Journal of Intelligent Manufacturing, 16, 407–421, 2005

Computational methods for determining the latest starting times and floats of tasks in interval-valued activity networks

DIDIER DUBOIS, HÉLÈNE FARGIER and JÉRÔME FORTIN

Irit/UPS, 118 route de narbonne, F-31062 Toulouse, France

Received Feburary 2004 and accepted November 2004

In project management, three quantities are often used by project managers: the earliest starting date, the latest starting date and the float of tasks. These quantities are computed by the Program Evaluation and Review Techniques/Critical Path Method (PERT/CPM) algorithm. When task durations are ill known, as is often the case at the beginning of a project, they can be modeled by means of intervals, representing the possible values of these task durations. With such a representation, the earliest starting dates, the latest starting dates and the floats are also intervals. The purpose of this paper is to give efficient algorithms for their computation. After recalling the classical PERT/CPM problem, we present several properties of the concerned quantities in the interval-valued case, showing that the standard criticality analysis collapses. We propose an efficient algorithm based on path enumeration to compute optimal intervals for latest starting times and floats in the general case, and a simpler polynomial algorithm in the case of series-parallel activity networks.

Keywords: PERT/CPM, earliest starting date, latest starting date, float, series-parallel graph

1. Introduction

In project or production management, an activity network is classically defined by a set of tasks (or activities) and a set of precedence constraints expressing which tasks cannot start before others are completed. When there are no resource constraints, we can display the network as a directed acyclic graph, where nodes represent tasks, and arcs precedence constraints. With such a network, the goal of a project manager is generally to minimize the makespan of the project. Three quantities are frequently calculated for each task of the project. They allow to identify the critical tasks: The *earliest starting date* est_i of a task i is the date before, which we cannot start a task without violation of a precedence constraint. The *latest starting date* lst_i is the date after, which we cannot start the task without delaying the end of the project. The *float* f_i is the difference between the latest starting date and the earliest starting date. A task is then critical if and only if its float is null.

Critical Path Method (CPM) and Program Evaluation and Review Techniques (PERT) are methods developed in order to calculate the earliest starting date, the latest starting date and the float of the tasks of a project, under the hypothesis that the tasks durations are well known (Bellman *et al.*, 1982). Those problems are traditionally called PERT/CPM problems.

In fact the tasks durations are often ill known at the time the project is designed. Scheduling problems in which durations are modeled by stochastic distributions have been studied since the 1950s (Loostma, 1989; Brige and Dempster, 1996). But the computational problems encountered with this approach are still challenging and are no yet fully mastered.

Another way to model uncertainty of the durations, based on a possibility-theoretic approach has been considered since the early 1980's, where the tasks durations are represented by (crisp or fuzzy) intervals (Prade, 1979; Dubois, 1983; Buckley, 1989; Loostma, 1997). Several authors have recently shown that calculating the float or discussing the criticality of a task in this context (something obvious in the classical problem) become a less trivial issue (Chanas *et al.*, 2002; Dubois *et al.*, 2003a,b. For instance, finding optimal intervals containing the value of the floats is a NP-hard problem (Chanas and Zielinski, 2003). Polynomial algorithms have been recently found for the calculation of the latest starting dates (Zielinski, 2005), but they only find approximate bounds of the floats.

The basic idea for computing optimal intervals for starting times and floats is to look for assignments of precise durations to activities, for which the bounds of these intervals are attained. Such complete assignments are called configurations. The main goal of this paper is to present the form of the optimal configurations, and new algorithms that calculate exact intervals for latest starting dates and floats in a network, where tasks durations are modeled by crisp intervals. Of course, these algorithms are not polynomial in the general case, but they have been tested on realistic scheduling problems for the experimental validation of their efficiency. We will also propose linear algorithms which compute the latest starting date and the float of a task in the particular case of series-parallel graphs.

2. The interval-valued scheduling problem

We define an interval-valued scheduling problem by a network $R = \langle \tau, C, T \rangle$, where τ is the set of tasks, C is the set of precedence constraints, and T a function that, to each task $i \in \tau$, assigns a set of possible durations D_i, under the form of a closed interval. $D_i = [d_i^-, d_i^+]$ means that the real duration d_i of the task i is not precisely known, but lies between d_i^- and d_i^+ ($d_i \in D_i$). In order to relate the interval case to the deterministic case of classical PERT/CPM problems, Buckley has defined the notion of configuration as follows (Buckley, 1989):

Definition 1. A configuration is a tuple $\Omega = (d_1, d_2, ..., d_n)$ *of durations such that* $\forall i \in \tau, d_i \in D_i$.

H denotes the set of all configurations: $H = \times_{i \in [1,n]} D_i$. The duration of task i in configuration Ω is $d_i(\Omega)$. A configuration defines an instance of deterministic scheduling problem (classical PERT/CPM problem), to which the PERT/CPM method can be applied. Using configurations, the possible values EST_i for the earliest starting date est_i, the possible values LST_i for the latest starting date lst_i and the possible values F_i for the float f_i are defined as follows (Dubois *et al.*, 2003b).

$$est_i \in EST_i = \{est_i(\Omega) | \Omega \in H\} \qquad (1)$$

$$lst_i \in LST_i = \{lst_i(\Omega) | \Omega \in H\} \qquad (2)$$

$$\begin{aligned} f_i \in F_i &= \{f_i(\Omega) | \Omega \in H\} \\ &= \{lst_i(\Omega) - est_i(\Omega) | \Omega \in H\} \end{aligned} \qquad (3)$$

Functions that define the earliest starting times, latest starting times and floats in terms of task durations are obviously continuous, hence the quantities EST_i, LST_i and F_i are closed intervals ($EST_i = [est_i^-, est_i^+]$, $LST_i = [lst_i^-, lst_i^+]$ and $F_i = [f_i^-, f_i^+]$). (Chanas *et al.*, 2002) propose to define criticality in interval-valued problems as follows: a task i is *possibly critical* if there exists a configuration $\Omega \in H$ in which i is critical in the usual sense. A task i is *necessarily critical* if i is critical in the usual sense in all configurations $\Omega \in H$. We can state the following propositions:

Proposition 1. A task i is possibly critical if and only if the lower bound of its float is null: $f_i^- = inf(F_i) = 0$.

Proof. By definition, $F_i = \{f_i(\Omega), \Omega \in H\}$. So $inf(F_i) = 0$ if and only if there exists a configuration $\Omega \in H$ such that $f_i(\Omega) = 0$, and that is the definition of a possibly critical task. □

Proposition 2. A task i is necessarily critical if and only if the upper bound of its float is null: $f_i^+ = sup(F_i) = 0$.

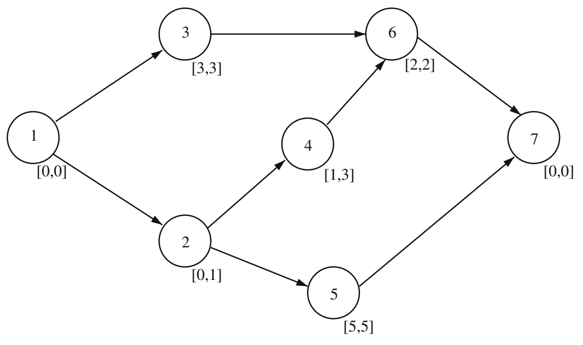

Fig. 1. Network which invalidates the converse of Proposition 3.

Proof. Same sketch of proof as Proposition 1. □

Proposition 3. If a task i is necessarily critical then $EST_i = LST_i$.

