Solving Fuzzy PERT Using Gradual Real Numbers

Jérôme FORTIN\textsuperscript{a}, Didier DUBOIS\textsuperscript{a},
\textsuperscript{a}IRIT/UPS 118 route de Narbonne,
31062, Toulouse, cedex 4, France,
e-mail: {fortin, dubois}@irit.fr

Abstract. From a set of a partially ordered tasks, one goal of the project scheduling problem is to compute earliest starting dates, latest starting dates and floats of the different tasks, and to identify critical tasks. When durations of tasks are not precisely known, the problem is much trickier. Recently we provided polynomial algorithms to compute upper bound of activity floats in the interval-valued model. The aim of this paper is to extend those new algorithms to the fuzzy-valued problem. To this end, we use the new notion of gradual numbers \cite{7}, that represents soft boundaries of a fuzzy interval. We show that algorithms in the interval-valued case can be adapted to fuzzy intervals considering them as crisp intervals of gradual numbers.

Keywords. Fuzzy PERT, Gradual Numbers, Floats

1. Introduction

Temporal Constraint Networks (TCN) represent relations between dates of events and also allow to express constraints on the possible durations of activities from intervals of values. TCN have been extended to take into account uncertainty of durations of some tasks in realistic applications by making a distinction between so-called contingent constraints and controllable ones \cite{11}. The resulting network becomes a decision-making problem under uncertainty. This paper reconsiders the most basic scheduling problem, that of minimizing the makespan of a partially ordered set of activities, in the context of incomplete knowledge. When the durations of tasks of a project scheduling problem are ill-known and modeled by intervals, the problem can be viewed as a special kind of TCN where all tasks are modeled by contingent constraints, and controllable constraints only describe precedence between tasks. Of course, the resulting network is always controllable if the graph of precedence constraints is acyclic. This paper answers the question of optimizing the total duration of such a network when constraints are set by fuzzy intervals.

One goal of project scheduling problem is to compute earliest starting dates, latest starting dates and floats of the different tasks, and to identify critical tasks. When tasks durations are precisely known, this problem is solved by the PERT/CPM Algorithm (Critical Path Method). When task durations are ill-known and lie in an interval, this problem is much more difficult. In recent paper \cite{8}, we give polynomial algorithms to
compute upper bound of floats in the interval-valued model. Here we prove that those algorithms can be adapted the fuzzy-valued problem. For this purpose, we will recall and use a recent notion introduced in fuzzy set theory, called gradual numbers [7]. This notion makes it possible to define and use gradual bounds of fuzzy intervals and then apply already known interval analysis ideas to the fuzzy counterpart of interval problems.

The paper is organized as follows; Section 2 gives the background needed to construct our new algorithms: Subsection 2.1 recalls the basic PERT/CPM problem of project scheduling. Subsection 2.2 re–states the recent results of [8] to solve the PERT/CPM problem when tasks are modeled by intervals. Then Subsection 2.3 gives general definition and intuitions of so–called gradual numbers. A subproblem is addressed in Section 3, that is the computation of earliest starting dates, latest starting dates and floats when durations of tasks are modeled by gradual numbers. And finally Section 4 gives algorithms to compute degrees of necessary criticality in polynomial time, and the least upper bound (LUB) of floats for each possibility degree $\lambda \in (0, 1]$.

2. Preliminaries

2.1. Classical PERT Model

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. Of major concern, is to minimize the ending time of the last task, also called the makespan of the network. For each task, three quantities have practical importance for the management of the activity network: The earliest starting time $est_{ij}$ of an activity $(i,j)$ is the date before which the activity cannot be started without violation of a precedence constraint. The latest starting time $lst_{ij}$ of an activity $(i,j)$ is the date after which the activity cannot be started without delaying the end of the project. The float $f_{ij}$ of an activity $(i,j)$ is the difference between the latest starting time $lst_{ij}$ and the earliest starting time $est_{ij}$. An activity is critical if and only if its float is equal to zero. Under the assumption of minimal makespan, critical tasks must be started and completed at prescribed time-points.

A directed, connected and acyclic graph $G = \langle V,A \rangle$, represents an activity network. We use the activity-on-arc convention. $V$ is the set of nodes (events), $|V| = n$, and $A$ is the set of arcs (activities), $|A| = m$. The set $V = \{1,2,\ldots,n\}$ is labeled in such a way that $i < j$ for each activity $(i,j) \in A$. Activity durations $d_{ij}$ (weights of the arcs) $(i,j) \in A$ are well known. See Figure 1 for a simple example of project scheduling problem. Task durations are the values given on the arc of the graph. Two nodes $1$ and $n$ are distinguished as the initial and final node, respectively. We also need some additional notations to speak about predecessor or successor of a task or a node, and subgraphs of $G$:

![Figure 1. Simple project scheduling problem](image-url)
• $\text{Succ}(i)$ (resp. $\text{Pred}(i)$) refers to the set of nodes that immediately follow (resp. precede) node $i \in V$.
• $\text{SUCC}(i, j)$ (resp. $\text{PRED}(i, j)$) denotes the set of all arcs that come after (resp. before) $(i, j) \in A$, and $\text{SUCC}(j)$ (resp. $\text{PRED}(j)$) stands for the set of all nodes that come after (resp. before) $j \in V$.
• $G(i, j)$ is the subgraph of $G$ composed of nodes succeeding $i$ and preceding $j$.
• $G(d_{ij} = d)$ is the graph where duration of task $(i, j)$ is replaced by $d$.

