preliminary remarks

Remark 1: Any optimal schedule of a UECT *intree* with idle periods on P_1 [1] can be transformed into another optimal schedule without idle time on it by *moving* the communicating task (T_1 precedes T_3) from P_2 to P_1 (illustrated in Fig. 7).

[1] As said previously this processor executes the root.

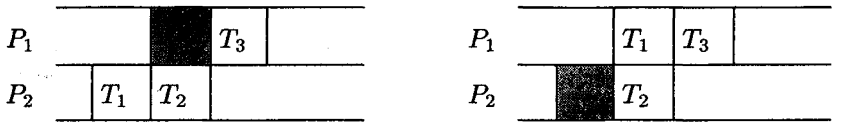


Figure 7. – Idle times can be removed onto P_1.

Remark 2: The algorithm provides schedules with no communication from P_1 to P_2.

Indeed, all the clusters allocated to P_2 are subtrees.

Remark 3: The only optimal schedule for a UECT chain, whatever the number of processors, is obtained by allocating the whole chain to one processor.

Remark 4: Given any optimal schedule of a UECT intree of n tasks on two processors, the less loaded processor (P_2) cannot compute more than $\lfloor \frac{n-1}{2} \rfloor$. Moreover, if P_1 is always busy, the load of P_2 cannot be greater than $\lfloor \frac{n-2}{2} \rfloor$.

4.2. Optimality

From Remarks 1 and 2, the set of schedules corresponds to the shape of Figure 8 is a dominant set.

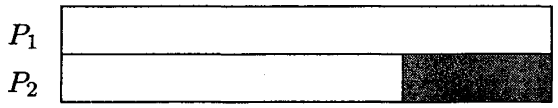


Figure 8. – A dominant set of schedules.

Throughout the algorithm, R is an upper bound for the number of unmarked tasks with level greater than level i that can be scheduled on P_2. From Remark 4, this property is true before the first stage. Three cases can occur at each stage of the CD algorithm:

- $|L| > 1$ and there exists a subtree such that $weight(T_i) \leq R$. In this case, the tasks of T_i are scheduled on P_2 and are marked. Thus the invariant is satisfied.
- $|L| > 1$ and all the subtree of level i are such that $weight(T_i) > R$. The root of T_i must be scheduled on P_1. If not, there could be a communication from P_1 to P_2. Thus, every unmarked task with level i remains unmarked (and then will be scheduled on P_1).
- $|L| = 1$. The unmarked task with level i remains unmarked, so, from remark 4 the new value of R follows.

We prove now that the algorithm ends, *i.e.* R decreases to 0. In the case $|L| > 1$ and $weight(T_i) \leq R$, R decreases strictly. In the two other cases, $currentLevel$ is increased by one, so, after a finite number of steps, R will decrease.

In conclusion, the algorithm yields schedules without idle time on P_1, and allocates the maximum number of tasks to the other processor keeping the previous property, thus providing optimal schedules.

5. EXTENSIONS

In this section, we propose an extension of this algorithm for intrees with non-UET execution times. We denote $P_2 \mid p = 1,\ c = 1,\ intree \mid C_{\max}$ the problem previously studied. By using the same notation the new problem is denoted $P_2 \mid p \in \{1,2\},\ c = 1,\ intree \mid C_{\max}$.

This problem without communication delays has already been studied. Nakajima *et al.* [NLH81] proposed a $O(n\log(n))$ algorithm to optimally schedule such trees. Using a tricky analysis ($O(n^2\log(n))$ in complexity) Du and Leung have proposed a generalization of the previous algorithm, providing optimal schedules for a restricted set $\{1,3\}$ of execution times [DL89]. They also proved the NP-completness of two problems with other restricted sets [DL88]. These results and others are summarized in the following table:

Authors	Complexity	Notation
[NLH81]	$O(n\log(n))$	$P_2\|tree, p_i \in \{1,2\}\|C\max$
[DL89]	$O(n^2\log(n))$	$P_2\|tree, p_i \in \{1,3\}\|C\max$
	OPEN	$P_m\|tree, p_i \in \{1,k\}\|C\max$
[DL88]	NP-complete	$P\|tree, p_i \in \{1,k\}\|C\max$
[DL88]	NP-complete	$P_2\|tree, p_i \in \{k^l, l \geq 0\}\|C\max$
[Chr89]	NP-complete	$P_\infty\|tree, p_i, c_{ij}\|C\max$

Considering the previous problem, our algorithm stopped with the condition $R = 0$. For the current problem, this condition has to be modified to: $R = 0$ **or** $((R = 1)$ **and** (no one-unit task is available in the visited subtree)). The second modification concerns the initialization value and the updating of R which becomes: $\left\lfloor \frac{n-1-\text{weight(root)}}{2} \right\rfloor$.