Proof. Let i be a necessarily critical task and suppose that $EST_i \neq LST_i$, then there exists a configuration $\Omega' \in \tau$ such that $est_i(\Omega') \neq lst_i(\Omega')$ (otherwise for all $\Omega \in H$, $est_i(\Omega') = lst_i(\Omega')$ and then $EST_i = LST_i$). Thus $f_i(\Omega') = lst_i(\Omega') - est_i(\Omega')$ and then $f_i(\Omega') > 0$ (without resource constraints, $lst_i(\Omega') \geq est_i(\Omega')$), which implies $F_i \neq [0, 0]$. □

Remark. As shown in the graph in Fig. 1, the converse of Proposition 3 is false: task 6 has for earliest and latest starting date intervals $EST_6 = LST_6 = [3, 4]$, but its float is $F_6 = [0, 1]$.

In fact, when $EST_i = LST_i$, the task is only possibly critical. The last proposition makes it clear that the interval containing the float of a task can not be calculated by means of the intervals containing the earliest and latest starting dates of this task. The float, from which the criticality of the task can be assessed, must be computed separately.

3. Critical path analysis and algorithms for the interval case

Critical path analysis in an interval-valued activity network is actually quite different from deterministic critical path analysis (Chanas *et al.*, 2002). Namely, there are many possibly critical paths. But in the case where all task durations are modeled by intervals (none of which is reduced to a singleton), there is at most one necessarily critical path, if any. Then the necessarily critical tasks are along this path. But most of the time, there is no necessarily critical path, while there may exist isolated critical tasks (obviously the first task and the last task are necessarily critical, even when no path is such). Finding these isolated critical tasks is important at the practical level and difficult at the computational level. One obvious way to do it is to check for tasks whose floats reduce to the singleton $\{0\}$, as shown above. We are going to show several graph-topological properties of interval-valued PERT/CPM problems, which lead to a new algorithm to calculate latest starting dates and floats. It is important to note that computing earliest and latest starting dates can be done in polynomial time (Zielinski, 2005), contrary to floats. It has been shown that asserting if a task is possibly critical is a NP-complete problem (Chanas and Zielinski, 2003), so the calculation of its float is consequently NP-hard.

3.1. *Notations*

Let $R = <\tau, C, T>$ be a network, C represents the precedence constraints: if i must be completed before j starts then $(i, j) \in C$. The set of tasks $\tau = \{1, 2, \ldots, n\}$ is labeled in such a way that $i < j$ if i has to precede j ($(i, j) \in C$). Without losing any generality, the network is supposed to have a single initial task, and a single final task; if a project has more than one task without predecessor (respectively, successor), we can create a fictitious task with a null duration, preceding (respectively, following) all the tasks. The initial task of R is the task 1 and ending task is the task n. $P_{i,j}$ denotes the set of paths from the task i to the task j in R. When not ambiguous, we shall denote by P the set of all paths from 1 to n ($P = P_{1,n}$).

Let p be a path, $W_p(\Omega)$ is the length of p in Ω:

$$W_p(\Omega) = \sum_{i \in p} d_i(\Omega) \tag{4}$$

If i is a task, $pred(i)$ (respectively, $succ(i)$) is the set of immediate predecessors (respectively, successors) of i: $pred(i) = \{j \mid (j, i) \in C\}$, and $succ(i) = \{j \mid (i, j) \in C\}$.

$PRED(i)$ (respectively, $SUCC(i)$) is the set of all predecessors of i (respectively, successors), obtained by transitivity.

3.2. *Computing earliest starting dates*

Classical PERT/CPM calculation is based on the propagation from the initial task to the ending task of the earliest starting date by a well-known recursive formula:

$$est_i = max_{j \in pred(i)}(est_j + d_j) \tag{5}$$

Traditionally, we set the initial task earliest starting date to zero ($est_1 = 0$), and minimizing the makespan comes down to letting the latest starting date of n equal its earliest starting date. The classical PERT/CPM algorithm which calculates the earliest starting date, the latest starting date, and the float of each task of a network has a linear time complexity in $O(n+m)$ where n is the number of tasks of the network and m is the number of precedence constraints.

In our problem, where task durations are described by intervals, the earliest starting date can still be computed by the following formula (Dubois and Prade, 1980):

$$EST_i = \widetilde{max}_{j \in pred(i)}(EST_j \oplus D_j) \tag{6}$$

In this formula, $\widetilde{max}$ (respectively, $\oplus$) is the operator maximum (respectively addition) defined on intervals: $\widetilde{max}([a,b],[c,d]) = [max(a,c), max(b,d)]$ and $[a,b] \oplus [c,d] = [a+c, b+d]$. According to formula (6), the calculation of this interval can be split in two parts: one for the greatest lower bound (GLB), and the other for the lowest upper bound (LUB) of the earliest starting date interval:

Proposition 4. Let $\underline{\Omega}$ and $\overline{\Omega}$ be the two configurations such that for all tasks $i \in \tau$: $d_i(\underline{\Omega}) = d_i^-$ and $d_i(\overline{\Omega}) = d_i^+$. Then, configuration $\underline{\Omega}$ minimizes the earliest starting date of all the tasks $i \in \tau$, and configuration $\overline{\Omega}$ maximizes their earliest starting dates.

Proof. See Chanas and Kamburowski (1981). □

In general, $\underline{\Omega}$ is called the optimistic configuration, and $\overline{\Omega}$ is called the pessimistic configuration. This proposition yields an algorithm which calculates the earliest starting date in a linear time complexity ($O(n+m)$), since it requires two classical PERT/CPM computations, on two extreme configurations.

3.3. *A brute-force algorithm for calculating latest starting dates and floats*

In the classical PERT/CPM problem, the latest starting dates and the floats are calculated as follows:

$$lst_i = min_{j \in succ(i)}(lst_j - d_i) \tag{7}$$

$$f_i = lst_i - est_i \tag{8}$$

In the case of interval data, one may be tempted to calculate latest starting dates and floats by means of Equations 7 and 8 extended to intervals. Unfortunately we can only show the following bracketing results:

$$LST_i \subseteq \widetilde{min}_{j \in succ(i)}(LST_j \ominus D_j) \tag{9}$$

$$F_i \subseteq LST_i \ominus EST_i \tag{10}$$

In these last formulas, $\widetilde{min}$ (respectively, $\ominus$) is the operator minimum (respectively subtraction) defined on intervals: $\widetilde{min}([a,b],[c,d]) = [min(a,c), min(b,d)]$ and $[a,b] \ominus [c,d] = [a-d, b-c]$. Proposition 4 already pointed out that it is not possible to compute F_i from EST_i and LST_i. Equation 9 (as Equation 10) would be not valid if we replaced $\subseteq$ by $=$ because of the dependency of the variables LST_i and D_j (Dubois, 1983; Dubois *et al.*, 2003b). The trivial problem of Fig. 2 easily illustrates this dependency. Of course for task 2 it holds that $EST_2 = LST_2 = [1,2]$. And if we calculate the latest starting date of task 1 by the formula (9) with the equality sign, we would obtain $LST_1 = LST_2 \ominus D_1 = [-1,1]$.

