Criticality Analysis of Activity Networks under Interval Uncertainty

Jérôme Fortin, Pawel Zielinski, Didier Dubois, Hélène Fargier

Criticality analysis of activity networks under interval uncertainty *†

Jérôme Fortin
LIRMM,
Université Montpellier 2 and CNRS,
161 rue Ada, 34392 Montpellier
Cedex 5 France
fortin@lirmm.fr

Paweł Zieliński
Institute of Mathematics and Computer Science,
Wrocław University of Technology,
Wybrzeże Wyspiańskiego 27,
50-370 Wrocław, Poland
pawel.zielinski@pwr.wroc.pl

Didier Dubois Hélène Fargier
I.R.I.T., Université Paul Sabatier,
118 route Narbonne, 31062 Toulouse Cedex 4, France,
{dubois,fargier}@irit.fr

Abstract

This paper reconsiders the PERT scheduling problem when information about task duration is incomplete. We model uncertainty on task durations by intervals. With this problem formulation, our goal is to assert possible and necessary criticality of the different tasks and to compute their possible earliest starting dates, latest starting dates and floats. This paper puts together various results and provides a complete solution to the problem. We present the complexity results of all considered subproblems and efficient algorithms to solve them.

Keywords: Project scheduling; Interval PERT; Criticality; Interval uncertainty

1 Introduction

An activity network is a partially ordered set of activities with given duration times, forming a directed acyclic graph. One basic problem when scheduling an activity network representing a project, is that of finding critical activities, and determining optimal starting times of activities, so as to minimize the makespan. The first step is to determine the earliest ending time of the project. This problem was posed in the fifties, in the framework of project management, by Malcolm *et al.* [37] and the basic underlying graph-theoretic approach, called Project Evaluation and Review Technique, is now popularized under the acronym PERT. The determination of critical activities is carried out via the so-called critical path method (Kelley [31]). The usual assumption in scheduling is that the duration of each task is precisely known, so that solving the PERT problem is rather simple. However, in project management, the durations of tasks are seldom precisely known in advance, at the time when the plan of the project is designed. Detailed specifications of the methods and resources involved for the realization of activities are often not available when the tentative plan is made up. This difficulty has been noticed very early by the authors that introduced the PERT approach. They proposed to model the duration of tasks by probability distributions, and tried to estimate the mean value and standard deviation of earliest

*Dedicated to the memory of Professor Stefan Chanas.

†The extended abstract version of this paper has appeared in *Proceedings of 11th International Conference on Principles and Practice of Constraint Programming* (CP2005) [22]. Paweł Zieliński was partially supported by Polish Committee for Scientific Research, grant 3T11C05430.

starting times of activities. Since then, there is an extensive literature on probabilistic PERT (see Elmaghraby [16], and Adlakha and Kulkarni [1] and Elmaghraby [17] for a bibliography and recent views). Even if the task duration times are independent random variables, it is admitted that the problem of finding the distribution of the ending time of a project is intractable, due to the dependencies induced by the topology of the network [24, 32, 36, 40, 45]. Another difficulty, not always pointed out, is the possible lack of statistical data validating the choice of activity duration distributions. In fact probability distributions permit to model the variability of repetitive tasks, but not uncertainty due to a lack of information [14, 19]. Even if statistical data are available, they may be partially inadequate because each project takes place in a specific environment, and is not the exact replica of past projects.

The simplest form of non-committal uncertainty representation for activity duration is the interval. Assigning some time interval I to an activity duration means that the actual duration of this activity will take some value within I, but it is not possible at present to predict which one. In this paper, every precise instantiation of duration times will be called a *configuration* (it is also called a scenario in the optimisation literature [34] and a possible world in formal logic). Using intervals for representing uncertainty in scheduling means that the part of uncertainty about task duration due to partial ignorance prevails because the available statistical information is considered too scarce or irrelevant. So our model is more adapted to unique projects involving non-repetitive tasks. The managerial value of intervals can be questioned. Indeed, experts may find it difficult to provide narrow intervals (because they may be wrong). Moreover using too wide intervals may make further analysis useless because non-informative. So, we do not propose the use of intervals as the definite answer to scheduling uncertainty. However the importance of studying interval PERT is motivated by several considerations:

- Interval uncertainty is present as soon as information is incomplete. So, solving the extreme case of pure intervals is a first step before considering more elaborate representations of uncertainty where both probability and intervals are combined.
- By and large, an expert working on a project may be reluctant to provide point values for the expected duration for each task. Allowing the collection of a minimal and a maximal duration may be felt more realistic, even if the elicitation process should force the expert to provide as narrow intervals as possible.
- A more elaborate approach could be to collect both intervals and plausible values from experts. Then the interval PERT results could be seen as providing safeguards on the expected behaviour of the project while a precise reference plan based on plausible estimates could still be used.

Note that resorting to subjective probability for eliciting duration times of non-repetitive tasks is not very convincing. Eliciting subjective probabilities would be burdensome to experts, and it is well-known that subjective probability is neither a faithful nor a stable representation of partial ignorance [14]. More precisely, if all that is known of an activity duration time d is that it lies between two bounds $a < b$, some may consider it natural to model this lack of knowledge by a uniform probability on $[a, b]$. However, if the scale were distorted by a monotonic function, say f, whereby $f(a) < f(d) < f(b)$, then it is clear that the resulting probability distribution on $[f(a), f(b)]$ would not be uniform. If one describes ignorance by uniform probabilities, then the rescaling function seems to generate information out of the blue. Moreover, it seems paradoxical to know the probability distribution precisely, while claiming ignorance. The interval representation stands for all probability distributions with support inside $[a, b]$, which can hardly be challenged as representing ignorance. Of course, following the betting procedure of Bayesians, ignorance forces the expert to bet using a uniform probability. The same betting rates would be produced if the expert knew that the values of d are indeed uniformly distributed. So, the subjective probabilities produced have an ambiguous meaning, and, under a cautious approach one should perhaps on the contrary interpret a subjective uniform probability as a mere interval, unless the expert knows that there is a genuine underlying random phenomenon. See [15] for a general approach to re-interpret subjective probabilities as expressing partial ignorance.

Strangely enough, the PERT analysis with ill-known processing times modeled by simple intervals does not seem to have received much attention in the literature. Yet, the computation of the minimal completion time of a project, the determination of critical paths and activities, the determination of activity floats have been considered as important problems and have been widely acknowledged to be pervaded with uncertainty. However the overwhelming part of the literature devoted to this topic adopts an orthodox stochastic approach, thus leading to a very complex problem that is still partially unsolved to-date [24]. To the best of the authors' knowledge, interval-valued PERT analysis seems to have existed only as a special case of fuzzy PERT studies that have been proposed since the late seventies [5, 10, 25, 26, 35, 39, 38, 42, 43, 44]. Recently, Conde [8] has used the *min-max regret criterion* to find a *robust* set of critical tasks in a project with interval task durations. This criterion is discussed in the book [34], which is entirely devoted to robust discrete optimization. In [8], exact and approximation algorithms are proposed to solve the min-max regret version of the longest path problem with interval weights in acyclic graphs (see [2, 27, 28] for a formulation and complexity results). A determined optimal min-max regret path, that minimizes the maximal deviation (regret) from optimum over all configurations, represents a robust set of critical tasks.

A basic result in standard (deterministic) activity network analysis states that a task is *critical* if and only if its earliest and latest starting dates are equal, and that critical tasks form critical paths, that is, longest paths from the initial node (event) to the final one. So finding the critical paths yields the critical tasks. When the tasks have ill-known durations, modeled by intervals, these results are no longer valid. Namely, floats can no longer be recovered from the intervals containing earliest and latest starting dates, and critical paths may no longer exist, even when critical tasks are present. Instead of being critical or not, as in the deterministic activity network analysis, three cases may be now observed for every task or path: it is either surely not critical (necessarily not critical) or surely critical (necessarily critical) or possibly critical. A task (a path) is *necessarily critical* if it is critical whatever the actual values of task durations turn out to be. A task (a path) is *possibly critical* if there are values of task durations leading to a configuration, in which it is critical. Necessarily critical paths may fail to exist while necessarily critical tasks may be isolated.

The notions of possible and necessary criticality of tasks and paths are closely related to the min-max regret version of the longest path problem in acyclic directed graphs with interval weights [27, 28]. Namely, every optimal min-max regret path is possibly critical and every necessary critical path is optimal min-max regret with zero maximal regret. Hence, an optimal min-max regret path can be viewed as a possibly critical one, which minimizes a "distance" to the necessary criticality. Furthermore, necessarily non-critical tasks can be removed from an instance of the min-max regret longest path without violating optimal min-max regret paths and necessarily critical tasks can be automatically added to a constructed optimal min-max regret path (see [29] for a deep discussion concerning the above relations).

This paper puts together various existing partial results about the complexity of finding intervals containing earliest and latest starting times and floats of tasks [6, 7, 11, 12, 47] and proposes new results, especially about evaluating necessary criticality, computing the maximal float of a task and computing the minimal latest starting times for all tasks, thus providing a complete solution to the PERT scheduling problem under the representation of interval-based uncertainty. A set of efficient algorithms for determining the criticality of tasks, the optimal intervals containing their earliest starting dates, latest starting dates, and their floats is provided. It is shown that the only strongly NP-hard problem is the one of finding minimal floats [7], which is closely related to asserting the possible criticality of a task that turned out to be strongly NP-complete [6]. All other problems turn out to be polynomial.

The paper is organized as follows. Section 2 motivates the work by showing the collapse of the usual criticality analysis concepts and techniques when duration times of tasks are imprecisely known. Section 3 proposes an exponential but practically efficient algorithm for achieving the complete criticality analysis under incomplete knowledge and a polynomial algorithm for computing the minima of latest starting dates for all task in a in network, based on an enumeration of paths. Section 4 presents polynomial algorithms for asserting necessary criticality of tasks, and

computing the optimal intervals containing their latest starting dates, as well as the maximum of float values. These algorithms are constructive in the sense that they select appropriate values of task duration times in a step by step manner, such that the computation of some bound comes down to computing a standard PERT problem. An illustrative example is provided in Section 5. Section 6 contains exhaustive complexity results and experimental evidence of the efficiency of algorithms.

2 Definition and challenge of the interval valued model

In this section, we briefly recall how to solve the classical PERT model. Then we formally introduce the interval-valued PERT model, and explain why the classical approach no longer works.

2.1 Solving standard PERT problem

An activity network is classically defined as a set of activities (or tasks) with given duration times, related to each other by means of precedence constraints. When there are no resource constraints, it can be represented by a directed, connected and acyclic graph $G =< V, A >$, where V is the set of nodes (events), $|V| = n$, and $A \subseteq V \times V$ is the set of arcs (activities), $|A| = m$. We use the activity-on-arc convention. The set $V = \{1, 2, \ldots, n\}$ is labeled in such a way that $i < j$ for each activity $(i, j) \in A$. Activity duration times d_{ij} (the weights of the arcs) $(i, j) \in A$ are known. Two nodes 1 and n are distinguished as the *initial* node (source) and *final* node (sink), respectively, (no activity enters 1 and no activity leaves n). Of major concern, is to minimize the ending time of the last task, also called the *makespan* of the network. Of interest to the project manager are *earliest starting dates*, *latest starting dates* and *floats* of activities. The *critical activities* have zero float. The essence of the PERT method are two recurrence formulae, *forward* and *backward* recursions. The earliest starting dates of events $k \in V$ and tasks $(k, l) \in A$, denoted by est_k and est_{kl} respectively, are determined by means of the forward recursion:

$$est_k = \begin{cases} 0 & \text{if } k = 1, \\ \max\limits_{j \in Pred(k)} (est_j + d_{jk}) & \text{otherwise,} \end{cases} \quad (1)$$

$$est_{kl} = est_k, \quad (2)$$

where $Pred(i)$ refers to the set of nodes that immediately precede node $i \in V$. The earliest starting date est_k of event k is the length of a longest path from the beginning of the project, represented by node 1, to node k. We arbitrarily fix the *starting time* of project to $est_1 = 0$. Of course the earliest starting date of a task (k, l) is equal to the earliest starting date of event k. The *earliest ending time* of the project is the earliest starting date of the last event n. In order to ensure minimal duration of the project, the latest date of event n, denoted by lst_n, is equal to its earliest starting date, $lst_n = est_n$. The latest starting date lst_k of event k is the latest time to complete tasks that are ended at k without delaying the end of the project and thus it is the *latest finishing date* of tasks ending at k. Difference $lst_n - lst_k$ is the length of a longest path from node k to node n. Hence, the latest starting date of task (k, l), denoted by lst_{kl}, is equal to $lst_l - d_{kl}$. The latest starting dates of events and tasks can be found by the use of the backward recursion:

$$lst_k = \begin{cases} est_n & \text{if } k = n, \\ \min\limits_{l \in Succ(k)} (lst_l - d_{kl}) & \text{otherwise,} \end{cases} \quad (3)$$

$$lst_{kl} = lst_l - d_{kl}, \quad (4)$$

where $Succ(i)$ refers to the set of nodes that immediately follow node $i \in V$. The float of task (k, l), which represents the length of the time window for the beginning of the execution of the task, is the difference between the latest starting date and the earliest starting date:

$$f_{kl} = lst_{kl} - est_{kl}. \quad (5)$$

The PERT method computes the earliest starting dates, latest starting dates and floats of tasks in $O(m+n)$ time. One can also determine, simultaneously, critical tasks and critical paths and thus a subgraph consisting of all critical tasks and paths.