THEOREM 1: *For* $P_2 \mid p \in \{1,2\},\ c = 1, intree \mid C_{\max}$, *the modified previous algorithm provides schedules such that:* $\omega \leq \omega_{\text{opt}} + 1$ *(where* ω_{opt} *denotes the optimal makespan).*

Proof: When at the current visited level only one task is not marked, or at least two different tasks are not marked (with some subtrees having a weight lower than or equal to R), then, the situation is the same as in the UECT case. A difference can occur when at the current visited level all the subtrees have a weight greater than R. The lightest one is chosen but it is not always possible to find a set of tasks such that their cumulative weight equals R. However, this weight is either equal to R or to R-1. □

Example

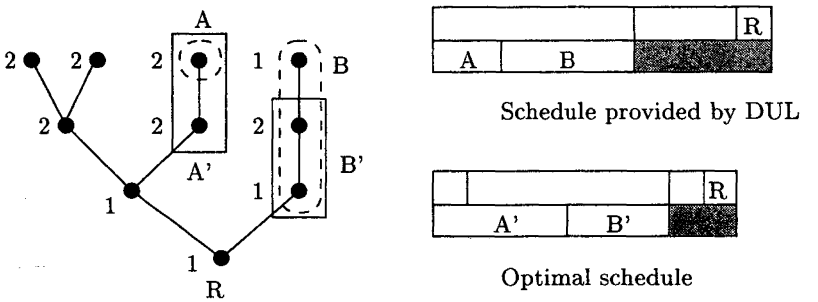


Figure 9. – An example of a non-UET intree.

Remark: For the problem $P_2 \mid p \in \{1, 2, ..., k\},\ c = 1, intree \mid C_{\max}$ with k integer, the same algorithm (with the same modification on the ending condition) yields schedules such that $\omega \leq \omega_{\text{opt}} + (k-1)$. The worst case is obtained for an intree consisting in three tasks, the root (whatever its weight), with its two predecessors of value k. The computation of the load gives the value $k-1$ to R.

6. CONCLUSION

We have presented in this paper a linear-time algorithm that yields optimal schedules for intrees with unit execution time tasks subject to unit communication delays on a two-processor system. Lawler, Veldhorst and Picouleau have also presented optimal linear-time algorithms for the same problem under the same set of hypotheses. In the two first algorithms, the goal of the choice of a *favorite child* and the goal of the *scheddist-label* are the same, *i.e.* finding a task to overlap each communication. In our work, we have the same consideration, but without looking explicitly at these tasks. The process consists in setting aside one task for the overlapping purpose. The second difference is that while an extension of our algorithm to m processors with $m > 2$ is not straightforward, it is simple for the three other

algorithms. However, is can be easily extended to the case of non UET tasks, and provides good schedules.

REFERENCES

[Chr89] P. CHRÉTIENNE, A Polynomial Algorithm to Optimally Schedule Tasks on a Virtual Distributed System Under Tree-like Precedence Constraints. *European J. Oper. Res.* **43** (1989) 225-230.

[DL88] J. DU and J.Y.-T. LEUNG, Scheduling tree-structured tasks with restricted execution times. *Inform. Process. Lett.* **28** (1988) 183-188.

[DL89] J. DU and J.Y.-T. LEUNG, Complexity of Scheduling Parallel Tasks Systems. *SIAM J. Discrete Math.* **2** (1989) 473-487.

[GRT97] F. GUINAND, C. RAPINE and D. TRYSTRAM, Worst-Case Analysis of Algorithms for Scheduling UECT Trees on m Processors. *IEEE Trans. Parallel and Distributed Systems* **8** (1997).

[GT93] F. GUINAND and D. TRYSTRAM, Optimal Scheduling of UECT Trees on Two Processors. Technical Report Rapport APACHE n° 3. Laboratoire de Modélisation et de Calcul - IMAG, Grenoble (1993).

[Hu61] T.C. HU, Parallel Sequencing and Assembly Line Problems. *Oper. Res.* **9** (1961) 841-848.

[Law93] E.L. LAWLER, Scheduling Trees on Multiprocessors with Unit Communication Delays, *Workshop on Models and Algorithms for Planning and Scheduling Problems.* Villa Vigoni, Lake Como, Italy (1993).

[LVV93] J.K. LENSTRA, M. VELDHORST and B. VELTMAN, The Complexity of Scheduling Trees with Communication Delays. *Lecture Notes in Comput. Sci. (ESA'93)* **726** (1993) 284-294.

[NLH81] K. NAKAJIMA, J.Y.-T. LEUNG and S.L. HAKIMI, Optimal Processor Scheduling of Tree Precedence Constrainted Tasks with Two Execution Times. *Performance Evaluation* **1** (1981) 320-330.

[Pic93] C. PICOULEAU, Étude des Problèmes d'Optimisation dans les Systèmes Distribués. Ph.D. Thesis, Paris VI, Paris - France (1993), in French.

[PY90] C.H. PAPADIMITRIOU and M. YANNAKAKIS, Towards an architecture-independent analysis of parallel algorithms. *SIAM J. Comput.* **19** (1990) 322-328.

[RS87] V.J. RAYWARD-SMITH, UET Scheduling with Unit Interprocessor Communication Delays. *Discrete Appl. Math.* **18** (1987) 55-71.

[Vel93] M. VELDHORST, A Linear Time Algorithm to Schedule Trees with Communication Delays Optimally on Two Machines, Technical Report RUU-CS-93-04. Department of computer science, Utrecht (1993).

[VRKL96] T.A. VARVARIGOU, V.P. ROYCHOWDHURY, T. KAILATH and E. LAWLER, Scheduling In and Out Forests in the Presence of Communication Delays. *IEE Trans. Parallel and Distributed Systems* **7** (1996).