So, the calculation of the latest starting dates and the floats are then less simple than the calculation of the earliest starting date. A brute-force method has been proposed by Dubois *et al.*, (2003). This method is based on the notion of

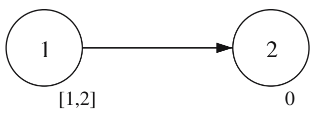

Fig. 2. Illustration of Equation (9).

```
Algorithm 1: Calculation of the latest starting dates and the floats of all the tasks of the network
Input: a network $R = \langle \tau, C, T \rangle$;
Output: The latest starting dates and the floats of all the tasks of the network;
begin
    foreach $i \in \tau$ do
        $lst_i^- = +\infty$;
        $lst_i^+ = 0$;
        $f_i^- = +\infty$;
        $f_i^+ = 0$;
        $d_i(\Omega) = d_i^-$;
    end
    instantiate(1);
end
```

```
Procedure instantiate (j)
begin
    if $j = n$ then
        $update(\Omega)$;
    else
        $instantiate(j+1)$;
        $d_j(\Omega) = d_j^+$;
        $instantiate(j+1)$;
        $d_j(\Omega) = d_j^-$;
end
```

```
Procedure update ($\Omega$)
begin
    Compute $lst_i(\Omega)$ and $f_i(\Omega)$ for all $i \in \tau$ by a classical PERT/CPM calculation;
    if $lst_i(\Omega) < lst_i^-$ then $lst_i^- = lst_i(\Omega)$;
    if $lst_i(\Omega) > lst_i^+$ then $lst_i^+ = lst_i(\Omega)$;
    if $f_i(\Omega) < f_i^-$ then $f_i^- = f_i(\Omega)$;
    if $f_i(\Omega) > f_i^+$ then $f_i^+ = f_i(\Omega)$;
end
```

extreme configurations which are configurations where tasks durations are assigned to their minimal or maximal possible values. Formally extreme configurations are defined as following:

Definition 2. An extreme configuration is $\Omega \in H$ *such that* $\forall i \in \tau, d_i(\Omega) = d_i^+ \ or d_i^-$.

H_{ext} denotes the set of all extreme configurations. There are obviously $|H_{\text{ext}}| = 2^n$ such configurations. In Dubois *et al.*, (2003), it is shown that the maximum and the minimum of $lst_i(.)$ and $f_i(.)$ are attained on specific extreme configurations.

This leads to Algorithm 1 which executes a classical PERT/CPM calculation on each extreme configuration. The instantiation of each configuration is made by the recursive procedure *instantiate*. The procedure *update* executes the classical PERT/CPM algorithm and updates the current values of the latest starting dates and floats if needed. Of course, the complexity of Algorithm 1 is exponential, precisely in $O((n+m)*2^n)$.

In Sections 4 and 5, we are going to prove several results, which significantly decrease the number of configurations that need to be tested.

3.4. *Optimal configurations for the latest starting date intervals*

In the PERT/CPM problem, the latest starting date is calculated by the recursive formula (7). But the basic definition of the latest starting date is :

$$lst_j = max_{p \in P} W_p - max_{p \in P_{j,n}} W_p \qquad (11)$$

In this formulation, two paths are involved: the longest path from 1 to n, $max_{p\in P} W_p$, which allows to calculate the earliest ending time of the project, and the longest path from j to n, $max_{p\in P_{j,n}} W_p$. With this formulation we easily can deduce the next results:

Proposition 5. The maximum of $lst_i(.)$ is reached on an extreme configuration Ω such that for all $j \notin SUCC(i) \cup \{i\}$, $d_j(\Omega) = d_j^+$.

The minimum of $lst_i(.)$ is attained on an extreme configuration Ω such that for all $j \notin SUCC(i) \cup \{i\}$, $d_j(\Omega) = d_j^-$.

Proof. See Dubois *et al.*, (2003). □

We are now going to show that there exists an extreme configuration which minimizes $lst_i(.)$ where all the task durations are assigned to their minimum d_i^- but the ones on a single path from task i to the ending task n:

Lemma 1. Let $i \in \tau$ be a task of R. There exists a path $p^* \in P_{i,n}$ such that an extreme configuration Ω defined as $d_j(\Omega) = d_j^- \forall j \notin p^*$ minimizes $lst_i(.)$.

Proof. See Appendix A.

Now, from the configuration Ω in the last lemma, we are going to show that all the durations of tasks j in path p^* can be set to their maximal values d_j^+. The obtained configuration will still minimize the latest starting date of i.

Proposition 6. Let $i \in \tau$ be a task of R. There exists a path $p^* \in P_{i,n}$ such that the extreme configuration Ω defined by:

$$d_j(\Omega) = \begin{cases} d_j^+ \ if \ j \in p^* \\ d_j^- \ otherwise \end{cases} \quad minimizes\ lst_i(.).$$

Proof. See Appendix A.

Remark. It is not sufficient to find the longest path $\overline{p}_{i,n}$ from i to n in the pessimistic configuration $\overline{\Omega}$, and then to calculate a classical PERT/CPM on the configuration Ω defined by $d_i(\Omega) = d_i^+$ if $i \in \overline{p}_{i,n}$, and $d_i(\Omega) = d_i^-$ otherwise. This is illustrated by Fig. 3. In the pessimistic configuration, the longest path from 2 to 6 is $\overline{p}_{2,6} = (2, 5, 6)$. In configuration Ω where all the durations are minimal but on $\overline{p}_{2,6} = (2, 5, 6)$, we find $lst_2(\Omega) = 1$ for the latest starting date of

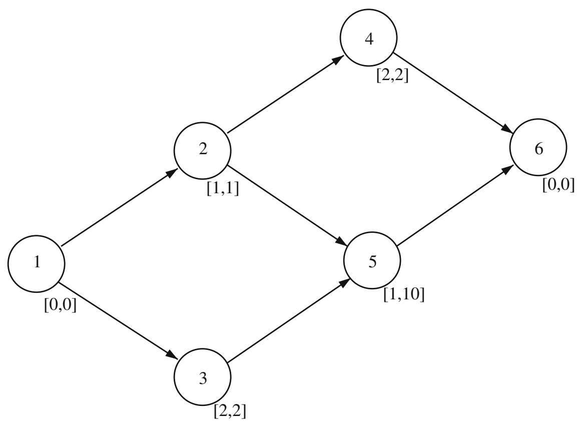

Fig. 3. Counter-example network.

task 2. In fact $lst_2^- = 0$ (with configuration $\Omega = (0, 1, 2, 2, 1, 0)$).

The same kind of propositions, now applied to the calculation of the LUB of the latest starting date, can be proved. The first of the two following propositions, is useful to find the latest starting dates of all the tasks in the network. The second one reduces the number of configurations to be tested, but the set of configurations is only useful for the calculation of the latest starting date of a single given task.

Proposition 7. Let $i \in \tau$ be a task of R. There exists a path $p^* \in P$ such that the extreme configuration Ω defined by $d_j(\Omega) = \begin{cases} d_j^+ \ if \ j \in p^* \\ d_j^- \ otherwise \end{cases}$ maximizes $lst_i(.)$.

Proof. See Appendix A.

Proposition 8. Let $i \in \tau$ be a task of R. There exists a path $p^* \in P$ such that the extreme configuration Ω defined by

$$d_j(\Omega) = \begin{cases} d_j^+ \ if \ j \notin SUCC(i) \cup \{i\} \\ d_j^+ \ if \ j \in p^* \\ d_j^- \ otherwise \end{cases} \quad maximizes\ lst_i(.).$$

Proof. See Appendix A.

Remark. We cannot say that the path p^* in the two last propositions has to be the longest path in the pessimistic configuration $\overline{\Omega}$. Figure 4 is a counter-example. In configuration $\overline{\Omega} = (0, 3, 2, 0)$

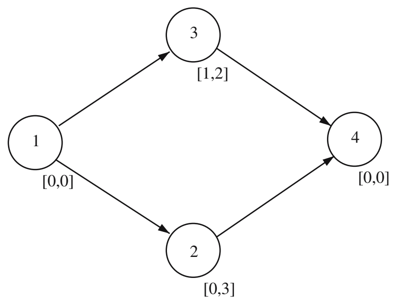

Fig. 4. Counter-example network.

the only critical path is $p^* = (1,2,4)$ but the maximum of the latest starting date of task 2 has to be calculated in configuration $\Omega = (0,0,2,0)$.

3.5. *Optimal configurations for float intervals*

The same kind of approach can be followed for computing the narrowest intervals containing the floats. The basic definition of the float is :

$$lst_j = max_{p\in P} W_p - max_{p\in P_{1,j}} W_p - max_{p\in P_{j,n}} max_{p\in P_{j,n}} W_p + d_j \tag{12}$$

It is the difference between the longest path of the network and the longest path of the network containing j.