In the next section, we will see that some obvious properties of this scheduling model such as: "there exists at least one critical path, which is a longest path in the network", or, "one can form a critical path with critical activities", or yet "each critical activity belongs to a critical path", do not hold anymore under incomplete information.

2.2 Interval valued problem definition

In practice, duration times of tasks are not well known for several reasons: Some design choices are not fixed because we design a predictive plan. We can also suppose that there can be some uncertainty about the duration times of tasks. It is clearly the case for subcontracted tasks. To take into account this uncertainty, we now model each task duration by an interval. It is easy for an expert working on a project to give a minimal and a maximal expected duration for each task. So activity duration times d_{ij} (weights of the arcs) $(i,j) \in A$ are only known to belong to time intervals $D_{ij} = [d_{ij}^-, d_{ij}^+]$, $d_{ij}^- \geq 0$. This means that we neither know the exact duration times of tasks, nor can we set them precisely. Now, depending on the effective duration of each task (that we do not precisely know), several earliest starting dates, latest starting dates and floats can be considered. We assume that the task durations are unrelated to one another. A vector $\Omega = (d_{ij})_{(i,j)\in A}$, $d_{ij} \in D_{ij}$, that represents an assignment of duration times d_{ij} to task $(i,j) \in A$ is called a *configuration* [3]. Thus every configuration expresses a realization of the duration times. We denote by $\mathscr{C}$ the set of all the configurations, i.e. $\mathscr{C} = \times_{(i,j)\in A}[d_{ij}^-, d_{ij}^+]$. The duration of task (i,j) in configuration Ω is denoted by $d_{ij}(\Omega)$, $d_{ij}(\Omega) \in D_{ij}$. Among the configurations of $\mathscr{C}$ a crucial role is played by the *extreme* ones, which belong to $\times_{(i,j)\in A}\{d_{ij}^-, d_{ij}^+\}$. Let $B \subseteq A$ be a given subset of activities. We define the extreme configuration Ω_B^+ as the configuration where all activities $(i,j) \in B$ have duration times d_{ij}^+ and all the remaining activities have duration times d_{ij}^-. Similarly, in configuration Ω_B^- all activities $(i,j) \in B$ have duration times d_{ij}^- and all the remaining activities have duration times d_{ij}^+.

Using configurations, the possible values $EST_{kl} = [est_{kl}^-, est_{kl}^+]$ for the earliest starting date est_{kl}, the possible values $LST_{kl} = [lst_{kl}^-, lst_{kl}^+]$ for the latest starting date lst_{kl} and the possible values $F_{kl} = [f_{kl}^-, f_{kl}^+]$ for the float f_{kl} can be rigorously defined as follows [12, 18]:

$$est_{kl}^- = \min_{\Omega\in\mathscr{C}} est_{kl}(\Omega), \quad est_{kl}^+ = \max_{\Omega\in\mathscr{C}} est_{kl}(\Omega), \tag{6}$$

$$lst_{kl}^- = \min_{\Omega\in\mathscr{C}} lst_{kl}(\Omega), \quad lst_{kl}^+ = \max_{\Omega\in\mathscr{C}} lst_{kl}(\Omega), \tag{7}$$

$$f_{kl}^- = \min_{\Omega\in\mathscr{C}} f_{kl}(\Omega), \quad f_{kl}^+ = \max_{\Omega\in\mathscr{C}} f_{kl}(\Omega), \tag{8}$$

where $est_{kl}(\Omega)$, $lst_{kl}(\Omega)$ and $f_{kl}(\Omega)$ denote the earliest and the latest starting date and the float of activity (k,l) in configuration Ω, respectively.

Criticality in interval-valued problems is defined as follows [4] : a task $(k,l) \in A$ (resp. a path p in network G from node 1 to node n) is *possibly critical* if there exists a configuration $\Omega \in \mathscr{C}$ in which (k,l) (resp. path p) is critical in the usual sense. A task $(k,l) \in A$ (resp. a path p in network G from 1 to n) is *necessarily critical* if (k,l) (resp. path p) is critical in the usual sense in all configurations $\Omega \in \mathscr{C}$.

In order to compute the intervals of possible values of earliest starting dates, latest starting dates and floats (see (6)-(8)), the same PERT algorithm has been traditionally used (see formulae (1)-(5)), the only difference being the use of the interval arithmetic instead of the classical arithmetic. For such a straightforward extension of the PERT algorithm, it turns out the forward recursion correctly computes the interval of possible earliest starting dates [5, 13, 23]. Indeed, one can obtain all the optimal lower and upper bounds of earliest starting dates of tasks by applying the forward recursion (formulae: (1), (2)) for configurations Ω_A^- and Ω_A^+, respectively. But the backward recursion fails to compute the set of possible latest starting dates [38, 39, 43] and in

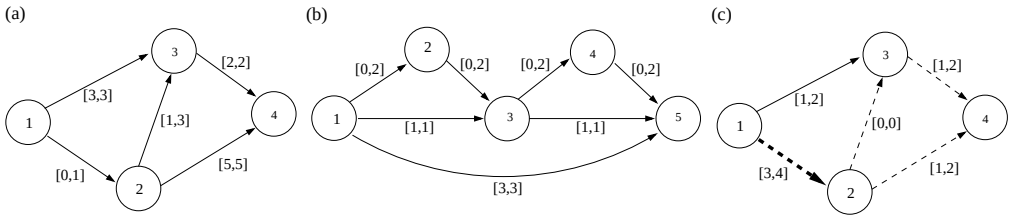


Figure 1: (a) A network in which the necessary criticality of $(3,4)$ cannot be deduced from $EST_{34} = LST_{34}$ (b) A network in which possibly critical activities do not form a possibly critical path (c) Dashed tasks form possibly critical paths, and the bold task is necessarily critical

consequence floats can no longer be recovered from this procedure. Let us illustrate the point that the backward recursion can not be used to compute the latest starting dates and the floats. We first recall definitions of arithmetical operations on intervals (see [41]). Let $* \in \{+, -, \min, \max\}$ and let $U = [u^-, u^+]$ and $W = [w^-, w^+]$ be two intervals such that $u^- \leq u^+$ and $w^- \leq w^+$. Then the following expression

$$Z = U *_I W = \{u * w \mid u \in U, w \in W\}$$

defines operation $*_I \in \{+_I, -_I, \min_I, \max_I\}$ on intervals and $Z = [z^-, z^+]$ is a resulting interval such $z^- \leq z^+$. The following formulae are instrumental when performing interval operations as defined above: $U +_I W = [u^- + w^-, u^+ + w^+]$, $U -_I W = [u^- - w^+, u^+ - w^-]$, $\min_I\{U, W\} = [\min\{u^-, w^-\}, \min\{u^+, w^+\}]$ and $\max_I\{U, W\} = [\max\{u^-, w^-\}, \max\{u^+, w^+\}]$.

Consider the one activity network with duration time $d_{12} \in D_{12} = [0,1]$. Its earliest starting date is $EST_1 = EST_{12} = [0,0]$. The earliest ending time of the project is ill-known: $EST_2 = EST_1 +_I D_{12} = [0,1]$. And thus the backward recursion yields the following information $EST_2 -_I D_{12} = [0,1] -_I [0,1] = [-1,1]$ on latest starting dates LST_1 and LST_{12}, which is of course false since the latest starting dates of event 1 and activity $(1,2)$ are always null in order to ensure a minimal project duration, $LST_1 = LST_{12} = [0,0] \subset EST_2 -_I D_{12} = [-1,1]$. This error comes from the fact that the variable which represents the duration appears two times in the computations: once in the forward recursion, and once in the backward one. Thus, applying interval arithmetic with linked variables always results in an over-imprecise bracketing of the exact result. For example, for all $x \in [0,1]$ $x - x = 0$, but $[0,1] -_I [0,1] = [-1,1]$. Some authors tried to repair this propagation error in several ways: one classical approach is the use of a non-standard interval arithmetic [26, 30, 44]. For example, some authors use for subtraction an "inverse" of the interval addition [26]. These operations are defined for fuzzy intervals, but their interval counterparts can be expressed as follows: $[u^-, u^+] - [w^-, w^+] = [u^- - w^-, u^+ - w^+]$. The use of this particular arithmetic does not lead to correct results for computing neither latest starting dates nor floats of tasks. In particular, the results may not be well formed intervals. For example, the difference between the intervals $[0,1]$ and $[0,2]$ would lead to $[0,-1]$ which is not an interval, $0 \not\leq -1$. Thus, techniques using non-standard interval arithmetic will generally fail to produce the correct values. In [42], symbolic computations on variable duration times were suggested. However, this technique is unwieldy and highly combinatorial. Accordingly, no complete, correct and efficient solution to the problem of scheduling under interval uncertainty has been reached up until now.

There are obvious relations between the criticality of a task and the possible values of its floats and its earliest and latest starting dates, namely, task (k,l) is possibly (resp. necessarily) critical if and only if $f_{kl}^- = 0$ (resp. $f_{kl}^+ = 0$). Moreover, if task (k,l) is necessarily critical then $EST_{kl} = LST_{kl}$. As shown in Figure 1a, the converse statement is false: task $(3,4)$ has $EST_{34} = LST_{34} = [3,4]$ for intervals of possible values of earliest and latest starting dates, but its interval of possible values of floats is $F_{34} = [0,1]$. It can be also noted that the float is no longer the difference between the latest starting date and the earliest starting date, for instance $LST_{34} -_I EST_{34} = [-1,1] \neq F_{34}$. In fact, when $EST_{kl} = LST_{kl}$, task (k,l) is only possibly critical. The last example makes it clear that the interval containing the floats of a task cannot 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.

Many intuitions learned on the deterministic problem fail in the interval version. For example, when duration times are precisely set, critical activities always form a critical path. But in the interval case, possibly critical activities do not always form a possibly critical path. In Figure 1b, task $(1, 3)$ is critical in the configuration where $d_{12} = d_{23} = 0$ and $d_{34} = d_{45} = 2$, and task $(3, 5)$ is critical in the configuration where $d_{12} = d_{23} = 2$ and $d_{34} = d_{45} = 0$, but the path $(1, 3, 5)$ is never critical. In Figure 1c, dashed arcs represent possibly critical tasks and the bold task $(1, 2)$ is the only necessarily critical task. So, it can be seen that some necessarily critical tasks can be isolated. There is no necessarily critical path, but several paths are possibly critical (paths consisting of dashed arcs in Figure 1c). However, it is easy to check that if there exists a necessarily critical path, then it is unique (under the assumption that all duration times are intervals) and it is then easy to identify [4]. But this case seems to appear rarely in practice. Therefore, the classical PERT results and algorithms cannot be directly adapted to the interval-valued problem.

The first important step for the resolution of the problems of computing the optimal bounds on the intervals of earliest starting dates, latest starting dates and floats (see (6)-(8)) was taken by Dubois *et al.* [12]. They showed that these bounds are attained on extreme configurations due to the monotonicity of functions that compute the quantities under concern. So a naive algorithm is to perform one PERT analysis on each extreme configuration which implies an $O((m + n)2^m)$ algorithm.

Other approaches have been developed to compute latest starting dates and floats. The first idea is to find a *small* subset of extreme configurations on which the bounds of the quantities of interest are attained. One such subset is defined by all extreme configurations in which all tasks have minimal duration times, but on a path from the initial node to the final one. On this path, task duration times are set to their upper bounds. The optimal upper bound on latest starting dates, the optimal lower and upper bounds on floats are reached on a configuration of this form. This idea will be presented in Section 3.