With these notations, we can give precise formulas defining earliest starting dates, latest starting dates and floats of tasks and events: The earliest starting date of an event $k$ is the longest path from the beginning of the project, represented by node 1, and node $k$. We arbitrary 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 following recursive formula give the key to compute earliest starting dates of events and tasks.

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

The earliest ending time for the project is the earliest starting time of the last event $n$. In order to ensure a minimal duration of the project, the latest starting date of event $n$ is equal to its earliest starting date ($lst_n = est_n$). The latest starting date of a task $(k, l)$ is the time after which we can not start its execution without delaying the end of the project, so it is in fact the difference between the earliest ending date of the project and the longest path from node $n$ and node $l$ minus the duration of the task $(k, l)$. We can now define the latest starting date of events and tasks by the following formulas:

$$lst_k = \begin{cases} 
est_n & \text{if } k = n \\
\min_{j \in \text{Succ}(k)} (lst_l - d_{lj}) & \text{otherwise}
\end{cases}$$

The float of a 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: $f_{kl} = lst_{kl} - est_{kl}$

The well known PERT/CPM algorithm computes the earliest starting dates, latest starting dates and float in linear time complexity ($O(m + n)$). It executes a forward recursion from node 1 to node $n$ to give the longest path from node 1 to each node of the graph. Then it runs a backward recursion to compute latest starting dates and so the floats.

The PERT algorithm give the following earliest starting dates, latest starting dates and floats of project of Figure 1:

- $est_{12} = 0$, $est_{23} = 2$, $est_{13} = 0$,
- $lst_{12} = 0$, $lst_{23} = 2$, $lst_{13} = 1$,
- $f_{12} = 0$, $f_{23} = 0$, $f_{13} = 1$.

2.2. Interval-Valued PERT Model

In this section we recall recent results published in the CP2005 conference proceedings [8]. Suppose the durations $d_{ij}$ of tasks $(i, j)$ of the precedent model are not precisely known, but are known to lie in an interval range $D_{ij} = [d^L_{ij}, d^U_{ij}]$. It means that the real exact duration of the task will be in $D_{ij}$, but it can neither be known nor chosen in this interval. This problem was first formulated by Buckley [1], and was completely solved very recently in [8].
An instantiation of all task durations of the network is called a configuration, and a instantiation of all task durations for which task duration is set to the minimal value $d_{ij}^{-}$ or maximal value $d_{ij}^{+}$ is called a extreme configuration. We note $\mathcal{C}$ the set of possible configurations: $\mathcal{C} = \times_{(i, j) \in D} D_{ij}$.

In this formulation, a task $(i, j)$ can be possibly critical, if there exists a configuration for which $(i, j)$ is critical in the usual sense. A task is not necessarily critical if there exists no configuration for which $(i, j)$ is critical in the usual sense. A task is necessarily critical if it is critical for each configuration. And a task is not necessarily critical is there exists at least one configuration for which the task is not critical in the usual sense. Finding possible and necessarily critical tasks is very important for a project manager. Other useful pieces of information are the possible earliest starting dates, possible latest starting dates and possible floats, which become intervals:

\[ \text{EST}_{kl} = [\text{est}_{kl}, \text{est}_{kl}^{+}] = \{\text{est}_{kl}(\Omega) | \Omega \in \mathcal{C}\} \]
\[ \text{LST}_{kl} = [\text{lst}_{kl}, \text{lst}_{kl}^{+}] = \{\text{lst}_{kl}(\Omega) | \Omega \in \mathcal{C}\} \]
\[ F_{kl} = [f_{kl}^{-}, f_{kl}^{+}] = \{f_{kl}(\Omega) | \Omega \in \mathcal{C}\} \]

where $\text{est}_{kl}(\Omega)$ (resp. $\text{lst}_{kl}(\Omega)$ and $f_{kl}(\Omega)$) represents the earliest starting date of $(k, l)$ (resp. latest starting date and float) in configuration $\Omega$.

### 2.2.1. Asserting Necessarily Criticality

Asserting if a task is possibly critical is an NP-Hard problem [2], while asserting is a task is necessarily critical is polynomially solvable [8]. As in the classical problem, criticality and floats are obviously connected:

**Proposition 1** An activity $(k, l) \in A$ is possibly (resp. necessarily) critical in $G$ if and only if $f_{kl}^{-} = 0$ (resp. $f_{kl}^{+} = 0$).