The bounds of the floats are again attained on particular extreme configurations, described by the next propositions:

Proposition 9. Let $i \in \tau$ be a task of R. There exists a path $p^* \in P$ containing i, such that the extreme configuration Ω defined by $d_j(\Omega) = \begin{cases} d_j^+ \ if \ j \in p^* \\ d_j^- \ otherwise \end{cases}$ *minimizes $f_i(.)$.*

Proposition 10. Let $i \in \tau$ be a task of R. There exists a path $p^* \in P$, such that the extreme configuration Ω defined by $d_j(\Omega) = \begin{cases} d_j^+ \ if \ j \in p^* \\ d_j^- \ otherwise \end{cases}$ *maximizes $f_i(.)$.*

Proposition 11. Let $i \in \tau$ be a task of R. There exists a path $p^* \in P$, such that the extreme configuration Ω defined by

$$d_j(\Omega) = \begin{cases} d_j^+ \ if \ j \notin PRED(i) \cup \{i\} \cup SUCC(i) \\ d_j^+ \ if \ j \in p^* \\ d_j^- \ otherwise \end{cases}$$

maximizes $f_i(.)$.

These propositions can be proved similary to the ones concerning latest starting times. Their proofs are omitted, for the sake of brevity.

3.6. *An improved algorithm for computing optimal intervals for latest starting times and floats*

In Section 6, we obtained results where some particular extreme configurations were highlighted as being potentially optimal ones. If H_i denotes the set of configurations, where all durations are minimal but on a path from i to n, then the execution of a classical PERT/CPM calculation on each configuration of H_i computes the GLB of the latest starting date (Proposition 6). And the execution of a PERT/CPM on H_1 also yields the LUB of the latest starting date(Proposition 7), the GLB and the LUB of the float of each task of i (Propositions 9 and 10).

The procedure *instantiate* of Algorithm 1 can be changed into the two following procedures in order to improve the algorithm.

```
Procedure $instantiate(j)$
begin
    $instantiate\_path(j)$;
    if $j \neq n$ then $instantiate(j+1)$;
end
```

```
Procedure $instantiate\_path(j)$
begin
    if $j = n$ then
        $update(\Omega)$;
    else
        $d_j(\Omega) = d_j^+$;
        foreach $k \in succ(j)$ do
            $instantiate\_path(k)$;
        $d_j(\Omega) = d_j^-$;
end
```

A call to $instantiate\_path(j)$ from the procedure *instantiate* constructs the set H_j. Algorithm $instantiate\_path(j)$ recursively assigns maximal task durations to a path from the current task j

to n. With this modified procedure, Algorithm 1 *instantiate* is called the *path-algorithm* in the rest of this paper.

Note that for each path p from a task i to n, there exists at least one path from 1 to n in which p is included. We can deduce that for all $i \in \tau$, $|H_1| \geq |H_i|$, where $|H_i|$ is the number of configurations in H_i. We can now estimate the time complexity of the algorithm as $O((n+m)*n*|P|)$. This complexity depends on the topology of the network.

If we only want to compute GLB and LUB of floats, we just have to compute a PERT/CPM on the configurations on H_1 instead of $\Omega \in H_1 \cup H_2 \cup \cdots \cup H_n$. The resulting time complexity is then $O((n+m)*|P|)$.

4. Experimental validation

The complexity of the path-algorithm presented in Section 3.6 depends on the topology of the activity network. That is why it should be tested on realistic scheduling problems. For this reason, we have tested our algorithm on some scheduling problem libraries (PSPLIB: http://129.187.106.231/psplib/).

Those instances of problems have been generated by the ProGen program, for activity network generation (Kolisch *et al.*, 1995; Kolisch and Sprecher, 1996). They are supposed to be representative of real scheduling problems. On those problems, tasks durations are precisely defined. For our test, we have added a relative uncertainty range of 20% to obtain intervals ($D_i = [d_i; d_i + 20\%]$). The choice of those intervals is not important for the test due to the fact that the path-algorithm complexity only depends on the topology of the associated graph. The following table compares the performance of the path-algorithm on libraries of scheduling problems with, respectively, 32, 62, 92 and 122 tasks (on 480, 480, 480 and 600 instances of problems, respectively). Two quantities are measured, the execution time (expressed in milliseconds) and the number of PERT/CPM calculations invoked by the algorithm. Results of the next table are presented in Figs. 5 and 6.

These tests show the path-algorithm can compute the latest starting dates and the floats of

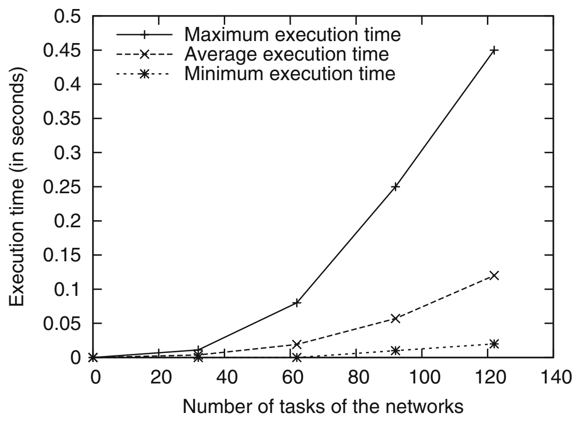

Fig. 5. Execution times of the path-algorithm.

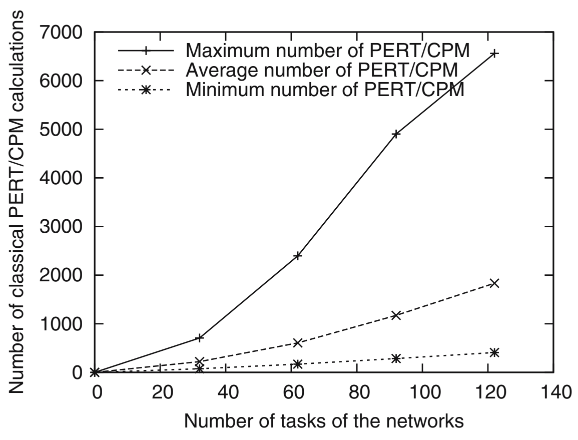

Fig. 6. Number of classical PERT/CPM calculations for the path-algorithm.

activities in big networks with acceptable execution times.

The next table presents the comparaison of average execution times (in millisecond) of the path-algorithm and the recent algorithms by Zielinski (2005), that compute latest starting dates in polynomial time. This comparaison is made on the same library of problem of 32, 62, 92 and 122 tasks.

The path-algorithm computes GLB and LUB of latest starting dates and floats in the same pass. As we see in the last table, the execution times of the path-algorithm are approximately equal to the sum of execution times of algorithms that polynomially compute the GLB and LUB of latest starting dates, which means that the path-algorithm runs very fast on realistic problems. However, polynomial algorithms should eventually be the best for really large problems.

Nb of tasks	32	62	92	122
Nb of networks tested	480	480	480	600
Average execution time	3.7	19	57	120
Minimal execution time	0	0	10	20
Maximal execution time	11	80	250	450
Average nb of PERT/CPM	222	607	1173	1833
Minimal nb of PERT/CPM	76	169	284	408
Maximal nb of PERT/CPM	708	2396	4903	6559

Nb of tasks	32	62	92	122
Path-algorithm	3.7	19	57	120
Polynomial algorithm for the GLB of the latest starting dates	1.1	5	30	23
Polynomial algorithm for the LUB of the latest starting dates	2.1	10	27	56

5. The case of series-parallel graphs

We propose here a study of scheduling problems where the associated graph of the network is series-parallel. A special case of series-parallel graph problems has been studied by Dubois *et al.*, (2003b), with a restricted definition of a series-parallel graph. For those networks, polynomial algorithms were proposed, that calculate the latest starting dates and the floats of the tasks. Here their algorithms are shown to apply to a more general notion of series-parallel graph.