Another idea, called *constructive approach*, consists in a direct construction of one configuration (and just one) for which the extrema of the quantity of interest (for instance the optimal upper bound on latest starting dates of a task), is attained. In practice, we have to consecutively instantiate duration times of all tasks in a network without changing the result of the quantity of concern. The way of instantiation of a task duration to its least or greatest value is not simple. If we can assign precise duration times task by task in order to construct an extreme configuration, of course without changing the result of the quantity of concern, then this approach will lead to polynomial algorithms (see Section 4). Unfortunately, such approach cannot be applied to compute the optimal lower bound on floats of a task – this problem is known to be strongly NP-hard in a general acyclic network [6] and NP-hard in a planar acyclic network [7]. These results follow from the fact that the problem of deciding possible criticality of a given task is strongly NP-complete in a general acyclic network and remains NP-complete even in acyclic planar network of node degree 3 [6, 7]. Moreover, if P≠NP, then the problem of computing the minimal float value of a task is not at all approximable even in acyclic planar networks of node degree 3 [47].

It is worth pointing out that all considered interval problems have been completely solved when a network is series-parallel [12, 18, 48]. A graph is said to be *series-parallel* if it is recursively defined as follows [46]. A graph consisting of two nodes joined by a single arc is series-parallel. If G_1 and G_2 are series-parallel, so are the graphs constructed by each of the operations: *parallel composition* – identify the source of G_1 with the source of G_2 and the sink of G_1 with the sink of G_2; *series composition* – identify the sink of G_1 with the source of G_2. It results from this definition that each series-parallel graph is acyclic, planar and has exactly one source and exactly one sink. For example, the graph pictured in Figure 1b is series-parallel, but the graph shown in Figure 1c is not. Fargier *et al.* [18] proposed an $O(n)$ algorithm for computing bounds on possible values of latest starting dates and bounds on possible values of floats of a task and $O(n)$ algorithms for evaluating possible and necessary criticality of a task in a series-parallel network. Therefore, applying algorithms provided in [18] to each task in a network for computing bounds on floats and bounds on latest starting dates of all tasks leads to methods that require $O(n^2)$ time (note that

for series-parallel graphs $m = O(n)$). In [48], the time for computing bounds on floats and bounds on latest starting dates and for evaluating the possible and necessary criticality for one task in a series-parallel network has been reduced to $O(\log n)$ by applying a more clever data structure. Therefore, computing bounds on floats and bounds on latest starting dates and evaluating the possible and necessary criticality of all tasks in a series-parallel network requires $O(n \log n)$ time.

In order to deal with the interval PERT problem in the general case, we introduce some additional notations.

- $SUCC(i,j)$ (resp. $PRED(i,j)$) denotes the set of all arcs that come after (resp. before) $(i,j) \in A$, and $SUCC(j)$ (resp. $PRED(j)$) stands for the set of all nodes that come after (resp. before) $j \in V$.
- $P_{u,v}$ is the set of paths $p(u,v)$ in G from node u to node v. We denote by P the set of all paths in G from node 1 to node n.
- $P_{1,(k,l),n}$ is the set of paths in G from node 1 to node n traversing task (k,l) and $P_{(k,l),n}$ represents the set of paths in G from node k to node n traversing task (k,l).
- $l_p(\Omega)$ denotes the length of a path $p \in P_{u,v}$ in Ω, $l_p(\Omega) = \sum_{(i,j)\in p} d_{ij}(\Omega)$.
- $G(i,j)$ is the subgraph of G composed of nodes succeeding i and preceding j.
- $G(d_{ij} = d)$ is the graph G in which duration of task (i,j) is replaced by d.

3 Path enumeration approach

In this section, we give two algorithms based on some properties in which paths have particular interest. The first one, proposed in [11], computes maximal latest starting dates, minimal and maximal floats of each task of a network in one execution. We call it the *path algorithm*. The second algorithm only computes minimal latest starting dates in polynomial time. It is called the *polynomial path algorithm*.

3.1 Path algorithm

We first recall that maximal latest starting dates and minimal and maximal floats of tasks are attained on extreme configurations in which task duration times on a path from node 1 to node n are set to their maximal values and all the remaining task duration times are set to their minimal ones.

Proposition 1 ([11]). *Let $(k,l) \in A$ be a task of G. There exist paths $p_1, p_2, p_3 \in P$ such that $\Omega^+_{p_1}$ maximizes $lst_{kl}(.)$, $\Omega^+_{p_2}$ minimizes $f_{kl}(.)$, $\Omega^+_{p_3}$ maximizes $f_{kl}(.)$ over the set $\mathscr{C}$.*

The key to construct the path algorithm (Algorithm 1), proposed in [11], is Proposition 1. The idea of the algorithm consists in performing the classical PERT analysis for each configuration Ω^+_p such that p is a path in network G from 1 to n. Clearly, the number of tested configurations is potentially exponential, but in practice the algorithm runs fast on realistic problems (see Section 6).

3.2 Polynomial path algorithm

We first recall a result given in [11], that describes the form of configurations where the minimal latest starting date of a given task (k,l) in a network G is attained.

Proposition 2 ([11]). *Let $(k,l) \in A$ be a task of network G. There exists a path $p_{kl} \in P_{(k,l),n}$ that induces the extreme configuration such that $\Omega^+_{p_{kl}} = \arg\min_{\Omega\in\mathscr{C}} lst_{kl}(\Omega)$ and $lst^-_{kl} = lst_{kl}(\Omega^+_{p_{kl}})$.*

Algorithm 1: [PATH ALGORITHM] A calculation of the maximal latest starting dates, the minimal and maximal floats of each task in network G

Input: A network G, interval duration times D_{uv}, $(u,v) \in A$.
Output: The maximal latest starting dates, the minimal and maximal floats of each task in G.
begin
  foreach *task* $(k,l) \in A$ **do**
    $lst^+_{kl} \leftarrow 0$; $f^+_{kl} \leftarrow 0$; $f^-_{kl} \leftarrow +\infty$;
  foreach *path* $p \in P$ **do**
    Compute $lst_{kl}(\Omega^+_p)$ and $f_{kl}(\Omega^+_p)$ for each $(k,l) \in A$ by the classical PERT;
    foreach *task* $(k,l) \in A$ **do**
      if $lst^+_{kl} < lst_{kl}(\Omega^+_p)$ **then** $lst^+_{kl} \leftarrow lst_{kl}(\Omega^+_p)$;
      if $f^+_{kl} < f_{kl}(\Omega^+_p)$ **then** $f^+_{kl} \leftarrow f_{kl}(\Omega^+_p)$;
      if $f^-_{kl} > f_{kl}(\Omega^+_p)$ **then** $f^-_{kl} \leftarrow f_{kl}(\Omega^+_p)$;
end

It is easily seen that optimal path $p_{kl} \in P_{(k,l),n}$, which induces configuration $\Omega^+_{p_{kl}}$, is a longest path from k to n traversing l in $\Omega^+_{p_{kl}}$. We will show that this path can be recursively constructed. Suppose that a path $p_{lu} \in P_{(l,u),n}$ which induces configuration $\Omega^+_{p_{lu}}$ such that $\Omega^+_{p_{lu}} = \arg\min_{\Omega \in \mathscr{C}} lst_{lu}(\Omega)$ is known for each node $u \in Succ(l)$. Then one can construct an optimal path p_{kl} among paths p_{lu}, where $u \in Succ(l)$.

Proposition 3. *Let (k,l) be a task of network G and let $p_{lu} \in P_{(l,u),n}$ be a path such that $lst^-_{lu} = lst_{lu}(\Omega^+_{p_{lu}})$, where $u \in Succ(l)$. Then $lst^-_{kl} = \min_{u \in Succ(l)} lst_{kl}(\Omega^+_{\{(k,l)\} \cup p_{lu}})$.*

Proof. From Proposition 2, it follows that there exists a path $p_{kl} \in P_{(k,l),n}$ such that $\Omega^+_{p_{kl}} = \arg\min_{\Omega \in \mathscr{C}} lst_{kl}(\Omega)$. Assume on the contrary that $lst^-_{kl} = lst_{kl}(\Omega^+_{p_{kl}}) < lst_{kl}(\Omega^+_{\{(k,l)\} \cup p_{lu}})$, $\forall u \in Succ(l)$. Clearly, $lst^-_{kl} = lst_{kl}(\Omega^+_{p_{kl}}) = \max_{p \in P} l_p(\Omega^+_{p_{kl}}) - l_{p_{kl}}(\Omega^+_{p_{kl}})$ and thus

$$lst_{lu}(\Omega^+_{p_{kl}}) = \max_{p \in P} l_p(\Omega^+_{p_{kl}}) - \max_{p \in P_{(l,u),n}} l_p(\Omega^+_{p_{kl}}) = \max_{p \in P} l_p(\Omega^+_{p_{kl}}) - l_{p_{kl}} + d^+_{kl} = lst^-_{kl} + d^+_{kl}$$

which forces $lst^-_{kl} = lst_{lu}(\Omega^+_{p_{kl}}) - d^+_{kl}$. On the other hand, formula for $lst_{lu}(\Omega^+_{\{(k,l)\} \cup p_{lu}})$ can be rewritten: $lst_{lu}(\Omega^+_{\{(k,l)\} \cup p_{lu}}) = \max_{p \in P} l_p(\Omega^+_{\{(k,l)\} \cup p_{lu}}) - l_{p_{lu}}(\Omega^+_{\{(k,l)\} \cup p_{lu}}) - d^+_{kl}$. From the assumption, we have: $lst^-_{kl} = lst_{lu}(\Omega^+_{p_{kl}}) - d^+_{kl} < lst_{kl}(\Omega^+_{\{(k,l)\} \cup p_{lu}})$ which yields $lst_{lu}(\Omega^+_{p_{kl}}) < \max_{p \in P} l_p(\Omega^+_{\{(k,l)\} \cup p_{lu}}) - l_{p_{lu}}(\Omega^+_{\{(k,l)\} \cup p_{lu}})$. Let $p' \in P$ be a longest path in G in the configuration $\Omega^+_{\{(k,l)\} \cup p_{lu}}$. We consider two cases. (i) if $(k,l) \notin p'$ then $\max_{p \in P} l_p(\Omega^+_{\{(k,l)\} \cup p_{lu}}) = \max_{p \in P} l_p(\Omega^+_{p_{lu}})$ and so $lst_{lu}(\Omega^+_{p_{lu}}) < lst^-_{lu}$, which contradicts the definition of lst^-_{lu}. (ii) if $(k,l) \in p'$ then (k,l) is critical in $\Omega^+_{\{(k,l)\} \cup p_{lu}}$ and consequently $lst_{kl}(\Omega^+_{\{(k,l)\} \cup p_{lu}}) = est^-_{kl}$ and $lst_{kl}(\Omega^+_{\{(k,l)\} \cup p_{lu}}) = lst^-_{kl}$, which contradicts the fact that $lst_{kl}(\Omega^+_{p_{kl}}) < lst_kl(\Omega^+_{\{(k,l)\} \cup p_{lu}})$. Therefore, $lst^-_{kl} = lst_{kl}(\Omega^+_{p_{kl}}) \geq lst_{kl}(\Omega^+_{\{(k,l)\} \cup p_{lu}})$. □

From Proposition 3, we deduce a polynomial algorithm for computing the minimal starting date of each task. Namely, the algorithm recursively finds a path p_{kl} for which configuration $\Omega^+_{p_{kl}}$ minimizes $lst_{kl}(\Omega)$ making use of paths p_{lu} such that $lst^-_{lu} = lst_{lu}(\Omega^+_{p_{lu}})$, where $u \in Succ(l)$, starting from nodes in $Pred(n)$. Its overall running time is $O(m^2(m+n))$. Fortunately, it is possible to reduce the complexity of the algorithm to $O(m^2)$. It is enough to store the length of a longest path $p^* \in P$ in optimal configuration $\Omega^+_{p_{lu}}$ for (l,u), denoted by l^*_{lu}, and the length of path p_{lu} in $\Omega^+_{p_{lu}}$ denoted by l_{lu}, for every $u \in Succ(l)$. Obviously, $lst^-_{lu} = l^*_{lu} - l_{lu}$. Let (l,v) be a task such that $lst^-_{kl} = lst_{kl}(\Omega^+_{\{(k,l)\} \cup p_{lv}})$ (such task always exists, see Proposition 3). The length l_{kl} of path p_{kl} in optimal configuration $\Omega^+_{\{(k,l)\} \cup p_{lv}}$ for (k,l) equals $d^+_{kl} + l_{lv}$. Let $p^* \in P$ be a longest path in $\Omega^+_{\{(k,l)\} \cup p_{lv}}$. We consider two cases: (i) if $(k,l) \in p^*$ then the length l^*_{kl} of

p^* equals $est^-_{kl} + d^+_{kl} + l_{lv}$ (note that $est^-_{kl} = est_{kl}(\Omega^-_A) = est_{kl}(\Omega^+_{\{(k,l)\}\cup p_{lv}})$). Hence $lst^-_{kl} = est^-_{kl}$. (ii) if $(k,l) \notin p^*$ then $l^*_{kl} = l^*_{lv}$ and $lst^-_{kl} = l^*_{kl} - l_{kl}$. The above remarks allow us to construct an $O(m^2)$ algorithm (see Algorithm 2).