In this paper we focus on asserting the necessarily criticality of a task, and on computing the upper bound of the float, when durations are modeled by fuzzy interval. The same approach can be applied for computing latest starting dates in polynomial time, or for asserting the possible criticality of a task and computing the lower bound of the float, but it can only be done with exponential time complexity. Because of the lack of space in conference proceedings, we choose to deal with necessarily criticality and upper bound of the float. A first algorithm that asserts the necessarily criticality of a task $(k, l)$ in the interval case when predecessors of $(k, l)$ are precisely known is based on the following propositions [8]:

**Proposition 2** An activity $(k, l) \in A$ is necessarily critical in $G$ if and only if $(k, l)$ is critical in an extreme configuration in which the duration of $(k, l)$ is at its lower bound and all activities from set $A \setminus \text{SUCC}(k, l) \setminus \text{PRED}(k, l) \setminus \{(k, l)\}$ have durations at their upper bounds.

**Proposition 3** Let $(k, l) \in A$ be a distinguished activity, and $(i, j)$ be an activity such that $(i, j) \in \text{SUCC}(k, l)$. Assume that every activity $(u, v) \in \text{PRED}(i, j)$ has precise duration. If $(k, l)$ is critical in $G(1, i)$, then:

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

If $(k, l)$ is not critical in $G(1, i)$, then:

$(k, l)$ is necessarily critical in $G \iff (k, l)$ is necessarily critical in $G(d_{ij} = d_{ij}^{+})$. 
Proposition 2 and 3 lead to an algorithm for asserting the necessary criticality of a
given activity \((k,l)\) in a network in which all activities that precede \((k,l)\) have precise
durations. The algorithm recursively assigns an exact duration to tasks succeeding \((k,l)\)
without changing the necessarily criticality of \((k,l)\).

The algorithm works as follows: At the initialization, we set all durations of tasks not
succeeding \((k,l)\) to their upper bound, and duration of \((k,l)\) to its lower bound (according
to Proposition 2). Then at the first iteration of the algorithm, we only consider the sub–
network \(G(1,l)\). In this network, all tasks durations are precisely fixed, so a classical
PERT/CPM run decides the necessary criticality of \((k,l)\) in \(G(1,l)\). Now, according
to Proposition 2 we can set to a precise duration tasks immediately succeeding node \(l\):
if \((k,l)\) is necessary critical in \(G(1,l)\), duration of tasks \(\langle l,m \rangle\) \((m \in Succ(l))\) are set
to their minimal duration, otherwise, they are set to their maximal duration. Then the
second iteration considers the sub–network \(G(1,l+1)\). At this step all tasks durations are
precisely set in this sub–network, we can check the criticality of \((k,l)\) with the standard
PERT/CPM algorithm, and so on. After the last iteration, all tasks durations are precisely
instantiated and \((k,l)\) is critical in this instantiation, if and only if task \((k,l)\) is necessary
critical in the original network \(G(1,n)\) (where duration of tasks not preceding \((k,l)\) are
intervals).

The next proposition will lead to a polynomial algorithm which asserts the necessary
criticality of a task \((k,l)\) in the general case:

**Proposition 4** Let \((k,l) \in A\) be a distinguished activity, and \((i,j)\) be an activity such that
\((i,j) \in PRED(k,l)\). If \((k,l)\) is necessarily critical in \(G(j,n)\), then:

\((k,l)\) is necessarily critical in \(G \iff (k,l)\) is necessarily critical in \(G(d_{ij} = d_{ij}^-)\)

If \((k,l)\) is not necessarily critical in \(G(j,n)\), then:

\((k,l)\) is necessarily critical in \(G \iff (k,l)\) is necessarily critical in \(G(d_{ij} = d_{ij}^+)\)

With the help of the previous algorithm, Proposition 4 leads to an algorithm which
recursively assigns an exact duration to tasks preceding \((k,l)\) without changing the nec-
essarily criticality of \((k,l)\). At the first iteration, we consider the sub–network \(G(k,n)\).
In this network all tasks preceding \((k,l)\) have precise durations (because there is no task
preceding \((k,l)\)). So we can invoke the previous algorithm to know if \((k,l)\) is necessary
critical in \(G(k,n)\). Now, with the help of Proposition 4, we can precisely set the dura-
tion of tasks just preceding node \(k\). At the second iteration, we consider the sub–network
\(G(k-1,n)\), where all tasks preceding \((k,l)\) are now precisely fixed, so we can call the
previous algorithm (with original interval durations for tasks succeeding \((k,l)\), and so
on. At the end of this Algorithm, all durations of tasks preceding \((k,l)\) are precisely set,
and \((k,l)\) is necessary critical in the original network if and only if it is necessary critical
in the instantiated network. So finally, putting together Proposition 3 and 4 we can assert
if a task \((k,l)\) is necessarily critical in \(O[mn]\) time complexity.