Definition 3. A graph is said to be series-parallel if and only if it is recursively defined in the following way:

- *A graph composed of two nodes related by an arc is series-parallel.*
- *Given G_1 and G_2 two series-parallel graphs, the result of one of the two following operations is a series parallel graph.*
  - ***Parallelization**: Identifying the initial node of G_1 with the initial node of G_2, and identifying the final node of G_1 with the final node of G_2.*
  - ***Serialization**: Identifying the initial node of G_1 with the final node of G_2*

There exist polynomial algorithms that recognize series-parallel graphs (Eppstein, Bein, 1992). We obviously note that a series-parallel graph has a single starting task, and a single ending task. Figure 7 is an example of series-parallel graph.

The algorithm proposed by Dubois *et al.*, (2003b) was based on the following definition of what we will call strongly series-parallel graphs. A graph is strongly series-parallel if:

- It is made of a single node.
- Or it has only one task without a predecessor (the initial task), it has only one task without successor (the final task), and the graph induced by the deletion of the initial and final tasks is made of disconnected series-parallel graphs.

We can note that all the strongly series-parallel graphs composed of at least two nodes are series-parallel according to the general definition. On the contrary, the graph of Fig. 7 is not

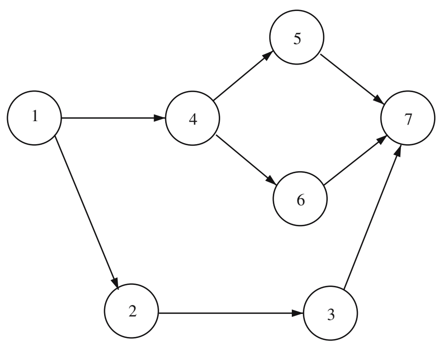

Fig. 7. Example of a series-parallel graph.

strongly series-parallel. We claim that the calculation of latest starting dates and floats proposed for strongly series-parallel graphs is still valid for general series-parallel graphs. The next lemma is useful for proving the propositions which validate the algorithm proposed by Dubois *et al.*, (2003b), but now for general series-parallel graphs:

Lemma 2. *Let G be a series-parallel graph. Let $\|G\|_g$ the sum of the number of nodes and arcs in G ($\|\cdot\|_g$ define a norm on the set of graphs), then:*

- *Either G contains only two nodes linked by an arc, and then $\|G\|_g = 3$.*
- *Or, G can be decomposed in two series-parallel sub-graphs G_1 and G_2 such that G is the result of the parallelization or the serialization of G_1 and G_2, and then $\|G\|_g > \|G_1\|_g$ and $\|G\|_g > \|G_2\|_g$*

Proof. By definition of the series-parallel graphs, the first case considered is obvious, and implies that for all series-parallel graph G, $\|G\|_g \geq 3$. In the second case, we just have to note that if G comes from the serialization of two graphs G_1 and G_2, then $\|G\|_g = \|G_1\|_g + \|G_2\|_g - 1$, and if G has been obtained by parallelization of G_1 and G_2 then, $\|G\|_g = \|G_1\|_g + \|G_2\|_g - 2$. □

5.1. *Calculation of the latest starting dates*

When the activity network is series-parallel, (we will speak about series-parallel networks) we can construct two particular extreme configurations which do maximize and minimize the latest starting date of a given task:

Proposition 12. Let R be a series-parallel network, i a task of R, Ω_1^G and Ω_2^G the two extreme configurations such that:

$$d_j(\Omega_1^G) = \begin{cases} d_j^+ \ if j \in SUCC(i) \cup \{i\} \\ d_j^- \ otherwise \end{cases}$$

$$d_j(\Omega_2^G) = \begin{cases} d_j^- \ if j \in SUCC(i) \cup \{i\} \\ d_j^+ \ otherwise \end{cases}$$

Then,

(i) Ω_1^G *minimizes the latest starting date of i.*
(ii) Ω_2^G *maximizes the latest starting date of i.*

Proof. See Appendix A.

This proposition validates Algorithm 2 proposed by Dubois *et al.*, (2003b) for the particular case of the strongly series-parallel graphs. It computes the interval of possible latest starting dates of a task i in a linear time. This algorithm has a time complexity in $O(n + m)$ as it computes two classical PERT/CPM on two special configurations.

5.2. *Calculation of the floats*

As for the latest starting dates, the float of a task in a series-parallel network can be computed from only two special configurations:

Proposition 13. Let R be a series-parallel network, i a task of R, Ω_1^G and Ω_2^G two configurations such that:

$$d_j(\Omega_1^G) = \begin{cases} d_j^+ \ if j \in PRED(i) \cup SUCC(i) \cup \{i\} \\ d_j^- \ otherwise \end{cases}$$

$$d_j(\Omega_2^G) = \begin{cases} d_j^- \ if j \in PRED(i) \cup SUCC(i) \cup \{i\} \\ d_j^+ \ otherwise \end{cases}$$

Then,

(i) Ω_1^G *minimizes the float of i.*
(ii) Ω_2^G *maximizes the float of i.*

Proof. The sketch of the proof is the same as in Proposition 12 for the calculation of the latest starting dates. □

Proposition 13 permits to validate Algorithm 3 of Dubois *et al.*, (2003b) and this time for general series-parallel graphs. Its complexity is the same as Algorithm 2, in $O(n+m)$.

6. Conclusion

In this paper, some new methods for the calculation of latest starting dates and floats in interval-valued PERT/CPM networks have been described. Some linear complexity algorithms are also presented for series-parallel network. The proposed algorithm, called path-algorithm as it relies on a path enumeration technique has been tested on general scheduling problems. Experimental results

Algorithm 2: Calculation of the latest starting date of a given task

```
Input: A series-parallel network $R = \langle \tau, C, T \rangle$;
       A task $i$;
Output: $LST_i = [lst_i^-, lst_i^+]$ the interval of the possible latest starting date of $i$;
begin
    foreach $j \in \tau$ do
        if $j \in SUCC(i) \cup \{i\}$ then $d_j(\Omega) = d_j^+$;
        else $d_j(\Omega) = d_j^-$;
    end
    Compute $lst_i(\Omega)$ by a classical PERT/CPM;
    $lst_i^- = lst_i(\Omega)$;
    foreach $j \in \tau$ do
        if $j \in SUCC(i) \cup \{i\}$ then $d_j(\Omega) = d_j^-$;
        else $d_j(\Omega) = d_j^+$;
    end
    Compute $lst_i(\Omega)$ by a classical PERT/CPM;
    $lst_i^+ = lst_i(\Omega)$;
end
```

Algorithm 3: Calculation of the float of a given task

```
Input: A series-parallel network $R = \langle \tau, C, T \rangle$;
       A task $i$;
Output: $F_i = [f_i^-, f_i^+]$ the interval of the possible floats of $i$;
begin
    foreach $j \in \tau$ do
        if $j \in SUCC(i) \cup \{i\} \cup PRED(i)$ then $d_j(\Omega) = d_j^+$;
        else $d_j(\Omega) = d_j^-$;
    end
    Compute $f_i(\Omega)$ by a classical PERT/CPM;
    $f_i^- = f_i(\Omega)$;
    foreach $j \in \tau$ do
        if $j \in SUCC(i) \cup \{i\} \cup PRED(i)$ then $d_j(\Omega) = d_j^-$;
        else $d_j(\Omega) = d_j^+$;
    end
    Compute $f_i(\Omega)$ by a classical PERT/CPM;
    $f_i^+ = lst_i(\Omega)$;
end
```

prove the interest of the algorithm as it can compute in a reasonable time latest starting dates and floats for problems with more than one hundred tasks. They have been compared with polynomials algorithms which computes only LUB and GLB of the latest starting dates, and the time execution is quite similar, which confirms the practical interest of the path-algorithm. Moreover, the principle of the path algorithm is simpler than the polynomial algorithms for latest starting times, and it copes with the exponential nature of the float determination problem. It seems possible to further develop some heuristics that find "good" configurations faster by means of *Branch&Bound* algorithm.