Algorithm 2: [POLYNOMIAL PATH ALGORITHM] Calculation of the minima of latest starting dates of all tasks in a network

Input: A network G, interval duration times D_{uv}, $(u,v) \in A$.
Output: The minima of latest starting dates of all the tasks in a network G.
begin
  $V \leftarrow V \cup \{n+1\}$; $A \leftarrow A \cup \{(n,n+1)\}$; $D_{nn+1} \leftarrow [0,0]$;
  Use the classical PERT to compute earliest starting date of each task of G in Ω^-_A;
  $lst^-_{nn+1} \leftarrow est_{nn+1}(\Omega^-_A)$; $l^*_{nn+1} \leftarrow est_{nn+1}(\Omega^-_A)$; $l_{nn+1} \leftarrow 0$;
  foreach (k,l) **such that** $k \leftarrow n-1$ **downto** 1 **do**
    Let (l,v) be a task such that $v \in Succ(l)$;
    $l_{kl} \leftarrow d^+_{kl} + l_{lv}$;
    if $est_{kl}(\Omega^-_A) + l_{kl} > l^*_{lv}$ **then**
      $l^*_{kl} \leftarrow est_{kl}(\Omega^-_A) + l_{kl}$;
      $lst^-_{kl} \leftarrow est_{kl}(\Omega^-_A)$;
    else
      $l^*_{kl} \leftarrow l^*_{lv}$;
      $lst^-_{kl} \leftarrow l^*_{kl} - l_{kl}$;
    foreach (l,u) **such that** $u \in Succ(l) \setminus \{v\}$ **do**
      if $est_{kl}(\Omega^-_A) + d^+_{kl} + l_{lu} > l^*_{lu}$ **then**
        $l^*_{kl} \leftarrow est_{kl}(\Omega^-_A) + d^+_{kl} + l_{lu}$;
        $l_{kl} \leftarrow d^+_{kl} + l_{lu}$;
        $lst^-_{kl} \leftarrow est_{kl}(\Omega^-_A)$;
      else
        if $l^*_{lu} - d^+_{kl} - l_{lu} < lst^-_{kl}$ **then**
          $l^*_{kl} \leftarrow l^*_{lu}$;
          $l_{kl} \leftarrow d^+_{kl} + l_{lu}$;
          $lst^-_{kl} \leftarrow l^*_{kl} - l_{kl}$;
end

4 Constructive approach

Zieliński [47] gave polynomial algorithms for computing optimal upper and lower bounds on latest starting times. We do the same for the optimal upper bound of floats. The basic structure is the same for the three algorithms. The first step instantiates the duration times of some tasks in a network without changing the result of the quantity of concern for a distinguished task. The next step instantiates the duration times of the remaining tasks in the network, while preserving the global possible or necessary criticality of the task. Once duration times of all tasks are precisely set, we need to compute the minimal duration Δ that, added to the duration time of the distinguished task, makes it possibly or necessarily critical in this particular configuration. Duration Δ will easily provide the concerned quantity.

4.1 Evaluating criticality

This section presents methods which can decide in polynomial time if a given task (k,l) is necessarily or possibly critical. First, under the assumption that the duration times of the predecessors of task (k,l) are precisely known, we recall algorithms, proposed in [47], that decide if (k,l) is necessarily or possibly critical. They constitute the basis for computing minimal and the maximal latest starting dates in polynomial time in [47]. We extend some of these results and give a general

algorithm that asserts if (k,l) is necessarily critical in a network G in polynomial time without any consideration of the duration times of tasks preceding (k,l). This result will lead in Section 4.3 to a polynomial algorithm which computes the maximal float of a task.

Let us recall characteristic conditions of the non-necessary criticality of tasks.

Lemma 1 ([11]). *A task $(k,l) \in A$ is not necessarily critical in G if and only if there exists a path $p \in P$ such that $(k,l) \notin p$, p is critical in configuration Ω_p^+ and no critical path in Ω_p^+ includes (k,l).*

Observation 1 ([47]). *A task $(k,l) \in A$ is not necessarily critical in G if and only if (k,l) is not critical in an extreme configuration in which the duration of (k,l) is at its lower bound and all tasks from set $A \setminus (SUCC(k,l) \cup PRED(k,l) \cup \{(k,l)\})$ have duration times set to their upper bounds.*

Now under the assumption that tasks preceding (k,l) have precise duration times, we can set the ones of tasks succeeding (k,l) at precise values while maintaining the status of (k,l) in terms of necessary criticality. It yields a configuration where (k,l) is critical if and only if it is necessarily critical in the interval-valued network. These duration times are given by Propositions 4 and 5.

Proposition 4. *Let $(k,l) \in A$ be a distinguished task, and (i,j) be a task such that $(i,j) \in SUCC(k,l)$. Assume that every task $(u,v) \in PRED(i,j)$ has precise duration. If (k,l) is critical in $G(1,i)$, then the following conditions are equivalent:*

(i) (k,l) is necessarily critical in G,

(ii) (k,l) is necessarily critical in $G(d_{ij} = d_{ij}^-)$.

Proof. $(i) \Longrightarrow (ii)$ Obvious.
$(i) \Longleftarrow (ii)$ We use a proof by contraposition. We need to prove that if (k,l) is critical in $G(1,i)$ and (k,l) is not necessarily critical in G, then (k,l) is not necessarily critical in $G(d_{ij} = d_{ij}^-)$. By assumption, (k,l) is not necessarily critical in G. From Lemma 1, it follows that there exists a path $p \in P$ such that $(k,l) \notin p$, p is critical in configuration Ω_p^+ and no critical path in Ω_p^+ includes (k,l). Since (k,l) is critical in $G(1,i)$, $(i,j) \notin p$ in Ω_p^+. Observe that $d_{ij}(\Omega_p^+) = d_{ij}^-$. From this and the fact that (k,l) is not critical in Ω_p^+, we conclude that (k,l) is not necessarily critical in $G(d_{ij} = d_{ij}^-)$. □

Proposition 5. *Let $(k,l) \in A$ be a distinguished task, and (i,j) be a task such that $(i,j) \in SUCC(k,l)$. Assume that every task $(u,v) \in PRED(i,j)$ has precise duration. If (k,l) is not critical in $G(1,i)$, then the following conditions are equivalent:*

(i) (k,l) is necessarily critical in G,

(ii) (k,l) is necessarily critical in $G(d_{ij} = d_{ij}^+)$.

Proof. A proof follows the definition of necessary criticality. □

Propositions 4 and 5, together with Observation 1, lead us to Algorithm 3 for asserting the necessary criticality of a given task (k,l) in a network in which all tasks that precede (k,l) have precise durations. The algorithm works as follows: first all duration times of tasks neither succeeding nor preceding (k,l) are set to their upper bounds and the earliest starting times of events not succeeding event l are computed by the classical PERT. Then the algorithm tests if (k,l) is critical in the subnetwork $G(1,l)$. We denote by $f_{kl}(u,v)$ the float of (k,l) computed in the network $G(u,v)$. Precise duration times are fixed for tasks immediately succeeding node l. This step is repeated recursively for network $G(1,l+1),\ldots,G(1,n-1)$. At the end of this process, all duration times are precisely set, and (k,l) is necessarily critical in the network with the interval durations if and only if (k,l) is critical in the network with the fixed durations. Computing $f_{kl}(1,i)$ and testing if (k,l) is critical in $G(1,i)$ can be done in constant time, since we already know est_j for all $j \in Pred(i)$, and so Algorithm 3 runs in $O(m)$.

Lemma 1, Propositions 4, 5 and Observation 1 have counterparts for asserting possible criticality. It suffices to swap duration times d_{uv}^- and d_{uv}^+. This leads to an $O(m)$ algorithm (see

Algorithm 3: Asserting whether a task is necessarily critical when duration times of predecessors are precisely known

Input: A network G, task (k,l), interval duration times D_{uv}, $(u,v) \in A$ and for every task in $PRED(k,l)$ the duration is precisely given.

Output: If $f_{kl}(1,n) = 0$, (k,l) is necessarily critical in G; and if $f_{kl}(1,n) > 0$, (k,l) is not necessarily critical in G.

/* Initialization */
foreach $(u,v) \in A \setminus (SUCC(k,l) \cup PRED(k,l) \cup \{(k,l)\})$ **do** $d_{uv} \leftarrow d^+_{uv}$;
$d_{kl} \leftarrow d^-_{kl}$; /* the partially instantiated configuration */
/* Computation */
Compute est_i of events $i \in V \setminus (SUCC(l) \cup \{l\})$ by the classical PERT in the partial configuration;
for $i \leftarrow l$ **to** $n-1$ **such that** $i \in SUCC(l) \cup \{l\}$ **do**
    Compute $f_{kl}(1,i)$;
    if $f_{kl}(1,i) = 0$ /* (k,l) *critical in* $G(1,i)$ */ **then**
        foreach $j \in Succ(i)$ **do** $d_{ij} \leftarrow d^-_{ij}$
    else
        foreach $j \in Succ(i)$ **do** $d_{ij} \leftarrow d^+_{ij}$
Compute $f_{kl}(1,n)$;
return $f_{kl}(1,n)$;

Algorithm 4: Asserting whether a task is possibly critical when duration times of its predecessors are precisely known

Input: A network G, task (k,l), interval duration times D_{uv}, $(u,v) \in A$ and for every task in $PRED(k,l)$ the duration is precisely given.

Output: If $f_{kl}(1,n) = 0$, (k,l) is possibly critical in G; and if $f_{kl}(1,n) > 0$, (k,l) is not possibly critical in G.

/* Initialization */
foreach $(u,v) \in A \setminus (SUCC(k,l) \cup PRED(k,l) \cup \{(k,l)\})$ **do** $d_{uv} \leftarrow d^-_{uv}$;
$d_{kl} \leftarrow d^+_{kl}$; /* the partially instantiated configuration */
/* Computation */
Compute est_i of events $i \in V \setminus (SUCC(l) \cup \{l\})$ by the classical PERT in the partial configuration;
for $i \leftarrow l$ **to** $n-1$ **such that** $i \in SUCC(l) \cup \{l\}$ **do**
    Compute $f_{kl}(1,i)$;
    if $f_{kl}(1,i) = 0$ /* (k,l) *is critical in* $G(1,i)$ */ **then**
        foreach $j \in Succ(i)$ **do** $d_{ij} \leftarrow d^+_{ij}$
    else
        foreach $j \in Succ(i)$ **do** $d_{ij} \leftarrow d^-_{ij}$
Compute $f_{kl}(1,n)$;
return $f_{kl}(1,n)$

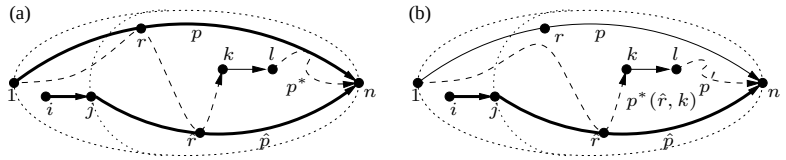


Figure 2: (a) Configuration $\Omega^* - \hat{r}$ on p^* after r (b) Configuration $\Omega^{'}$ (tasks with the maximal duration times are in bold)

Algorithm 4) for asserting the possible criticality of tasks whose predecessors have deterministic durations [47].

We now present an algorithm for evaluating the necessary criticality of a fixed task $(k,l) \in A$ in network G with interval duration times, without any restriction. The key to the algorithm lies in Propositions 6 and 7 that enable a network with interval duration times to be replaced by another network with precise durations for tasks preceding a fixed (k,l), in such a way that (k,l) is necessarily critical in the former if and only if it is necessarily critical in the latter.

Proposition 6. *Let $(k,l) \in A$ be a distinguished task, and (i,j) be a task such that $(i,j) \in PRED(k,l)$. If (k,l) is necessarily critical in $G(j,n)$, then the following conditions are equivalent:*
(i) (k,l) is necessarily critical in G,
(ii) (k,l) is necessarily critical in $G(d_{ij} = d_{ij}^-)$.