The two previous algorithms used conjointly instantiate all task duration in a net-
work without changing the necessary criticality of a given task \((k,l)\). That means that
\((k,l)\) is necessarily critical if and only if its float is null in the so–built configuration. If
\((k,l)\) is necessary critical the LUB of the float have been computed and is \(f_{kl}^* = 0\). But,
if this float is not null in the constructed configuration, then the computed value of the
float is not the correct LUB of float of \((k,l)\). We call this computed value the potential
float of \((k,l)\).
2.2.2. Computing Least Upper Bound of Floats

The key idea to compute the least upper bound (LUB) on float $f^+_{kl}$ of activity $(k,l)$ in $G$, is to increase the minimal duration $d^*_{kl}$ until $(k,l)$ becomes necessarily critical. Then it is proved that the LUB of float is equal to the overall increment of $d^*_{kl}$ [8]:

**Proposition 5** Let $f^+_{kl}$ be the minimal nonnegative real number such that $(k,l)$ is necessarily critical in $G(d^*_{kl} = d^*_{kl} + f^+_{kl})$. Then $f^+_{kl} = f^*_{kl}$.

Let us explain how to find this increment. One can note that if a task $(k,l)$ is necessarily critical in $G$, then there exists a path $p$ from node 1 to node $k$ such that for every node $j \in p$, $(k,l)$ is necessarily critical in $G(j,n)$ [8]. So the minimal increment that can make $(k,l)$ necessarily critical is given by next lemma:

**Lemma 1** Let $\Delta = \min \{f^+_{kl}(j,n) \mid f^+_{kl}(j,n) \neq 0\}$ where $f^+_{kl}(j,n)$ is the potential float of task $(k,l)$ in $G(j,n)$, $j \in \text{PRED}(k)$ (i.e. $f^+_{kl}(j,n)$ is the float in the configuration induced by the algorithm which asserts if $(k,l)$ is necessary critical).

Then for all $\epsilon < \Delta$, activity $(k,l)$ is not necessarily critical in $G(j,n)$, $j \in \text{PRED}(k)$, with duration $d^*_{kl} = d^*_{kl} + \epsilon$ (the configuration constructed for asserting the necessary criticality is the same).

Moreover there exists $j^* \in \text{PRED}(k)$ such that $(k,l)$ becomes necessarily critical in $G(j^*,n)$ with duration $d^*_{kl} = d^*_{kl} + \Delta$ (the configuration constructed for asserting the necessary criticality is not the same).

The algorithm which computes the LUB of floats works as follow: For each node $i$ preceding $(k,l)$ it computes the potential float of $(k,l)$ in $G(i,n)$ with the algorithm which assert the necessary criticality of $(k,l)$. If the potential float of $(k,l)$ in $G(1,n)$ is not 0, then the minimal potential float computed is added to the minimal duration $d^*_{kl}$ of $(k,l)$, and the algorithm is run again from the beginning until $(k,l)$ becomes necessary critical in $G$. The total added value to the minimal duration $d^*_{kl}$ so far is then the LUB of the floats of $(k,l)$.

We are going to see that those algorithms can be easily adapted to the fuzzy-valued version of this problem, using the new notion of *gradual numbers*, thus providing in polynomial time the degree of necessarily criticality of a task.

2.3. Gradual Numbers

To solve the fuzzy interval valued model PERT, we will use in this paper the recent notion of *gradual numbers* [7].

What is often called a fuzzy number [12,5] is the extension of an interval, not a real number. Hence, fuzzy interval is a better name for a fuzzy set of reals whose cuts are intervals. In a fuzzy interval, it is the interval that models incomplete knowledge (we know that some parameter lies between two bounds), not the fuzziness per se. Intervals model uncertainty in a Boolean way: a value in the interval is possible; a value outside is impossible. What fuzziness brings is to make the boundaries of the interval softer, thus making uncertainty gradual.

In order to model the essence of graduality without uncertainty, gradual numbers are defined as follow:
**Definition 1 (Gradual real number [7])** A gradual real number (or gradual number for short) \( \tilde{r} \) is defined by an assignment function \( \mathcal{A}_r \) from \((0, 1]\) (the unit interval minus 0) to the reals. We note \( \mathcal{G} \) the set of all gradual real numbers.

A gradual number is a number parametrized by a value \( \alpha \) which range in \((0, 1]\). Intuitively, an element \( \alpha \in (0, 1] \) is the degree of plausibility of a scenario, and \( \mathcal{A}_r(\alpha) \) is the value of some parameter in this scenario. Using the notion of gradual number, we can describe a fuzzy interval \( M \) by an ordered pair of specific gradual numbers \((\tilde{m}^-, \tilde{m}^+)\). \( \tilde{m}^- \) is called the gradual lower bound of \( M \) and \( \tilde{m}^+ \) the gradual upper bound. To ensure the well known shape of a fuzzy interval, several properties of \( \tilde{m}^- \) and \( \tilde{m}^+ \) must hold:

- the domains of \( \mathcal{A}_{\tilde{m}^-} \) and \( \mathcal{A}_{\tilde{m}^+} \) must be \((0, 1]\)
- \( \mathcal{A}_{\tilde{m}^-} \) need to be increasing
- \( \mathcal{A}_{\tilde{m}^+} \) need to be decreasing
- \( \tilde{m}^- \) and \( \tilde{m}^+ \) must be well ordered (\( \mathcal{A}_{\tilde{m}^-} \leq \mathcal{A}_{\tilde{m}^+} \)).