Another research line is the generalization of the path-algorithm algorithm to problems where tasks durations are represented by fuzzy intervals instead of clearcut ones.

Appendix A: proofs

Lemma 1

Proof. Let Ω^* be a configuration which minimizes $lst_i(.)$ it must be such such that:

$$d_j(\Omega^*) = \begin{cases} d_j^- \ or \ d_j^+ \ if \ j \in SUCC(i) \cup \{i\} \\ d_j^- \qquad otherwise \end{cases}$$

According to Proposition 5, such a configuration always exists. From equation 12 and by construction:

$$lst_i(\Omega^*) = max_{p\in P_{1,n}} W_p(\Omega^*) - max_{p\in P_{i,n}} W_p(\Omega^*)$$

Let p^* be a path of $P_{i,n}$ such that $W_{p^*}(\Omega^*) = max_{p\in P_{i,n}} W_p(\Omega^*)$. We define the configuration Ω^- such that: $d_j(\Omega^-) = \begin{cases} d_j(\Omega^*) & if \ j \in p^* \\ d_j^- & otherwise \end{cases}$.
Since for all $j \in \tau$, $d_j(\Omega^-) \le d_j(\Omega^*)$ then for $p \in P$, $W_p(\Omega^-) \le W_p(\Omega^*)$, and $max_{p\in P_{1,n}} W_p(\Omega^-) \le max_{p\in P_{1,n}} W_p(\Omega^*)$. Now, $max_{p\in P_{i,n}} W_p(\Omega^-) = W_{p^*}(\Omega^*) = max_{p\in P_{i,n}} W_p(\Omega^*)$. Hence $lst_i(\Omega^-) \le lst_i(\Omega^*)$.

Since Ω^* minimizes $lst_i(.)$, then $lst_i(\Omega^-) = lst_i(\Omega^*)$, and we can deduce that Ω minimizes $lst_i(.)$ too. And moreover in Ω^-, all tasks j outside p^* have a task duration assigned to d_j^-. So, any configuration Ω^*minimizing lst_i can be changed into one of the form prescribed in the lemma. Only such kinds of configuration need to be explored.

Proposition 6.
Proof: Let Ω^- be a configuration which minimizes $lst_i(.)$ and path $p^* \in P_{i,n}$ such that: $d_j(\Omega^-) = d_j^- \ \forall \ j \notin p^*$, p^* is such that $W_{p^*}(\Omega^-) = max_{p\in P_{i,n}} W_p(\Omega^-)$. From Lemma 1, such a configuration always exists.
We can now define the configuration Ω_{p^*} such that:

$$d_j(\Omega_{p^*}) = \begin{cases} d_j^+ & if \ j \in p^* \\ d_j^- & otherwise \end{cases}$$

We know that:

$$\begin{aligned} lst_i(\Omega^-) &= max_{p\in P_{1,n}} W_p(\Omega^-) - max_{p\in P_{i,n}} W_p(\Omega^-) \\ &= max_{p\in P_{1,n}} W_p(\Omega^-) - W_{p^*}(\Omega^-) \end{aligned}$$

So let p be a path from $P_{1,n}$. We denote $p \cap p^* = \{i \in \tau | i \in p$ and $i \in p^*\}$, and $p \setminus p^* = \{i \in \tau | i \in p$ and $i \notin p^*\}$. Now, we can write:

$$\begin{aligned} W_p(\Omega^-) - W_{p^*}(\Omega^-) &= \textstyle\sum_{j\in p} d_j(\Omega^-) - \sum_{j\in p^*} d_j(\Omega^-) \\ &= \textstyle\sum_{j\in p\setminus p^*} d_j(\Omega^-) \\ &\quad + \textstyle\sum_{j\in p\cap p^*} d_j(\Omega^-) \\ &\quad - \textstyle\sum_{j\in p^*\setminus p} d_j(\Omega^-) \\ &\quad - \textstyle\sum_{j\in p^*\cap p} d_j(\Omega^-) \\ &= \textstyle\sum_{j\in p\setminus p^*} d_j(\Omega^-) \\ &\quad - \textstyle\sum_{j\in p^*\setminus p} d_j(\Omega^-) \end{aligned}$$

And of course, in the same way, we calculate:
$W_p(\Omega_{p^*}) - W_{p^*}(\Omega_{p^*}) = \sum_{j\in p\setminus p^*} d_j(\Omega_{p^*}) - \sum_{j\in p^*\setminus p} d_j(\Omega_{p^*})$

And because for all $j \in p \setminus p^*$, $d_j(\Omega_{p^*}) = d_j^- = d_j(\Omega^-)$, we know that $\sum_{j\in p\setminus p^*} d_j(\Omega^-) = \sum_{j\in p\setminus p^*} d_j(\Omega_{p^*})$.
And for all $j \in p^* \setminus p$, $d_j(\Omega_{p^*}) \ge d_j(\Omega^-)$ then $\sum_{j\in p^*\setminus p} d_j(\Omega_{p^*}) \ge \sum_{j\in p^*\setminus p} d_j(\Omega^-)$ (because $j \in p^*$), we deduce that:
$W_p(\Omega^-) - W_{p^*}(\Omega^-) \ge W_p(\Omega_{p^*}) - W_{p^*}(\Omega_{p^*})$ And then, because this last equation holds for all the paths $p \in P_{1,n}$, we can deduce that $lst_i(\Omega^-) \ge lst_i(\Omega_{p^*})$. Since Ω^- minimizes $lst_i(.)$, we can conclude that $lst_i(\Omega^-) = lst_i(\Omega_{p^*})$ and then that Ω_{p^*} minimizes $lst_i(.)$. □

Proposition 7.
Proof: Let Ω^* be a configuration which maximizes $lst_i(.)$. It is such that:

$$d_j(\Omega^*) = \begin{cases} d_j^- \ or \ d_j^+ \ if \ j \in SUCC(i) \cup \{i\} \\ d_j^+ \qquad otherwise \end{cases}.$$

Such a configuration always exist (see Proposition 5). By construction:
$lst_i(\Omega^*) = max_{p\in P_{1,n}} W_p(\Omega^*) - max_{p\in P_{i,n}} W_p(\Omega^*)$
Let $p^* \in P$ be a path such that $W_{p^*}(\Omega^*) = max_{p\in P_{1,n}} W_p(\Omega^*)$. Therefore, we know that:

$$\begin{aligned} lst_i(\Omega^*) &= max_{p\in P_{1,n}} W_p(\Omega^*) - max_{p\in P_{i,n}} W_p(\Omega^*) \\ &= W_{p^*}(\Omega^*) - max_{p\in P_{i,n}} W_p(\Omega^*) \end{aligned}$$

Then define the configuration Ω^+ such that:

$$d_j(\Omega^+) = \begin{cases} d_j^+ \ if \ j \in p^* \\ d_j^- \ otherwise \end{cases}$$

Then, we can write:

$$\begin{aligned} lst_i(\Omega^+) &= max_{p\in P_{1,n}} W_p(\Omega^+) - max_{p\in P_{i,n}} W_p(\Omega^+) \\ &\ge W_{p^*}(\Omega^+) - max_{p\in P_{i,n}} W_p(\Omega^+) \end{aligned}$$