Proof. The proof goes in the similar manner to the one of Proposition 4. □

Proposition 7. *Let $(k,l) \in A$ be a distinguished task, and (i,j) be a task such that $(i,j) \in PRED(k,l)$. If (k,l) is not necessarily critical in $G(j,n)$, then the following conditions are equivalent:*
(i) (k,l) is necessarily critical in G,
(ii) (k,l) is necessarily critical in $G(d_{ij} = d_{ij}^+)$.

Proof. $(i) \Longrightarrow (ii)$ Straightforward.
$(i) \Longleftarrow (ii)$ We need to show that if (k,l) is not necessarily critical in $G(j,n)$ and (k,l) is necessarily critical in $G(d_{ij} = d_{ij}^+)$, then (k,l) is necessarily critical in G. To prove this, assume on the contrary that (k,l) is not necessarily critical in G and (k,l) is necessarily critical in $G(d_{ij} = d_{ij}^+)$. From Lemma 1, it follows that there exists a path $p \in P$ such that $(k,l) \notin p$, p is critical in configuration Ω_p^+ and no critical path in Ω_p^+ includes (k,l) or equivalently (k,l) is not critical in Ω_p^+. We will show that for each such configuration, where (k,l) is not critical, the other assumptions lead to construct a critical path that traverses (k,l), which results in a contradiction. By assumption, (k,l) is not necessarily critical in $G(j,n)$. Then there exists a path $\hat{p} \in P(j,n)$ such that $(k,l) \notin \hat{p}$, $\hat{p}$ is critical in configuration $\Omega_{\hat{p}}^+$ and no critical path of $G(j,n)$ traverses (k,l) in $\Omega_{\hat{p}}^+$ (see Lemma 1).

Consider the extreme configuration induced by $p \cup \hat{p} \cup \{(i,j)\}$ and denote it by Ω^* ($\Omega^* = \Omega^+_{p \cup \hat{p} \cup \{(i,j)\}}$). Note that (k,l) is critical in Ω^*, because (k,l) is necessarily critical in $G(d_{ij} = d_{ij}^+)$. Thus there exists a critical path $p^* \in P$ using (k,l) in Ω^*. We define r to be the last common node of $p^*(1,k)$ and p ($r = \max\{v \mid v \in V, v \in p^*(1,k), v \in p\}$) and define $\hat{r}$ to be the last common node of $p^*(1,k)$ and $\hat{p}$ ($\hat{r} = \max\{v \mid v \in V, v \in p^*(1,k), v \in \hat{p}\}$).

We claim that if node $\hat{r}$ exists, then $\hat{r} = r$ or node $\hat{r}$ lies on p^* before node r. Suppose, contrary to our claim, that $\hat{r}$ lies on p^* after r (see Figure 2a). Then subpath $p^*(\hat{r},n)$, $p^*(\hat{r},n) = p^*(\hat{r},k) \cup p^*(k,n)$, is at least as long as subpath $\hat{p}(\hat{r},n)$ in configuration Ω^*. Notice that $p^*(\hat{r},k)$ is one of the longest paths from $\hat{r}$ to k in Ω^*. We may now decrease some task duration times to their lower bounds in configuration Ω^* in the following way (see Figure 2b): for every $(u,v) \in A$, $d_{uv}(\Omega^{'}) = d_{uv}^+$ if $(u,v) \in \hat{p}$ or $(u,v) = (i,j)$; d_{uv}^- otherwise. Duration $d_{ij}(\Omega^{'}) = d_{ij}^+$, and, by assumption, (k,l) is necessarily critical in $G(d_{ij} = d_{ij}^+)$. Consequently (k,l) is critical in this new configuration $\Omega^{'}$. Hence, there exists a critical path $p^{'} \in P$ traversing (k,l). Since node $\hat{r}$ lies on

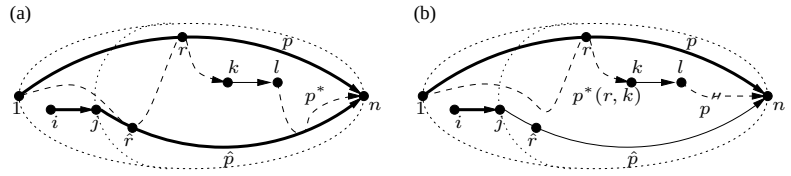


Figure 3: (a) Configuration Ω^* – $\hat{r}$ on p^* before r (b) Configuration $\Omega^{''}$

p^* after node r, $l_{p^*(\hat{r},k)}(\Omega^*) = l_{p^*(\hat{r},k)}(\Omega^{'})$. Therefore path $p^*(\hat{r},k) \cup p^{'}(k,n)$ is at least as long as subpath $\hat{p}(\hat{r},n)$ in configuration $\Omega^{'}$. Decreasing $d_{ij}(\Omega^{'})$ to its lower bound gives configuration $\Omega^+_{\hat{p}}$. Observe that the lengths of paths $p^*(\hat{r},k) \cup p^{'}(k,n)$ and $\hat{p}(\hat{r},n)$ remain unchanged. Hence, there exists a path in $G(j,n)$ composed of two subpaths $\hat{p}(j,\hat{r})$ and $p^*(\hat{r},k) \cup p^{'}(k,n)$ that is at least as long as $\hat{p}$, which is impossible because we have assumed that no critical path goes through (k,l) in $G(j,n)$ in configuration $\Omega^+_{\hat{p}}$.

We can now return to the main proof. Consider configuration Ω^*. The previous claim shows that if node $\hat{r}$ exists, then $\hat{r} = r$ or node $\hat{r}$ lies on p^* before node r (see Figure 3a). In the case that $\hat{r}$ does not exist, the proof proceeds in the same manner. From the above and the criticality of p^* in Ω^*, it follows that subpath $p^*(r,n) = p^*(r,k) \cup p^*(k,n)$, is at least as long as subpath $p(r,n)$ in this configuration. Notice that $p^*(r,k)$ is one of longest paths from r to k in Ω^*. Decreasing some of duration times in Ω^* to their lower bounds, we obtain configuration $\Omega^{''}$ in the following form (see Figure 3b): for every $(u,v) \in A$, $d_{uv}(\Omega^{''}) = d^+_{uv}$ if $(u,v) \in p$ or $(u,v) = (i,j)$; $d_{uv}(\Omega^{''}) = d^-_{uv}$ otherwise. Duration $d_{ij}(\Omega^{''}) = d^+_{ij}$, by assumption, (k,l) is necessarily critical in $G(d_{ij} = d^+_{ij})$, which implies the criticality of (k,l) in this new configuration $\Omega^{''}$. Hence, there exists a critical path $p^{''} \in P$ using (k,l). By the claim, $l_{p^*(r,k)}(\Omega^*) = l_{p^*(r,k)}(\Omega^{''})$. It follows that path $p^*(r,k) \cup p^{''}(k,n)$ is at least as long as subpath $p(r,n)$ in configuration $\Omega^{''}$. If $(i,j) \notin p$, we may decrease duration $d_{ij}(\Omega^{''})$ to its lower bound. Again by the claim, the lengths of subpaths $p^*(r,k) \cup p^{''}(k,n)$ and $p(r,n)$ remain unchanged in this new configuration and so $p^*(r,k) \cup p^{''}(k,n)$ is still at least as long as $p(r,n)$. It is easily seen that this new configuration is equal to Ω^+_p. If $(i,j) \in p$, configurations $\Omega^{''}_p$ and Ω^+_p are equal. Consequently, path $p(1,r) \cup p^*(r,k) \cup p^{''}(k,n)$ is at least as long as p and moreover $p(1,r) \cup p^*(r,k) \cup p^{''}(k,n)$ uses (k,l). This contradicts our assumption that no critical path in Ω^+_p includes (k,l). □

We are now in a position to give an algorithm (Algorithm 5) for asserting the necessary criticality of a prescribed task in a general acyclic network. At each step of the algorithm, tasks between j and k have precise duration times (so Algorithm 3 can be invoked), and Algorithm 5 assigns precise duration times to tasks preceding j, while preserving the criticality of task (k,l). Since Algorithm 3 runs in $O(m)$, Algorithm 5 requires $O(mn)$ time.

Unfortunately, Propositions 6 and 7 can not be adapted to the study of possible criticality, and asserting if a task is possibly critical, in general, is strongly NP-complete [6]. So, while these results are instrumental for computing maximal floats, the same approach cannot be applied to compute minimal floats.

4.2 Computing tightest bounds on latest starting dates

Let us present a polynomial algorithm, proposed in [47], that computes maximal latest starting dates. Let us recall the following simple but important result that allows us to reduce the set of configurations $\mathscr{C}$.

Proposition 8 ([12]). *The minimal upper bound on latest starting dates lst^+_{kl} of task (k,l) in G is attained on an extreme configuration in which the duration of (k,l) is at its lower bound and all tasks that do not belong to set $SUCC(k,l)$ have duration times at their upper bounds.*

Algorithm 5: Asserting necessary criticality of task (k, l)

Input: A network $G =< V, A >$, activity (k, l), interval duration times D_{uv}, $(u, v) \in A$.
Output: If $f_{kl}(1, n) = 0$, (k, l) is necessarily critical in G; and if $f_{kl}(1, n) > 0$, (k, l) is not necessarily critical in G.

```
/* Initialization */
foreach (u, v) ∈ A \ (SUCC(k, l) ∪ PRED(k, l) ∪ {(k, l)}) do d_uv ← d⁺_uv;
d_kl ← d⁻_kl;
/* Computation */
for j ← k downto 2 such that j ∈ PRED(k) ∪ {k} do
    f_kl(j, n) ← Algorithm 3 with G(j, n) and without the initialization phase;
    if f_kl(j, n) = 0 then
        foreach i ∈ Pred(j) do d_ij ← d⁻_ij
    else
        foreach i ∈ Pred(j) do d_ij ← d⁺_ij
f_kl(j, n) ← Algorithm 3 with G(1, n) and without the initialization phase;
return f_kl(j, n);
```

The main idea of the algorithm for determining lst^+_{kl} of a given task $(k, l) \in A$ is based on Lemma 2. It consists in determining the minimal nonnegative real number f^*_{kl} that, added to the lower bound of the duration interval of a specified task (k, l), makes it necessarily critical.

Lemma 2 ([47]). *Let f^*_{kl} be the least nonnegative real number such that (k, l) is necessarily critical with a duration $d^-_{kl} + f^*_{kl}$. Then $lst^+_{kl} = est^+_{kl} + f^*_{kl}$.*

Thus, Proposition 8 and Lemma 2 suggest an algorithm for determining maximal latest starting dates (Algorithm 6). The algorithm proceeds as follows. First the duration times of tasks not succeeding (k, l) are set to their maximal values, and the duration of (k, l) is set to d^-_{kl} (see Proposition 8). From this point on, it computes maximal earliest starting dates of (k, l). Then it tests if (k, l) is necessarily critical in the partially instantiated configuration by calling Algorithm 3. During this call, the algorithm retains all values $f_{kl}(1, i)$ determined by Algorithm 3. If (k, l) is not necessarily critical, the smallest $f_{kl}(1, i)$, previously computed, is the least duration δ to be added to d_{kl}. Then, the algorithm adds this δ to d_{kl} and repeats the previous steps. Since Algorithm 6 makes at most n calls to Algorithm 3, its complexity is $O(nm)$.

Algorithm 6: Computation of maximal latest starting dates of a given task

Input: A network G, task (k, l), interval duration times D_{uv}, $(u, v) \in A$.
Output: The maximal latest starting date lst^+_{kl}.

```
/* Initialization */
foreach (u, v) ∈ A \ (SUCC(k, l) ∪ {(k, l)}) do d_uv ← d⁺_uv;
d_kl ← d⁻_kl;
Compute est⁺_kl by the classical PERT;
Δ ← 0;
/* Computation */
f_kl(1, n) ← Algorithm 3 without the initialization;
/* we can retain all the values f_kl(1, i) computed by Algorithm 3 */
while f_kl(1, n) > 0 do
    δ ← min_{i∈SUCC(l)}{f_kl(1, i) | f_kl(1, i) ≠ 0};
    Δ ← Δ + δ;
    d_kl ← d⁻_kl + Δ;
    f_kl(1, n) ← Algorithm 3 without the initialization;
    /* we can retain all the values f_kl(1, i) computed by Algorithm 3 */
return est⁺_kl + Δ
```

There exist counterparts to Proposition 8, Lemma 2 and Algorithm 6 for minimal latest starting

dates. They are exactly symmetrical. Hence, the sketch of the algorithm for computing minimal latest starting dates (Algorithm 7), proposed in [47], is the same as Algorithm 6 and it also runs in $O(nm)$ time.