Such a pair of gradual numbers intuitively describe a fuzzy interval with membership function:

\[
\mu_M(x) = \begin{cases} 
\sup\{\lambda | \mathcal{A}_{\tilde{m}^-}(\lambda) \leq x \} & \text{if } x \in \mathcal{A}_{\tilde{m}^-}((0, 1]) \\
1 & \text{if } \mathcal{A}_{\tilde{m}^-}(1) \leq x \leq \mathcal{A}_{\tilde{m}^+}(1) \\
\sup\{\lambda | \mathcal{A}_{\tilde{m}^+}(\lambda) \geq x \} & \text{if } x \in \mathcal{A}_{\tilde{m}^+}((0, 1]) \\
0 & \text{otherwise}
\end{cases}
\]

An upper semi continuous membership function \( \mu_M(\cdot) \), can be described by the pair of gradual reals \((\tilde{m}^-, \tilde{m}^+)\) defined by

\[
\begin{align*}
\mathcal{A}_{\tilde{m}^-} : (0, 1] &\rightarrow \mathbb{R} \\
\lambda &\mapsto \mathcal{A}_{\tilde{m}^-}(\lambda) = \inf\{x|\mu_M(x) \geq \lambda\} \\
\mathcal{A}_{\tilde{m}^+} : (0, 1] &\rightarrow \mathbb{R} \\
\lambda &\mapsto \mathcal{A}_{\tilde{m}^+}(\lambda) = \sup\{x|\mu_M(x) \geq \lambda\}
\end{align*}
\]

To know more about general framework of gradual numbers, reader should refer to the following papers: [7,3,6].

The sum of gradual numbers \( \tilde{r} \) and \( \tilde{s} \) is simply defined by summing their assignment functions. It is \( \tilde{r} + \tilde{s} \) such that \( \forall \alpha \in (0, 1], \mathcal{A}_{\tilde{r} + \tilde{s}}(\alpha) = \mathcal{A}_r(\alpha) + \mathcal{A}_s(\alpha) \). Most algebraic properties of real numbers are preserved for gradual real numbers, contrary to the case of fuzzy intervals. For example, the set of the gradual real numbers with the addition operation forms a commutative group with identity \( \tilde{0} \) (\( \mathcal{A}_0(\lambda) = 0, \forall \lambda \in (0, 1]\)). Indeed the gradual real number \( \tilde{r} \) has an inverse \( -\tilde{r} \) under the addition: \( \mathcal{A}_{-\tilde{r}}(\alpha) = -\mathcal{A}_r(\alpha) \), and \( \tilde{r} + (\tilde{r}) = 0 \). Classical arithmetic operations are defined intuitively: If \( \tilde{r} \) and \( \tilde{s} \) are two gradual numbers,

- addition: \( \mathcal{A}_{\tilde{r} + \tilde{s}}(\alpha) = \mathcal{A}_r(\alpha) + \mathcal{A}_s(\alpha) \).
- subtraction: \( \mathcal{A}_{\tilde{r} - \tilde{s}}(\alpha) = \mathcal{A}_r(\alpha) - \mathcal{A}_s(\alpha) \).
- multiplication: \( \mathcal{A}_{\tilde{r} \times \tilde{s}}(\alpha) = \mathcal{A}_r(\alpha) \times \mathcal{A}_s(\alpha) \).
- division: \( \mathcal{A}_{\frac{\tilde{r}}{\tilde{s}}}(\alpha) = \frac{\mathcal{A}_r(\alpha)}{\mathcal{A}_s(\alpha)} \) if \( \forall \lambda \in (0, 1], \mathcal{A}_s(\alpha) \neq 0 \).
- maximum: \( \max(\mathcal{A}_r, \mathcal{A}_s)(\alpha) = \max(\mathcal{A}_r(\alpha), \mathcal{A}_s(\alpha)) \).
- minimum: \( \min(\mathcal{A}_r, \mathcal{A}_s)(\alpha) = \min(\mathcal{A}_r(\alpha), \mathcal{A}_s(\alpha)) \).

Note that contrary to real numbers, the maximum operation on gradual numbers is not selective: \( \max(\mathcal{A}_r, \mathcal{A}_s) \neq \mathcal{A}_r \) or \( \mathcal{A}_s \). There are sub–ranges of \((0, 1]\) where \( \max(\mathcal{A}_r, \mathcal{A}_s) = \mathcal{A}_r \) and it is \( \mathcal{A}_s \) in the complementary range. If \( \mathcal{A}_r \) and \( \mathcal{A}_s \) are linear, these ranges are of the form \((0, \theta]\) and \([\theta, 1]\) for a threshold \( \theta \).
3. The Gradual-Valued Project Scheduling Problem

As in the previous section, an activity network is represented by a directed, connected and acyclic graph $G = <V,A>$. But now, activity durations $\tilde{d}_{ij}$ ($(i,j) \in A$) are defined by gradual numbers. It means that all durations depend on a parameter to be chosen (for instance indexing scenarios).