So let p be a path from $P_{i,n}$. We can write:

$$\begin{aligned} W_{p^*}(\Omega^*) - W_p(\Omega^*) &= \textstyle\sum_{j\in p^*} d_j(\Omega^*) - \sum_{j\in p} d_j(\Omega^*) \\ &= \textstyle\sum_{j\in p^*\setminus p} d_j(\Omega^*) \\ &\quad + \textstyle\sum_{j\in p^*\cap p} d_j(\Omega^*) \\ &\quad - \textstyle\sum_{j\in p\setminus p^*} d_j(\Omega^*) \\ &\quad - \textstyle\sum_{j\in p\cap p^*} d_j(\Omega^*) \\ &= \textstyle\sum_{j\in p^*\setminus p} d_j(\Omega^*) \\ &\quad - \textstyle\sum_{j\in p\setminus p^*} d_j(\Omega^*) \end{aligned}$$

And also: $W_{p^*}(\Omega^+) - W_p(\Omega^+) = \sum_{j\in p^*\setminus p} d_j(\Omega^+) - \sum_{j\in p\setminus p^*} d_j(\Omega^+)$.
For all $j \in p^*$, $d_j(\Omega^+) = d_j^+ \ge d_j(\Omega^*)$, then $\sum_{j\in p^*\setminus p} d_j(\Omega^+) \ge \sum_{j\in p^*\setminus p} d_j(\Omega^*)$.
And for all $j \in p \setminus p^*$, $j \notin p^*$ then $d_j(\Omega^+) = d_j^- \le$

$d_j(\Omega^*)$, and so, $\sum_{j\in p\backslash p^*} d_j(\Omega^*) \le \sum_{j\in p\backslash p^*} d_j(\Omega^+)$. We can conclude that: $W_{p^*}(\Omega^*) - W_p(\Omega^*) \le W_{p^*}(\Omega^+) - W_p(\Omega^+)$

Now, this last inequality for all paths $p \in P_{i,n}$, hence $lst_i(\Omega^*) \le lst_i(\Omega^+)$. Since Ω^* maximizes $lst_i(.)$, we can conclude that $lst_i(\Omega^*) = lst_i(\Omega^+)$ and then that Ω^+ maximizes $lst_i(.)$. □

Proposition 8.

Proof: Let Ω^+ and p^* be the configuration and the path of the Proposition 7.

Let Ω^{p^*} be the configuration such that:

$$d_j(\Omega^{p^*}) = \begin{cases} d_j^+ & if\ j \notin SUCC(i) \cup \{i\} \\ d_j^+ & if\ j \in p^* \\ d_j^- & otherwise. \end{cases}$$

Then we can write $lst_i(\Omega^{p^*}) = max_{p\in P_{1,n}} W_p(\Omega^{p^*}) - max_{p\in P_{i,n}} W_p(\Omega^{p^*})$. By construction of Ω^{p^*}, for all $p \in P$,

$W_p(\Omega^{p^*}) \ge W_p(\Omega^+)$ then $max_{p\in P_{1,n}} W_p(\Omega^{p^*}) \ge max_{p\in P_{1,n}} W_p(\Omega^+)$.

And for all $p \in P_{i,n}$, $W_p(\Omega^{p^*}) = W_p(\Omega^+)$, hence $max_{p\in P_{i,n}} W_p(\Omega^{p^*}) = max_{p\in P_{i,n}} W_p(\Omega^+)$.

Then $lst_i(\Omega^{p^*}) \ge lst_i(\Omega^+)$, hence Ω^{p^*} maximizes $lst_i(.)$. □

Proposition 12.

Proof: By recursion on the size of R.

When the graph G of R contains only two nodes ($\|G\|_g = 3$), Propositions (i) and (ii) are obviously true. Now suppose that Propositions (i) and (ii) are true for all networks such that the associated graph G' verifies $\|G'\|_g \le k$ $(k > 2)$. Then let us show that they are also true for every network in which the associated graph G verifies $\|G\|_g = k+1$.

In the following, $\alpha(G)$ will denote the initial node of G, $\omega(G)$ the final node, $P_{k,l}^G$ (or $P_{k,l}$ if there is no ambiguity) the set of all paths from node k to node l in the graph G, and $lst_k^G(\Omega)$ the latest starting date of k for configuration Ω in the graph G.

Let us remember that Equation (12) computes the latest starting date with the help of the paths:

$$lst_i(\Omega) = max_{p\in P_{\alpha(G),\omega(G)}} W_p(\Omega) - max_{p\in\ P_{i,\omega(G)}} W_p(\Omega)$$

- If i is the initial task of G, then the latest starting date of i is null for all the configurations then this date is minimized by Ω_1^G, and maximized by Ω_2^G defined in the proposition.
- If i is the final task of G, then according to the previous equation, we deduce:
  $lst_i(\Omega) = max_{p\in P_{\alpha(G),\omega(G)}} W_p(\Omega) - d_{\omega(G)}(\Omega)$
  Now $W_p(\Omega) = \sum_{j\in p} d_j$ and moreover for all $p \in P_{\alpha(G),\omega(G)}$, we know that $\omega(G) \in P$ so we obtained $lst_{\omega(G)}(\Omega) = max_{p\in P_{\alpha(G),\omega(G)}} \sum_{j\in p\backslash\{\omega(G)\}} d_j$ and we can conclude that Ω_1^G minimizes the latest starting date of $\omega(G)$, and Ω_2^G maximizes this latest starting date.
- If i is neither the initial task, nor the final one of G, as G is series-parallel and G has more than two nodes, according to Lemma 2, G can be decomposed into two sub-graphs G_1 and G_2 such that G comes from the parallelization or the serialization of G_1 and G_2 with $\|G_1\|_g \le k$ and $\|G_2\|_g \le k$.

○ *First case: parallelization*

We can suppose without loss of generality that i is in G_1. By definition of parallelization, $\alpha(G) = \alpha(G_1) = \alpha(G_2)$ and $\omega(G) = \omega(G_1) = \omega(G_2)$, then i is neither the initial task, nor the final one of G_1. Then we can write the following equations:

$$\begin{aligned} lst_i^G(\Omega) &= max_{p\in P_{\alpha(G),\omega(G)}} W_p(\Omega) \\ &\quad - max_{p\in P_{i,\omega(G)}} W_p(\Omega) \\ &= max\big(max_{p\in P^{G_1}_{\alpha(G_1),\omega(G_1)}} W_p(\Omega), \\ &\qquad max_{p\in P^{G_2}_{\alpha(G_2),\omega(G_2)}} W_p(\Omega)\big) \\ &\quad - max_{p\in P^{G_1}_{i,\omega(G_1)}} W_p(\Omega) \end{aligned}$$

- If $max_{p\in P^{G_1}_{\alpha(G_1),\omega(G_1)}} W_p(\Omega) \ge max_{p\in P^{G_2}_{\alpha(G_2),\omega(G_2)}} W_p(\Omega)$ then
  $lst_i^G(\Omega) = lst_i^{G_1}(\Omega)$, and by recursion hypothesis, $\Omega_1^{G_1}$ minimizes this latest starting date. Now, for all nodes k of the graph G_1, $d_k(\Omega_1^{G_1}) = d_k(\Omega_1^G)$. Then Ω_1^G minimizes the latest starting date of i in G. In the same way, Ω_2^G maximizes this latest starting date.
- If $max_{p\in P^{G_1}_{\alpha(G_1),\omega(G_1)}} W_p(\Omega) \le max_{p\in P^{G_2}_{\alpha(G_2),\omega(G_2)}} W_p(\Omega)$ then

$$lst_i^G(\Omega) = max_{p\in P^{G_2}_{\alpha(G_2),\omega(G_2)}} W_p(\Omega) - max_{p\in P^{G_1}_{i,\omega(G_1)}} W_p(\Omega)$$

Now $\omega(G)$ is the only task which is part of a path of $P^{G_2}_{\alpha(G_2),\omega(G_2)}$ and of $P^{G_1}_{i,\omega(G_1)}$ Moreover, $\omega(G)$ is part of all those paths, and so we can write:

$$lst_i^G(\Omega) = max_{p\in P^{G_2}_{\alpha(G_2),\omega(G_2)}} \sum_{j\in p\backslash\{\omega(G)\}} d_j(\Omega) - max_{p\in P^{G_1}_{i,\omega(G_1)}} \sum_{j\in p\backslash\{\omega(G)\}} d_j(\Omega)$$

Now, for all $j \in p\backslash\{\omega(G)\}$ such that $p \in P^{G_2}_{\alpha(G),\omega(G)}$, we know that $j \notin SUCC(i)\cup\{i\}$, and more, for all $j \in p\backslash\{\omega(G)\}$ such that $p \in P^{G_1}_{i,\omega(G)}$, we know that $j \in SUCC(i)\cup\{i\}$, then, Ω_1^G minimizes

$$max_{p\in P^{G_2}_{\alpha(G_2),\omega(G_2)}} \sum_{j\in p\backslash\{\omega(G)\}} d_j(\Omega),$$

and maximizes

$$max_{p\in P^{G_1}_{i,\omega(G_1)}} \sum_{j\in p\backslash\{\omega(G)\}} d_j(\Omega).$$

Then Ω_1^G minimizes $lst_i^G(\Omega)$, and in the same way, Ω_2^G maximizes $lst_i^G(\Omega)$.