Algorithm 7: Computation of the minimal latest starting date of a given task

Input: A network G, activity (k,l), interval duration times D_{uv}, $(u,v) \in A$.
Output: Minimal latest starting date lst^-_{kl}.
/* Initialization */
foreach $(u,v) \in A \setminus (SUCC(k,l) \cup \{(k,l)\})$ **do** $d_{uv} \leftarrow d^-_{uv}$;
$d_{kl} \leftarrow d^+_{kl}$;
Compute est^-_{kl} by the classical PERT;
$\Delta \leftarrow 0$;
/* Computation */
$f_{kl}(1,n) \leftarrow$ Algorithm 4;
/* we can retain all the values $f_{kl}(1,i)$ computed by Algorithm 4 */
while $f_{kl}(1,n) > 0$ **do**
  $\delta \leftarrow \min_{i \in SUCC(l)}\{f_{kl}(1,i) | f_{kl}(1,i) \neq 0\}$;
  $\Delta \leftarrow \Delta + \delta$;
  $d_{kl} \leftarrow d^+_{kl} + \Delta$;
  $f_{kl}(1,n) \leftarrow$ Algorithm 4;
  /* we can retain all the values $f_{kl}(1,i)$ computed by Algorithm 4 */
return $est^-_{kl} + \Delta$

4.3 Computing maximal floats

In order to compute the maximal float of task $(k,l) \in A$, we first set the duration times of the tasks neither preceding nor succeeding (k,l) according to the following Lemma 3. The maximal float of (k,l) after this partial instantiation is the same as in the original network G.

Lemma 3 ([12]). *The maximal float f^+_{kl} of task (k,l) in G is attained on an extreme configuration in which the duration of (k,l) is at its lower bound and all tasks from set $A \setminus (SUCC(k,l) \cup PRED(k,l) \cup \{(k,l)\})$ have duration times at their upper bounds.*

Then we increase step by step the duration of (k,l) from $d_{kl} = d^-_{kl}$ until (k,l) becomes necessarily critical. Testing can be done in $O(mn)$ time (see Algorithm 5). Lemmas 4 and 6 give the hint to find the increment of d_{kl} at each step of the algorithm. According to Proposition 9, this incremental technique eventually yields f^+_{kl}.

Lemma 4. *Let tasks $(i,j) \in PRED(k,l)$ have precise durations in G. Then (k,l) is necessarily critical in G if and only if there exists a path $p \in P(1,k)$ such that for every node $j \in p$, (k,l) is necessarily critical in $G(j,n)$.*

Proof. ($\Longrightarrow$) Let us denote by p a longest path from 1 to k. Note that tasks $(i,j) \in PRED(k,l)$ have precise durations. From the necessary criticality of (k,l) in G, it follows that path p is part of a longest path from 1 to n and this path uses (k,l) for each configuration. Thus for every node $j \in p$, the subpath $p(j,n)$ is critical path in $G(j,n)$. Since this holds true for each configuration, (k,l) is necessarily critical in $G(j,n)$.
($\Longleftarrow$) Just take $j = 1$. $G(1,n) = G$ and so (k,l) is necessarily critical in G. □

There is a link between the maximal latest starting date lst^+_{kl} and the maximal float value f^+_{kl} of task (k,l) under the assumption that tasks $(i,j) \in PRED(k,l)$ have precise duration times.

Lemma 5 ([47]). *Let tasks $(i,j) \in PRED(k,l)$ have precise durations in G. Then $f^+_{kl} = lst^+_{kl} - est^+_{kl}$.*

Thus, the maximal float value f^+_{kl} can be computed by means of the algorithms for determining lst^+_{kl} presented previously.

Lemma 6. *Let $\Delta = \min_j\{f^+_{kl}(j,n) \mid (k,l)$ is not necessarily critical in $G(j,n)\}$ where $f^+_{kl}(j,n)$ is the maximal float of (k,l) in $G(j,n)$, $j \in PRED(k)$. Then for all $\epsilon < \Delta$, task (k,l) is not necessarily critical in $G(j,n)$, $j \in PRED(k)$, with duration $d_{kl} = d^-_{kl} + \epsilon$. Moreover there exists $j^* \in PRED(k)$ such that (k,l) is not necessarily critical in $G(j^*,n)$ with duration $d_{kl} = d^-_{kl}$, and (k,l) becomes necessarily critical in $G(j^*,n)$ with duration $d_{kl} = d^-_{kl} + \Delta$.*

Proof. Consider a node j such that (k,l) is not necessarily critical in $G(j,n)$. By Observation 1, one can assume that task (k,l) has duration time of the form $d_{kl} = d^-_{kl}$. Let Ω be an extreme configuration where the float of (k,l) attains its maximal value $f^+_{kl}(j,n) > 0$ in $G(j,n)$, and $\epsilon < \Delta \leq f^+_{kl}(j,n)$ $(d_{kl}(\Omega) = d^-_{kl})$. p' denotes a longest path in $G(j,n)$ in Ω, while p'' stands for a longest path in $G(j,n)$ in Ω using (k,l). Therefore, $f^+_{kl}(j,n) = l_{p'}(\Omega) - l_{p''}(\Omega) > \epsilon$. Let us define the configuration Ω' such that $d_{uv}(\Omega') = d^-_{kl} + \epsilon$ if $(u,v) = (k,l)$; $d_{uv}(\Omega') = d_{uv}(\Omega)$ otherwise. Path p' remains a longest path in Ω' with same length, and p'' is still a longest path traversing (k,l) with length $l_{p''}(\Omega') = l_{p''}(\Omega) + \epsilon$. Since the float of (k,l) in Ω' is $l_{p'}(\Omega') - l_{p''}(\Omega') > 0$, (k,l) is not necessarily critical in $G(j,n)$, $j \in PRED(k)$, with duration $d_{kl} = d^-_{kl} + \epsilon$.

Consider a node j^* such that (k,l) is not necessarily critical in $G(j^*,n)$, and set $\Delta = f^+_{kl}(j^*,n)$. Then for all configurations, the difference between a longest path in $G(j^*,n)$ and a longest path using (k,l) is less or equal than Δ. If we increase the duration of (k,l) to $d^-_{kl} + \Delta$, a longest path, in this new configuration, will traverse (k,l) and thus (k,l) will be necessarily critical in $G(j^*,n)$ with duration $d_{kl} = d^-_{kl} + \Delta$. □

Proposition 9. *Let f^*_{kl} be the least nonnegative real number such that (k,l) is necessarily critical in $G(d_{kl} = d^-_{kl} + f^*_{kl})$. Then $f^+_{kl} = f^*_{kl}$.*

Proof. Consider any configuration Ω. Let $p^{'}$ be a longest path in Ω and p'' be a longest path including (k,l) in Ω. Note that $f_{kl}(\Omega) = l_{p'}(\Omega) - l_{p''}(\Omega)$. Let us now modify configuration Ω and denote it by Ω^x. Configuration Ω^x is defined as follows: for every $(u,v) \in A$ $d_{uv}(\Omega^x) = d_{uv}(\Omega) + x$ if $(u,v) = (k,l)$; $d_{uv}(\Omega^x) = d_{uv}(\Omega)$ otherwise, where x is a nonnegative real number. It is clear that $l_{p''}(\Omega^x) = l_{p''}(\Omega) + x$ and $p^{'}$ remains a longest path including (k,l) in new configuration Ω^x. Consider the following two cases: $(x < f_{kl}(\Omega))$. Then $l_{p''}(\Omega^x) \leq l_{p'}(\Omega)$, and so $p^{'}$ is still a critical path in Ω^x and has the same length as in Ω. This gives $f_{kl}(\Omega^x) = f_{kl}(\Omega) - x$. $(x \geq f_{kl}(\Omega))$. Then $p^{''}$ becomes a critical path and so $f_{kl}(\Omega^x) = 0$. Thus, for all Ω and x, equation $f_{kl}(\Omega^x) = \max(f_{kl}(\Omega) - x, 0)$ holds. In particular, for $x = f^*_{kl}$ and configuration $\Omega = \Omega^*$ such that Ω^* maximizes the float of task (k,l) in G. From the definition of f^*_{kl} we get $f_{kl}(\Omega^{f^*_{kl}}) = 0$, hence $f_{kl}(\Omega^*) - f^*_{kl} \leq 0$, and finally $f^+_{kl} \leq f^*_{kl}$.

Suppose that $f^+_{kl} < f^*_{kl}$. Set $y = (f^*_{kl} + f^+_{kl})/2$. Then, for every Ω, $f_{kl}(\Omega^y) = \max(f_{kl}(\Omega) - y, 0) = 0$. Note that y is a nonnegative real number, smaller than f^*_{kl}, such that (k,l) is necessarily critical in $G(d_{kl} = d^-_{kl} + y)$, which contradicts the definition of f^*_{kl}. Hence, we conclude that $f^+_{kl} = f^*_{kl}$. □

From the above results, it follows an algorithm that computes the maximal float of task (k,l) in polynomial time (Algorithm 8). The algorithm first tests the necessary criticality of (k,l) in G by calling Algorithm 5. If so, then its float is null. Otherwise, it determines the minimal duration δ that must be added to the duration of (k,l) to change the instantiation made by Algorithm 5 for at least one task preceding (k,l). Moreover, the instantiation made by Algorithm 5 will change if and only if (k,l) becomes necessarily critical for at least one new subnetwork $G(i,n)$. This δ is the least positive value of the maximal float of (k,l) in networks $G(i,n)$, where $i \in PRED(k)$. Since the duration times of predecessors were previously fixed, these values of the maxima of floats are differences between the maxima of latest starting dates and the minima of earliest starting dates (see Lemma 5). Algorithm 8 executes at most n times the `while` loop. At each iteration of `while` loop, (k,l) becomes necessarily critical in at least one new subnetwork $G(j,n)$. Thus the loop is executed at most n times, and thus Algorithm 8 requires $O(n^3m)$ time.

Algorithm 8: Computation of the maximal float of a given task

Input: A network G, task (k,l), interval duration times D_{uv}, $(u,v) \in A$.
Output: The maximal float f_{kl}^{+}.
$\Delta \leftarrow 0$;
$f_{kl}(1,n) \leftarrow$ Algorithm 5;
while $f_{kl}(1,n) > 0$ **do**
  $\delta \leftarrow \infty$;
  foreach $i \in PRED(k)$ **do**
    $[est_{kl}^{*}, lst_{kl}^{*}] \leftarrow$ Algorithm 6 in $G(i,n)$;/*Algorithm 6 returns additionally est_{kl}^{+}*/
    $f_{kl}^{*} \leftarrow lst_{kl}^{*} - est_{kl}^{*}$;
    if $f_{kl}^{*} > 0$ **then** $\delta \leftarrow \min\{\delta, f_{kl}^{*}\}$;
  $\Delta \leftarrow \Delta + \delta$;
  $d_{kl} \leftarrow d_{kl}^{-} + \Delta$;
  $f_{kl}(1,n) \leftarrow$ Algorithm 5 without the initialization;
return Δ

5 Illustrative example

In order to illustrate the presented algorithms, let us consider the execution of the algorithms on the project network given in Figure 4.

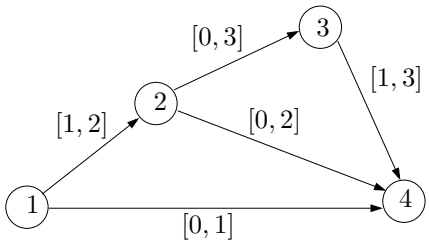
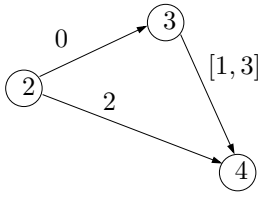


Figure 4: A project network

We wish to compute the maximal float of task $(2,3)$ by means of Algorithm 8. In this section, Ω_{par} represents a partial configuration, that is, the precise duration times of instantiated tasks and the interval duration times of the non-instantiated tasks. In our example, the knowledge about duration times is represented by the partial configuration Ω_{par} (initially the duration times are ill-known): $D_{12} = [1,2]$, $D_{23} = [0,3]$, $D_{34} = [1,3]$, $D_{24} = [0,2]$, $D_{14} = [0,1]$. We now analyze a complete execution of Algorithm 8 in order to compute the maximal float of task $(2,3)$.

Algorithm 8 After initializing Δ to 0, Algorithm 8 calls Algorithm 5.