This problem formulation is given to validate the PERT algorithm with gradual numbers which is used in next sections. However we can see this problem as an optimization problem on a family of scenarios: for a given $\lambda$, $\tilde{d}_{ij}(\lambda)$ is a precise duration, and so configuration (set of the durations) at degree $\lambda$ is a particular scenario. This parametric approach can model dependencies between tasks durations, but only with one degree of freedom. Figure 2 provides gradual durations of tasks of the project of Figure 1. $\tilde{d}_{13}(\lambda) = 2 \ \forall \lambda$ means that the duration of task $(1,3)$ is 2 in each considered scenario. We can note on this example that the duration of tasks $(1,2)$ and $(2,3)$ are correlated, and the more time the task $(1,2)$ will require, the faster the task $(2,3)$ will be executed.

Computing earliest starting date, latest starting date and float of this problem is very simple: we just need to run the standard PERT Algorithm, where $+, -, \max$ and $\min$ are the four operations on gradual numbers seen in the previous section. This is due to the fact that all algebraic properties needed to apply the PERT Algorithm are preserved by the gradual numbers, for example the fact that $<\mathbb{G}, \max, +>$ is an idempotent semi-ring, validates the forward recursion of the PERT Algorithm. The backward recursion is possible since the addition operation as an inverse ($<\mathbb{G}, +>$ is a commutative group), so contrary to the interval–valued problem, we do not run the risk of counting any uncertainty twice. For more detail, the reader should refer to [7,9].

Algorithm 1 works as follow: Line 1 is the initialization to $\tilde{0}$ of all the earliest starting dates of events (nodes of the graph). Part from line 2 to line 5 is the standard forward recursion which computes earliest starting dates. Line 6 is initialization of the latest starting dates. Lines 7 to 11 is the standard backward recursion which computes latest starting dates and float. The main difference between this algorithm and the usual PERT/CPM algorithm is in the form of the results, because $\max$ and $\min$ are not selective, a characteristic of a task may not correspond to a single configuration. For instance, a task may be critical for some values of $\lambda \in [0,1]$ and not critical for other values.

We present on Figure 3 the results of the computation of the earliest starting date, latest starting date and float of task $(2,3)$. It means that for all scenario generated with $\lambda \geq \frac{1}{7}$, the task $(2,3)$ is critical, but it is not for $\lambda \leq \frac{1}{7}$. In practice, we must store this information describing a more complex situation than in the deterministic case.
Algorithm 1: GradualPERT

Input: A network $G = \langle V, A \rangle$, activity duration is known as gradual numbers.

Output: Gradual earliest starting date $\tilde{est}_{ij}$, gradual latest starting date $\tilde{lst}_{ij}$ and gradual float $\tilde{f}_{ij}$ of each task $(i, j)$ of the network.

1 foreach $i \in [1..n]$ do $\tilde{est}_i \leftarrow 0$
2 for $i \leftarrow 1$ to $n - 1$ do
3 foreach $j \in \text{Succ}(i)$ do
4 $\tilde{est}_{ij} \leftarrow \tilde{est}_i$
5 $\tilde{est}_j \leftarrow \max(\tilde{est}_j, \tilde{est}_i + \tilde{d}_{ij})$
6 foreach $i \in [1..n]$ do $\tilde{lst}_i \leftarrow \tilde{est}_n$
7 for $j \leftarrow n$ downto 2 do
8 foreach $i \in \text{Pred}(j)$ do
9 $\tilde{lst}_{ij} \leftarrow \tilde{lst}_j - \tilde{d}_{ij}$
10 $\tilde{f}_{ij} = \tilde{lst}_{ij} - \tilde{est}_{ij}$
11 $\tilde{lst}_i \leftarrow \min(\tilde{lst}_i, \tilde{lst}_j - \tilde{d}_{ij})$

Figure 3. Gradual earliest starting date, latest starting date and float

4. Fuzzy Interval Valued PERT

In this section, we suppose that durations of tasks of the project are given by fuzzy intervals. A simple model of a fuzzy task duration for a regular PERT user can be a triangular fuzzy interval whose core is the most plausible duration and whose support is the interval of all possible durations. We show that we can assert the necessarily criticality degree of a task in polynomial time complexity. Then we will give an algorithm to obtain the gradual upper bound of the float of a given task.

4.1. Asserting Necessarily Criticality

We have seen in Section 2.2 that we can precisely set the duration of all tasks of the network without changing the criticality of a given task $(k, l)$ when task duration are modeled by crisp intervals. To assert the degree of necessary criticality of a task $(k, l)$ in the fuzzy case, we can use the same type of algorithm, using gradual numbers instead of real numbers. But from Proposition 3 (resp. 4), to fix the precise gradual duration of a task $(i, j)$, we have to test if the task $(k, l)$ is critical in the subnetwork $G(j, n)$ (resp. $G(1, j)$). We have seen in the previous section that on a gradual PERT, tasks can be critical for some possibility degree and not critical for others (see example on Figure 3).
This means that the gradual duration assigned to task \((i, j)\) may involve both upper and lower gradual bounds of the duration \(\hat{D}_{ij} = [\hat{d}_{ij}^-, \hat{d}_{ij}^+]\).