- *Second case: serialization*
  If G can be decomposed in two subgraphs by serialization, then we know that $\alpha(G) = \alpha(G_2)$, $\omega(G_2) = \alpha(G_1)$ and $\omega(G) = \omega(G_1)$. Let us note that if a path p begins in G_2 and finishes in G_1, then p can be decomposed into two paths p_1 and p_2 such that p_1 is a path of G_2, p_2 is a path of G_1, and the initial node of p_2 is $\alpha(G_1)$ and the final node of p_1 is $\omega(G_2)$. So we can write $W_p(\Omega) = W_{p_1}(\Omega) + W_{p_2}(\Omega) - d_{\alpha(G_1)}$. We have to subtract $d_{\alpha(G_1)}$ to not count it twice.

  - If i is part of G_1 and only G_1 then:

$$lst_i^G(\Omega) = max_{p\in P^{G_2}_{\alpha(G),\omega(G_2)}} W_p(\Omega) + max_{p\in P^{G_1}_{\alpha(G_1),\omega(G)}} W_p(\Omega) - d_{\alpha(G_1)}(\Omega) - max_{p\in P^{G_1}_{i,\omega(G)}} W_p(\Omega)$$

$$Solst_i^G(\Omega) = max_{p\in P^{G_2}_{\alpha(G),\omega(G_2)}} W_p(\Omega) - d_{\alpha(G_1)}(\Omega) + lst_i^{G_1}(\Omega)$$

Now $\Omega_1^{G_2}$ minimizes $max_{p\in P_{\alpha(G),\omega(G_2)}} W_p(\Omega) - d_{\alpha(G_1)}(\Omega)$ and by recursion hypothesis $\Omega_1^{G_1}$ minimizes $lst_i^{G_1}(\Omega)$. So Ω_1^G minimizes $lst_i^G(\Omega)$ (because for all nodes l of G_1, $d_l(\Omega_1^{G_1}) = d_l(\Omega_1^G)$)) and Ω_2^G maximizes $lst_i^G(\Omega)$.

  - If i is in G_2, then

$$lst_i^G(\Omega) = max_{p\in P^{G_2}_{\alpha(G),\omega(G_2)}} W_p(\Omega) + max_{p\in P^{G_1}_{\alpha(G_1),\omega(G)}} W_p(\Omega) - d_{\alpha(G_1)}(\Omega) - max_{p\in P^{G_1}_{\alpha(G_1),\omega(G)}} W_p(\Omega) - max_{p\in P^{G_2}_{i,\omega(G_2)}} W_p(\Omega) + d_{\alpha(G_1)}(\Omega)$$

If we simplify, we have:

$$lst_i^G(\Omega) = max_{p\in P^{G_2}_{\alpha(G),\omega(G_2)}} W_p(\Omega) - max_{p\in P^{G_2}_{i,\omega(G_2)}} W_p(\Omega)$$

And then $lst_i^G(\Omega) = lst_i^{G_2}(\Omega)$, and according to the recurrsion hypothesis, we can conclude that $\Omega_1^{G_2}$ minimizes $lst_i^{G_2}(\Omega)$ and that $\Omega_2^{G_2}$ maximizes $lst_i^{G_2}(\Omega)$. Finally, Ω_1^G minimizes $lst_i^G(\Omega)$ and Ω_2^G maximizes $lst_i^G(\Omega)$.

In all cases, we have shown that configuration Ω_1^G (respectively, Ω_2^G) minimizes (respectively maximizes) the latest starting date of i, for any network G such that $\|G\|_g = k+1$, then the Propositions (i) and (ii) are proved by recursion for all sizes of graphs.

References

Bein, W. W., Kamburowski, J. and Stallmann, M. F. M. (1992) Optimal reductions of two-terminal directed acyclic graphs. *SIAM Journal of Computation*, **21**(6), 1112–1129.

Bellman, R., Esogbue, A. and Nabeshima, I. (1982) *Mathematical Aspects of Scheduling and Applications*, Pergamon Press, Oxford, UK.

Brige, J. and Dempster, M. (1996) Stochastic programming approaches to stochastic scheduling. *Global Optimisation*, **9**, 383–409.

Buckley, J. (1989) Fuzzy PERT in *Applications of Fuzzy set Methodologies in Industrial Engineering*, G. W. Evans, W. Karwowski and M. R. Wilhelm (eds), Elsevier, Amsterdam, pp. 103–114.

Chanas, S., Dubois, D. and Zielinski, P. (2002) On the sure criticality of tasks in activity networks with imprecise durations. *IEEE Transactions on Systems, Man, and Cybernetics*, **34**, 393–407.

Chanas, S. and Kamburowski, J. (1981) The use of fuzzy variables in PERT. *Fuzzy Set and Systems*, **5**, 1–19.

Chanas, S. and Zielinski, P. (2003) On the hardness of evaluating criticality of activities in planar network with duration intervals. *Operations Research Letters*, **31**, 53–59.

Dubois, D. (1983) Modèles mathématiques de l'imprécis et de l'incertain en vue d'applications aux techniques d'aide à la décision Thèse d'État, Université Scientifique et Médicale de Grenoble, France.

Dubois, D., Fargier, H. and Fortemps, P. (2003a) Fuzzy scheduling: modeling flexible constraints vs. coping with incomplete knowledge. *European Journal of Operational Research*, **147**, 231–252.

Dubois, D., Fargier, H. and Galvagnon, V. (2003b) On latest starting times and floats in activity networks with ill-known durations. *European Journal of Operational Research*, **147**, 266–280.

Dubois, D. and Prade, H. (1980) Fuzzy models for operations research. *Fuzzy Sets and Systems, Theory and Applications*, Academic Press, chapter III-4, pp. 242–251.

Eppstein, D. (1992) Parallel recognition of series-parallel graphs. *Information and Computation*, **98**, 41–55.

Kolisch, R. and Sprecher, A. (1996) Psplib – a project scheduling library. *European Journal of Operational Research*, **96**, 205–216.

Kolisch, R., Sprecher, A. and Drexl, A. (1995) Characterization and generation of a general class of resource-constrained project scheduling problems. *Management Science*, **41**, 1693–1703.

Loostma, F. (1989) Stochastic and fuzzy PERT. *European Journal of Operational Research*, **43**, 174–183.

Loostma, F. (1997) *Fuzzy Logic for Planning and Decision-Making*, Kluwer Academic Publisher, Dordrecht.

Prade, H. (1979) Using fuzzy set theory in a scheduling problem: a case study. *Fuzzy Sets and Systems*, **2**, 153–165.

Zielinski, P. (2005) On computing latest starting times and floats of activities in networks with imprecise durations. *Fuzzy Sets and Systems*, **150**, 53–76.