Algorithm 5 The initialization of Algorithm 5 assigns precise duration times to tasks $(2,3)$, $(2,4)$ and $(1,4)$ obtaining the partial configuration Ω_{par}: $D_{12} = [1,2]$, $D_{23} = 0$, $D_{34} = [1,3]$, $D_{24} = 2$, $D_{14} = 1$. Then the `for` loop is executed. At the first iteration, Algorithm 3 is called for the subnetwork $G(2,4)$ without the initialization. It means that Algorithm 3 only considers the network composed of tasks: $(2,3)$ with duration 0, $(3,4)$ with duration $[1,3]$ and task $(2,4)$ with duration 2 (see the next network).



Algorithm 3 The classical PERT is first performed for determining the earliest starting time of event 2, which is equal to null. Then the `for` loop of Algorithm 3 is executed and the float of $(2,3)$ is computed in the network constituted only by this task $(2,3)$. Its float is obviously null and then Algorithm 3 assigns the minimal duration to task $(3,4)$. The partial configuration Ω_{par} is then $D_{12} = [1,2]$, $D_{23} = 0$, $D_{34} = 1$, $D_{24} = 2$, $D_{14} = 1$.
At the second iteration of the `for` loop of Algorithm 3, the float of $(2,3)$ is computed in the subnetwork $G(2,4)$ with durations: $D_{23} = 0$, $D_{34} = 1$, $D_{24} = 2$. The float is 1 in this subnetwork. Thus, Algorithm 3 returns 1 to Algorithm 5.

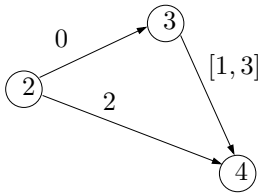
Task $(2,4)$ is not critical in $G(2,4)$, in consequence Algorithm 5 sets the duration times of the tasks preceding node 2 to their upper bounds. This leads to the partial configuration Ω_{par}: $D_{12} = 2$, $D_{23} = 0$, $D_{34} = [1,3]$, $D_{24} = 2$, $D_{14} = 1$. The `for` loop terminates. Before the end of Algorithm 5, Algorithm 3 is invoked for $G(1,4)$. This call is the same as an execution of the `for` loop with $j = 1$.

Algorithm 3 The classical PERT is first called for determining the earliest starting times of events 1 and 2, which are equal to 0 and 2, respectively. Then the `for` loop of Algorithm 3 is executed and the float of task $(2,3)$ is computed in $G(1,3)$ (the network is constituted by only tasks $(1,2)$ and $(2,3)$). Its float is obviously null and then Algorithm 3 assigns the minimal duration to task $(3,4)$. The partial configuration Ω_{par} is then $D_{12} = 2$, $D_{23} = 0$, $D_{34} = 1$, $D_{24} = 2$, $D_{14} = 1$.
At the second iteration of the `for` loop of Algorithm 3, the float of $(2,3)$ is computed in the subnetwork $G(1,4)$ with the partial configuration. The float of $(2,3)$ is 1 in this subnetwork and Algorithm 3 returns 1 to Algorithm 5.

Algorithm 5 returns 1 to Algorithm 8.

$(2,3)$ is not necessarily critical in G so Algorithm 8 runs the `while` loop. In this loop, maximal latest starting dates are computed in all subnetworks $G(i,4)$, $i \in PRED(2)$ (`for` loop). Algorithm 6 is first called in $G(2,4)$.

Algorithm 6 lst^{+}_{23} is determined in $G(2,4)$ as follows : the duration times of tasks not succeeding $(2,3)$ are set to their upper bounds and duration of $(2,3)$ is set to its lower bound. Thus, the partial configuration Ω_{par} is $D_{23} = 0$, $D_{34} = [1,3]$, $D_{24} = 2$ (see the network below).



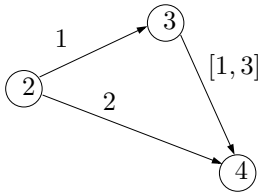
In the considered network, the maximal earliest starting date of $(2,3)$ is 0. Then, Algorithm 6 calls Algorithm 3 without the initialization.

Algorithm 3 f_{23} is null in $G(2,3)$, since there is only task $(2,3)$ in the network (this float is retained by Algorithm 6). Then the duration of task $(3,4)$ is set to $d^{-}_{23} = 1$, and finally f_{23} in $G(2,4)$ is computed in configuration Ω_{par}: $D_{23} = 0$, $D_{34} = 1$, $D_{24} = 2$. This float is equal to 1 and is returned to Algorithm 6.

Table 1: The intervals of values of earliest and latest starting times and of floats and criticality evaluations for all tasks of the network given in Figure 4

task (i,j)	EST_{ij}	LST_{ij}	F_{ij}	possibly critical.	necessarily critical
$(1,2)$	$[0,0]$	$[0,0]$	$[0,0]$	yes	yes
$(1,4)$	$[0,0]$	$[1,8]$	$[1,8]$	no	no
$(2,3)$	$[1,2]$	$[1,3]$	$[0,1]$	yes	no
$(2,4)$	$[1,2]$	$[1,8]$	$[0,6]$	yes	no
$(3,4)$	$[1,5]$	$[1,6]$	$[0,1]$	yes	no

The minimal float computed by the previous algorithm is $\Delta = 1$, and thus the duration of $(2,3)$ is increased by 1. Then Algorithm 3 is called one more time in $G(2,4)$, with duration times Ω_{par}: $D_{23} = 1$, $D_{34} = [1,3]$, $D_{24} = 2$ (see the network below).



Algorithm 3 During the execution of the algorithm, the partial configuration Ω_{par}: $D_{23} = 1$, $D_{34} = 1$, $D_{24} = 2$ is constructed. This allows to deduce that $(2,3)$ is necessarily critical in the network $G(2,4)$ (with the possible duration times set before the call of the algorithm).

$(2,3)$ is necessarily critical in $G(2,4)$, for a duration of $(2,4)$ of $d_{23} = d_{23}^- + 1$, so the latest starting date of $(2,3)$ is 1 in $G(2,4)$.

In the configuration constructed by the previous Algorithm 6, the earliest starting date of $(2,3)$ is null. Thus, the value f_{kl}^* is equal to 1, and δ is set to 1.
In the second iteration of the `for` loop, Algorithm 8 calls Algorithm 6 in $G(1,4)$ to compute the upper bound on the latest starting dates of $(2,3)$ when its duration is $d_{23} = 0$.

Algorithm 6 After the initialization, the partial configuration Ω_{par} is $D_{12} = 2$, $D_{23} = 0$, $D_{34} = [1,3]$, $D_{24} = 2$, $D_{14} = 1$. The Algorithm is executed as previously and returns 3 for the latest starting date of $(2,3)$ in $G(1,4)$.

In the constructed configuration, the earliest starting date of $(2,3)$ is 2, $f_{kl}^* = 1$ and δ still equals 1. Now Algorithm 8 sets the duration of $(2,3)$ to $d_{23}^- + \delta$. Algorithm 5 is called with the possible durations Ω_{par}: $D_{12} = [1,2]$, $D_{23} = 1$, $D_{34} = [1,3]$, $D_{24} = [0,2]$, $D_{14} = [0,1]$.

Algorithm 5 The algorithm works as previously. This time task $(2,3)$ is necessarily critical and Algorithm 5 returns 0.

Task $(2,3)$ is necessarily critical in the constructed network, so the `while` loop terminates. Algorithm 8 returns 1. This value is the total duration added to the lower bound of $(2,3)$ to make it necessarily critical. Therefore, the maximal float of $(2,3)$ is 1. Task $(2,3)$ is not necessarily critical.
Table 1 reports all the computational results on the intervals of possible values of earliest and latest starting times and floats and on criticality evaluations for all tasks of the project network given in Figure 4.

Table 2: The complexity of the interval problems and the running times of the algorithms

Earliest starting date (all tasks)	MinEST	P	$O(n+m)$	[5]
	MaxEST	P	$O(n+m)$	[5]
Latest starting date (one task)	MinLST	P	$O(mn)$	Algorithm 6 [47]
	MaxLST	P	$O(mn)$	Algorithm 7 [47]
Latest starting date (all tasks)	MinLST	P	$O(m^2)$	Algorithm 2
Float (all tasks)	MinF	NP-hard	$O((m+n)\|P\|)$	Algorithm 1 [11]
Float (one task)	MaxF	P	$O(n^3m)$	Algorithm 8

Table 3: The running times of the algorithms for computing quantities for all tasks in a network

Algorithm 1 (Path Algorithm MaxLST, MinF, MaxF)	$O((m+n)\|P\|)$
Algorithm 2 (Polynomial Path Algorithm MinLST)	$O(m^2)$
Algorithm 6 (MaxLST)	$O(m^2n)$
Algorithm 7 (MinLST)	$O(m^2n)$
Algorithm 8 (MaxF)	$O(n^3m^2)$

6 Complexity and experimental results

Table 2 summarizes the complexity of the different problems of the PERT on intervals. Moreover, Table 2 gives the running times of the best known algorithms that compute the quantities of interest. In particular, the path algorithm that computes minimal and maximal floats and maximal latest starting dates requires $O((n+m)|P|)$ time, where $|P|$ is the number of paths of a network. This running time depends on the topology of a network. Note that, some algorithms need only one execution to compute a quantity (for example, minimal starting dates) for all tasks of a network. Other ones need to be executed for each task. This means that the comparison of their running times can be difficult. Table 3 gives the running time of each of the proposed algorithm for computing the quantity of interest for all tasks in a network and recalls the quantities computed by the algorithms. MaxLST, MinLST, MaxF and MinF denote maximal latest starting dates, minimal latest starting dates, maximal floats and minimal floats, respectively.

We now present some computational results in order to evaluate the performance of all the considered algorithms on a scheduling problem library of 600 networks having 120 tasks. Those instances of problems can be found in the PSPLIB library. They have been generated by ProGen, the program for activity network generation [33], which can be downloaded from the PSPLIB web site[1]. We have added a range of 20% to task duration times to obtain intervals. Table 4 reports minimal, maximal and average execution times (in milliseconds) of five algorithms tested on those 600 problems. All the algorithms were written in C language and ran on a PC computer equipped with 2.5GHz CPU. As seen from the experimental results, the path algorithm (potentially exponential) for simultaneously computing the maxima of latest starting dates and the minima and maxima of floats is very efficient in practice. This comes from the fact that the number of paths in these networks is not so huge, i.e. between 408 and 6559 different paths. Of course, one can construct more complex networks, but PSPLIB is known to be relevant to realistic scheduling problems. On the other hand, the polynomial algorithm that computes maximal floats is not so efficient for these scheduling problems. These results show that the proposed algorithms are very efficient for problems involving about one hundred tasks.

[1] http://129.187.106.231/psplib/

Table 4: Minimal, maximal and average execution times (in milliseconds) of five algorithms tested on 600 problems from PSPLIB

	Min	Max	Avg.
Algorithm 1 (Path Algorithm MaxLST, MinF, MaxF)	6	138	36
Algorithm 2 (Polynomial Path Algorithm MinLST)	1.4	2.2	1.6
Algorithm 6 (MaxLST)	4	16	8.1
Algorithm 7 (MinLST)	41	9	19
Algorithm 8 (MaxF)	210	2 860	1 090

Table 5: Average execution times (in milliseconds) of the algorithms tested on problems from RANGEN

	Order Strength OS										
	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
Algorithm 1	4	35	111	249	583	1 534	5 161	27 000	303 117		2
Algorithm 2	1	5	6	6	6	6	5	4	3	2	1
Algorithm 6	4	28	57	76	87	89	79	58	30	12	2
Algorithm 7	5	44	96	138	174	195	181	142	80	21	2
Algorithm 8	16	600	2 280	4 926	9 520	16 321	22 100	23 712	16 625	5 562	78

We have made a second batch of experiments in order to evaluate the algorithms on networks with different complexities. For this purpose we have used the RANGEN 1 project generator which can be downloaded from the RANGEN web page[2]. RANGEN 1 is particularly useful because it can generate networks with various order strengths, denoted by OS. This coefficient measures the hardness of a problem instance. The order strength is the number of precedence relations divided by the theoretical maximal number of precedence relations in a network. More details on RANGEN can be found in [9]. We have generated by RANGEN nine sets of 100 networks with 100 tasks in which the values of OS are equal to $i \cdot 0.1$, $i = 1, \ldots, 9$, one network with OS equals 0 (all tasks of this network can be executed in parallel) and one network in which OS equals 1 (all tasks are series). Again, we have ignored resource constraints of these problems and added a range of 20% to task duration times to obtain intervals. Table 5 reports execution times of the algorithms on the generated problems by RANGEN. It is easy to observe that the computational time of most of the algorithms depends on values of OS. Figure 5 presents execution times of the algorithms. The computational time of the path algorithm drastically increases when the value of OS increases. This is due to the fact that the number of potential paths in the networks increases when OS is increased (if OS is very close to 1 then networks become parallel ones). The other algorithms are quite efficient. In particular, the polynomial algorithm for computing maximal floats is more efficient than the path algorithm when OS is greater than 0.7. Each polynomial algorithm achieves its maximal execution time on networks with OS parameter between 0.5 and 0.7.