Algorithm 2:

**Input:** A network \(G\), activity \((k, l)\), fuzzy interval durations \(\hat{D}_{uv} = [\hat{d}_{uv}^-, \hat{d}_{uv}^+]\), \((u, v) \in A\) and for every task in \(PRED(k, l)\) the duration \(\hat{d}_{kl}\) is precisely given (but gradual).

**Output:** \(\hat{f}_{kl}^\lambda\), the “potential” maximal float of task \((k, l)\): \(\hat{f}_{kl}^\lambda(\lambda) = 0\) for all possibility degree for which the task is necessarily critical, \(\hat{f}_{kl}^\lambda(\lambda) > 0\) for all possibility degree for which the task is not necessarily critical, but in this case \(\hat{f}_{kl}^\lambda(\lambda)\) is not the exact float of \((k, l)\).

1. **foreach** \((u, v) \notin SUCC(k, l)\) do \(\hat{d}_{uv} \leftarrow \hat{d}_{uv}^+\);
2. \(\hat{d}_{kl} \leftarrow \hat{d}_{kl}^-\);
3. **for** \(i \leftarrow 1\) to \(n - 1\) **such that** \(i \in SUCC(l) \cup \{l\}\) do
4.  **foreach** \(\lambda \in (0, 1]\) such that \(\hat{f}_{kl}^\lambda(\lambda) = 0\) do
5.   **foreach** \(j \in Succ(i)\) do \(\hat{d}_{ij}(\lambda) \leftarrow \hat{d}_{ij}^-\);
6.  **foreach** \(\lambda \in (0, 1]\) such that \(\hat{f}_{kl}^\lambda(\lambda) > 0\) do
7.   **foreach** \(j \in Succ(i)\) do \(\hat{d}_{ij}(\lambda) \leftarrow \hat{d}_{ij}^+\);
8. return \(\hat{f}_{kl}^\lambda\);

Line 1 of Algorithm 2 initializes gradual durations of all tasks of the network not succeeding \((k, l)\) to their maximal gradual bounds, according to Proposition 2. Lines 3~8 assign gradual values for the tasks succeeding \((k, l)\), depending on the criticality of \((k, l)\), with respect to Proposition 4. The creation of the gradual duration \(\hat{d}_{ij}\) (Lines 5 and 8) may seem complex, but is very simple to implement in practice, for example, when durations are given by piecewise linear fuzzy intervals [10]. Then gradual bounds of those interval can be simply modeled by the lists of their kinks. All computed gradual durations are also piecewise linear, since operations \(+, -, \text{max}\) and \(\text{min}\) preserve the piecewise linearity. In this context, finding all \(\lambda \in (0, 1]\) such that \(\hat{f}_{kl}^\lambda(\lambda) = 0\) remains to find the pairs of adjacent kinks whose abscissa is 0.

Note that the result \(\hat{f}_{kl}^\lambda\) of Algorithm 2 is not the correct maximal gradual float \(\hat{f}_{kl}^\lambda\) of \((k, l)\): it is correct only if \(\hat{f}_{kl}^\lambda(\lambda) = 0\). So this Algorithm is useful to know the Necessary degree criticality of a task when its predecessors are precisely (but gradually) fixed, and will be invoked in the next algorithm.

Algorithm 3 computes the necessary criticality degree of a given task \((k, l)\) when all task durations are fuzzy interval valued. It recursively assigns precise (but gradual) durations to tasks preceding node \(k\), due to to Proposition 4. The result is a gradual “potential” maximal float \(\hat{f}_{kl}^\lambda\) of task \((k, l)\), which means that \((k, l)\) is necessarily critical at degree at least \(\lambda\) if and only if \(\hat{f}_{kl}^\lambda(\lambda) = 0\). But if \(\hat{f}_{kl}^\lambda(\lambda) > 0\), then \(\hat{f}_{kl}^\lambda(\lambda)\) is not the correct maximal float of \((k, l)\) at possibility degree \(\lambda\). Computing the exact maximal float \(\hat{f}_{kl}^\lambda(\lambda)\) at possibility degree \(\lambda\) is the subject of the next paragraph.
Algorithm 3:

Input: A network $G$, activity $(k,l)$, fuzzy interval durations $D_{uv} = [d_{uv}^L, d_{uv}^U]$, $(u,v) \in A$.

Output: $\bar{f}_{kl}^r$ the “potential” maximal float of task $(k,l)$: $\bar{f}_{kl}^r(\lambda) = 0$ for all possibility degree for which the task is necessarily critical, $\bar{f}_{kl}^r(\lambda) > 0$ for all possibility degree for which the task is not necessarily critical, but in this case $\bar{f}_{kl}^r(\lambda)$ is not the exact float of $(k,l)$.