7 Conclusion

This paper provides a complete solution to the criticality analysis of activity networks when the duration times of tasks are modeled by means of intervals. It is shown that moving from precise to imprecise durations radically changes the nature and complexity of the problem, ruining the

[2] http://www.projectmanagement.ugent.be/rangen.php

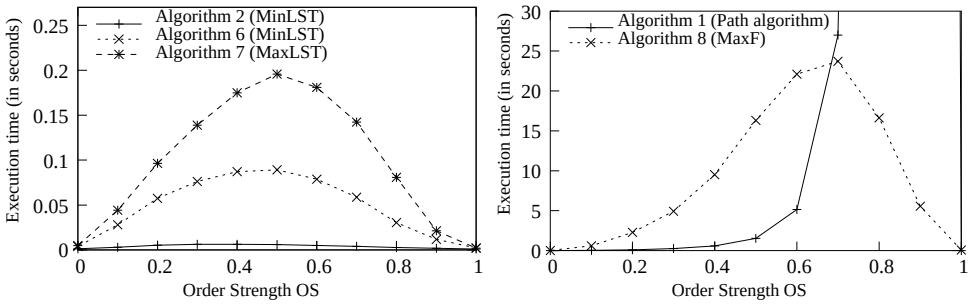


Figure 5: Execution times of of the algorithms on RANGEN benchmark

classical critical path method. The part of the problem pertaining to computing minimal float values, becomes strongly NP-hard [6] and remains NP-hard even if in acyclic planar network of node degree 3 [7]. Moreover, it is not at all approximable, unless P=NP [47]. The other questions remain polynomially solvable, although not straightforwardly so.

An interesting question is how to exploit the results of an interval criticality analysis. The end-user may not be accustomed to obtaining imprecise floats or latest starting times. A task that has a large maximal float is less likely to become critical in practice on a real configuration, than a task that has a small maximal float . On the other hand, all tasks having positive minimal float are ensured not to be critical whatever the actual configuration turns out to be. So any interval PERT software would present results to the user by eliminating tasks with positive minimal floats first, and drawing the user's attention on necessary critical tasks first and then on tasks having small maximal floats. It is clear that, insofar as obtained results are sufficiently precise, the information derived from an interval criticality analysis is richer than the one provided by a standard criticality analysis, because it is supposed to cover many possible scenarios at the same time.

The presented results can be also applied to the criticality analysis of activity networks with imprecise duration times of tasks modeled by fuzzy intervals, whose membership functions are regarded as possibility distributions for the values of the unknown duration times called fuzzy-valued PERT. Fuzzy intervals are more informative than crisp ones as they also account for plausible values of task duration times. They address the dilemma between safe but uniformative intervals and narrow but possibly wrong ones. This gradual extension of interval PERT can be computed via the standard decomposition in α-cuts that consists in decomposing the fuzzy-valued PERT into a family of the interval-valued PERT problems, which can be solved by algorithms presented here (see for details [12, 47]). Another method for addressing fuzzy interval PERT exploits a notion of *gradual number* [21]. Gradual numbers provide a new outlook on fuzzy intervals that can be viewed as classical intervals of gradual numbers. This allows us to directly apply the interval algorithms, proposed in this paper to the fuzzy-valued PERT problems (see [20] for an example of algorithms for computing upper bounds on floats).

It is worth pointing out that the presented results can be useful tools for min-max regret version of the longest (shortest) path problem with interval weights in acyclic graphs (see [2, 27, 28]). In particular, the algorithms for evaluating the necessary criticality and for computing the maximal float of an activity (arc) are useful in a preprocessing of an instance of the problem, since a necessarily critical activity (arc) can be automatically added to a constructed min-max regret path. In consequence, one can decompose an instance with a necessarily critical arc into two separate instances in smaller subgraphs [29]. An activity (arc) with a small maximal float can be considered as a candidate for belonging to a min-max regret path.

References

[1] V. G. Adlakha and V. G. Kulkarni. A classified bibliography of research on stochastic PERT networks: 1966- 1987. *INFOR*, 27:272–296, 1989.

[2] I. Averbakh and V. Lebedev. Interval data minmax regret network optimization problems. *Discrete Applied Mathematics*, 138:289–301, 2004.

[3] J. J. Buckley. Fuzzy PERT. In G. Evans, W. Karwowski, and M. Wilhelm, editors, *Applications of Fuzzy Set Methodologies in Industrial Engineering*, pages 103–114. Elsevier, Amsterdam, Oxford, New York, Tokyo, 1989.

[4] S. Chanas, D. Dubois, and P. Zieliński. On the sure criticality of tasks in activity networks with imprecise durations. *IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics*, 34:393–407, 2002.

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

[6] S. Chanas and P. Zieliński. The computational complexity of the criticality problems in a network with interval activity times. *European Journal of Operational Research*, 136:541–550, 2002.

[7] S. Chanas and P. Zieliński. On the hardness of evaluating criticality of activities in planar network with duration intervals. *Operation Research Letters*, 31:53–59, 2003.

[8] E. Conde. A minmax regret approach to the critical path method with task interval times. *European Journal of Operational Research*, 197:235–242, 2009.

[9] E. Demeulemeester, M. Vanhoucke, and W. Herroelen. A random network generator for activity-on-the-node networks. *Journal of Scheduling*, 6:13–34, 2003.

[10] D. Dubois. Modèles mathématiques de l'imprécis et de l'incertain en vue d'applications aux techniques d'aide à la décision. Thèse d'état de l'Université Scientifique et Médicale de Grenoble et de l'Institut National Politechnique de Grenoble, 1983.

[11] D. Dubois, H. Fargier, and J. Fortin. Computational methods for determining the latest starting times and floats of tasks in interval-valued activity networks. *Journal of Intelligent Manufacturing*, 16:407–422, 2005.

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

[13] D. Dubois and H. Prade. *Fuzzy Sets and Systems: Theory and Applications*. Academic Press, New York, 1980.

[14] D. Dubois, H. Prade, and P. Smets. Representing partial ignorance. *IEEE Trans. on Systems, Man and Cybernetics*, 26:361–377, 1996.

[15] D. Dubois, H. Prade, and P. Smets. A definition of subjective possibility. *International Journal of Approximate Reasonning*, 48:352–364, 2008.

[16] S. E. Elmaghraby. *Activity Networks: Project Planning and Control by Network Models*. Wiley, New York, 1977.

[17] S. E. Elmaghraby. On criticality and sensitivity in activity networks. *European Journal of Operational Research*, 127:220–238, 2000.

[18] H. Fargier, V. Galvagnon, and D. Dubois. Fuzzy PERT in series-parallel graphs. In *9-th IEEE International Conference on Fuzzy Systems*, pages 717–722, San Antonio, TX, 2000.

[19] S. Ferson and L. Ginzburg. Different methods are needed to propagate ignorance and variability. *Reliability Engineering and Systems Safety*, 54:133–144, 1996.

[20] J. Fortin and D. Dubois. Solving Fuzzy PERT using gradual real numbers. In L. Penserini, P. Peppas, and A. Perini, editors, *Starting AI Researcher's Symposium (STAIRS), Riva del Garda, Italy, 28/08/2006-01/09/2006*, volume 142 of *Frontiers in Artificial Intelligence and Applications*, pages 196–207, http://www.iospress.nl/, 2006. IOS Press.

[21] J. Fortin, D. Dubois, and H. Fargier. Gradual numbers and their application to fuzzy interval analysis. *IEEE Transactions on Fuzzy Systems*, 16(2):388–402, 2008.

[22] J. Fortin, P. Zieliński, D. Dubois, and H. Fargier. Interval analysis in scheduling. In P. van Beek, editor, *Principles and Practice of Constraint Programming - CP 2005*, volume 3709 of *Lecture Notes in Computer Science*, pages 226–240. Springer-Verlag, 2005.

[23] I. Gazdik. Fuzzy network planning. *IEEE Transactions on Reliability*, 32:304–313, 1983.

[24] J. N. Hagstrom. Computational complexity of pert problems. *Networks*, 18:139–147, 1988.

[25] M. Hapke, A. Jaszkiewicz, and R. Słowiński. Fuzzy project scheduling system for software development. *Fuzzy Sets and Systems*, 67:101–107, 1994.

[26] M. Hapke and R. Słowiński. Fuzzy priority heuristics for project scheduling. *Fuzzy Sets and Systems*, 86:291–299, 1996.

[27] O. E. Karasan, M. C. Pinar, and H. Yaman. The robust shortest path problem with interval data. *Optimization Online*, 2001.

[28] A. Kasperski and P. Zieliński. The robust shortest path problem in series-parallel multidigraphs with interval data. *Operations Research Letters*, 34:69–76, 2006.

[29] A. Kasperski and P. Zieliński. Minmax regret approach and optimality evaluation in combinatorial optimization problems with interval and fuzzy weights. *European Journal of Operational Research*, 200:680–687, 2010.

[30] A. Kaufmann and M. M. Gupta. *Fuzzy Mathematical Models in Engineering and Management Science*. North Holland, Amsterdam, 1991.

[31] J. E. Kelley. Critical path planning and scheduling - mathematical basis. *Operations Research*, 9:296–320, 1961.

[32] G. B. Kleindorfer. Bounding distributions for a stochastic acyclic network. *Operations Research*, 19:1586–1601, 1971.

[33] R. Kolisch and A. Sprecher. Psplib - a project scheduling library. *European Journal of Operational Research*, 96:205–216, 1996.

[34] P. Kouvelis and G. Yu. *Robust Discrete Optimization and its applications*. Kluwer Academic Publishers, 1997.

[35] F. A. Loostma. *Fuzzy logic for planning and decision-making*. Kluwer Academic Publishers, Dordrecht, 1997.

[36] A. Ludwig, R. H. Möhring, and F. Stork. A Computational Study on Bounding the Makespan Distribution in Stochastic Project Networks. *Annals of Operations Research*, 102:49–64, 2001.

[37] D. G. Malcolm, J. H. Roseboom, C. E. Clark, and W. Fazar. Application of a Technique for Research and Development Program Evaluation. *Operations Research*, 7:646–669, 1959.

[38] C. S. McCahon. Using PERT as an Approximation of Fuzzy Project-Network Analysis. *IEEE Transactions on Engineering Management*, 40:146–153, 1993.

[39] C. S. McCahon and L. S. E. Project network analysis with fuzzy activity times. *Computers and Mathematics with Applications*, 15:829–838, 1988.

[40] I. Meilijson and A. Nadas. Convex majorization with an application to the length of critical paths. *Journal of Applied Probability*, 16:671–677, 1979.

[41] R. E. Moore. *Methods and Applications of Interval Analysis*. SIAM, Philadelphia, 1979.

[42] S. H. Nasution. Fuzzy critical path method. *IEEE Transactions on Systems, Man, and Cybernetics*, 24:48–57, 1994.

[43] H. Prade. Using fuzzy sets theory in a scheduling problem: a case study. *Fuzzy Sets and Systems*, 2:153–165, 1979.

[44] H. Rommelfanger. Network analysis and information flow in fuzzy environment. *Fuzzy Sets and Systems*, 67:119–128, 1994.

[45] A. W. Shogan. Bounding Distributions for a Stochastic PERT Network. *Networks*, 7:359–381, 1977.

[46] J. Valdes, R. E. Tarjan, and E. L. Lawler. The recognition of series parallel digraphs. *SIAM Journal on Computing*, 11:298–313, 1982.

[47] P. Zieliński. On computing the latest starting times and floats of activities in a network with imprecise durations. *Fuzzy Sets and Systems*, 150:53–76, 2005.

[48] P. Zieliński. Efficient Computation of Project Characteristics in a Series-Parallel Activity Network with Interval Durations. In G. Della Riccia, D. Dubois, R. Kruse, and H. J. Lenz, editors, *Decision Theory and Multi-Agent Planning*, number 482 in CISM Courses and Lectures, pages 111–130. Springer-Verlag, Wien, New York, 2006.