1 foreach $j \leftarrow k$ downto 2 such that $i \in PRED(k) \cup \{k\}$ do
2 \hspace{1em} $f_{kl}^r \leftarrow$ Algorithm 2 with $G(j,n)$;
3 foreach $\lambda \in (0,1]$ such that $\bar{f}_{kl}^r(\lambda) = 0$ do
4 \hspace{2em} foreach $i \in Pred(j)$ do $d_{ij}(\lambda) \leftarrow \bar{d}_{ij}^r(\lambda)$;
5 foreach $\lambda \in (0,1]$ such that $\bar{f}_{kl}^r(\lambda) > 0$ do
6 \hspace{2em} foreach $i \in Pred(j)$ do $d_{ij}(\lambda) \leftarrow d_{ij}^r(\lambda)$;
7 return $\bar{f}_{kl}^r$.

Algorithm 4:

Input: A network $G$, activity $(k,l)$, fuzzy interval durations $D_{uv}$, $(u,v) \in A$.

Output: The gradual least upper bound $\bar{f}_{kl}^+$ of floats of $(k,l)$.

1 foreach $i \in Pred(k)$ do
2 \hspace{1em} $\bar{f}_{kl}^i(i,n) \leftarrow \bar{f}_{kl}^r$ with Algorithm 3 in network $G(i,n)$;
3 $\bar{f}_{kl}^+ \leftarrow 0$; $\bar{d}_{kl} \leftarrow \bar{d}_{kl}^r$;
4 while $\bar{f}_{kl}^i(1,n) \neq 0$ do
5 \hspace{2em} foreach $\lambda \in (0,1]$ do
6 \hspace{3em} $\Delta(\lambda) \leftarrow \min\{\bar{f}_{kl}^i(j,n)(\lambda) \mid j \in PRED(k), \bar{f}_{kl}^i(j,n)(\lambda) \neq 0\}$;
7 \hspace{3em} $\bar{d}_{kl} \leftarrow \bar{d}_{kl} + \Delta$;
8 \hspace{2em} foreach $i \in Pred(k)$ do
9 \hspace{3em} $\bar{f}_{kl}^i(i,n) \leftarrow \bar{f}_{kl}^i$ with Algorithm 3 in network $G(i,n)$;
10 return $\bar{f}_{kl}^+$.

4.2. Computing Gradual Least Upper Bound of Floats

We can now compute the gradual upper bound of the floats of a task $(k,l)$. The idea is to make $(k,l)$ necessarily critical for all $\lambda$ by adding a gradual duration $\Delta$ to $\bar{d}_{kl}^r$. This is done by Algorithm 4. The first step of the algorithm computes $\bar{f}_{kl}^r(i,n)$, the “potential float” of $(k,l)$ in the network $G(i,n)$ (lines 1 and 2). In practice, only one iteration of Algorithm 3 is needed to obtain $\bar{f}_{kl}^r(i,n)$ for all $i \in Pred(k)$. The second step of the algorithm constructs a gradual duration $\Delta$ to add to $\bar{d}_{kl}^r$ in order to make $(k,l)$ necessarily critical in at least one more sub–network $G(i,n)$, for all possibility degree (lines 5 and 6). In practice, constructing $\Delta$ is easy, we just need to restrict all the $\bar{f}_{kl}^i(i,n)$ on a domain for which $\bar{f}_{kl}^i(i,n)(\lambda) \neq 0$ and $\bar{f}_{kl}^i(1,n)(\lambda) \neq 0$, and take their minimum, as defined in Section 2.3. Algorithm 4 repeats this step until $(k,l)$ becomes necessarily critical in $G(1,n)$ for all possibility degree. The total gradual increase of $\bar{d}_{kl}^r$ is then the gradual float
of \((k,l)\). By definition of the gradual numbers, the obtained \(\hat{f}^+_{kl}\) is such that \(\hat{f}^+_{kl}(\lambda)\) is the exact LUB of floats of task \((k,l)\) at possibility degree \(\lambda\), and the degree of criticality is \(1 - \inf \{ \lambda | \hat{f}^+_{kl}(\lambda) = 0 \} \).

**Remark:** In the interval–valued problem, asserting possible criticality and computing lower bound of floats are NP–Hard problems [8]. However efficient algorithms exist and can be adapted to the fuzzy interval case with the help of gradual numbers. One of these algorithms consists in running a PERT/CPM on configurations where all task durations are set to their lower bounds but on a path from node 1 to node \(n\) [4]. The lower bound of the float of a task is then the minimal computed float over the configurations issued from all possible paths. This method is directly applicable to the fuzzy–valued problem, using gradual bounds of fuzzy intervals, instead of usual interval end–points.

### 5. Conclusion

In this paper, we generalize some recent results of [8] for project scheduling problems under uncertainty, when task durations are modeled by fuzzy intervals. We give algorithms that assert the degree of necessarily criticality of a given task in polynomial time. And we construct an algorithm to compute the least upper bound of floats in polynomial time for each possibility degree \(\lambda \in (0, 1]\). Those algorithms are based on the very recent notion of gradual number, which provides a new way of looking at fuzzy intervals as classical intervals of gradual numbers. Thus allowing to apply interval analysis methods to fuzzy intervals directly.

### References


