Fast minimum float computation in activity networks under interval uncertainty

Thierry Garaix[1,2,3], Christian Artigues[1,2] and Cyril Briand[1,2]

[1] CNRS ; LAAS ; 7 avenue du colonel Roche, F-31077 Toulouse, France
[2] Université de Toulouse ; UPS, INSA, INP, ISAE ; LAAS ; F-31077 Toulouse, France
[3] D.A.I., Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129, Torino, Italy

Abstract

This paper takes interest in project scheduling under uncertainty. The project is modeled as an activity-on-node network, each activity having an uncertain duration represented by an interval. The problem of computing the minimum float of each activity over all duration scenarios is addressed. For solving this NP-hard problem, Dubois *et al.* [5] and Fortin *et al.* [7] have recently proposed an algorithm based on path enumeration. In this paper, new structural properties of optimal solutions are established, as well as a new lower bound, which is used for designing an efficient branch-and-bound procedure. For matter of comparison, two mixed integer programming formulations are also proposed. Computational experimentations have been conducted on a large variety of randomly generated problem instances. The results show that the proposed branch-and-bound procedure is very fast and consistently outperforms the MIP formulations and the path enumeration algorithm.

1 Introduction

An important part of recent scheduling research aims at tackling uncertainty, which occurs, under various forms, in practical applications. Dealing with activity scheduling, a basic form of uncertainty involves uncertain durations. There are several ways to model such uncertainty. The duration of each activity can be seen as a random variable with a given probability distribution, which falls in the category of stochastic scheduling [11, 1]. When probability distributions are not available, the so-called robust scheduling approaches prefer to model uncertainty as a set of possible execution scenarios, aiming at providing performance guarantees on a worst-case analysis basis [10, 8].

In the latter category, we consider interval uncertainty for the activity durations. Each activity has a duration that belongs to a given closed interval, which yields an infinite number of scenarios [9]. In this context, one of the simplest scheduling problems is considered: the (resource-unconstrained) project scheduling problem with simple precedence constraints, or PERT scheduling. This central problem consists in computing the activity earliest start times, latest start times and total floats. When durations are precisely known, these parameters can easily be determined by computing longest paths in the corresponding activity-on-node (or activity-on-arc) network. Under interval uncertainty, the underlying best- and worst-case scenario analysis aims at computing the minimum and maximum values of the activity earliest start times, latest start times and floats. Indeed, it is of interest for a decision maker to know what would be the optimistic and pessimistic latest start time of an activity, without questioning the announced project completion time. As well, determining the minimum float of an activity yields information about its degree of potential criticality under the worst-case scenario. According to Chanas and Zielínski [2], computing the minimum float of an activity is strongly NP-hard, and remains NP-hard in an acyclic planar network of node degree 3 [3], all other problems remaining polynomial. Complexity results and their origin are synthesized in a recent paper by Fortin *et al* [7].

In the present paper, the NP-hard minimum float computation problem is addressed. For this problem, Dubois *et al.* propose the so-called path algorithm [5], which used path enumeration. Computational experiments reported in [7] show that, while the path algorithm works well for low-to-medium density precedence networks, its performance collapses for high density networks. In this paper, new structural properties of optimal solutions are exhibited that allow to propose an alternative algorithm which computes efficiently minimum floats for all types of networks, including high density precedence networks.

The paper is structured as follows. Section 2 formally defines the minimum activity float problem. Section 3 recalls some already known structural properties of optimal solutions and proposes new ones. Section 4 presents the path algorithm of Dubois *et al* and provides four new solving approaches: an improved variant of the path algorithm, a branch-and-bound algorithm, as well as two mixed-integer programming formulations. Section 5 is devoted to intensive computational experiments to compare the various approaches. Conclusions are drawn in Section 6.

2 Notations and problem definition

2.1 Activity-on-node network

An activity-on-node project network $G(V, A)$ is considered. The set of nodes $V = \{0, \ldots, n+1\}$ represents activities and the set of arcs A represents simple precedence relations among the activities such that G is a directed acyclic graph (DAG). Let $\Gamma_i = \{j \in V | (i,j) \in A\}$ denote the set of successors of activity i and let $\Gamma_i^{-1} = \{j \in V | (j,i) \in A\}$ denote the set of predecessors of activity i. Nodes 0 and $n+1$ are dummy start and end activities such that $\Gamma_0^{-1} = \emptyset$ and $\Gamma_{n+1} = \emptyset$, respectively. G being connected, all other sets of predecessors and successors are non-empty.

2.2 Duration scenarios

The duration of an activity $i \in V$ is denoted d_i, which is an uncertain parameter belonging to a given interval $[d_i^{\min}, d_i^{\max}]$. As 0 and $n+1$ correspond to dummy start and end activities, respectively, we consider that $d_0^{\min} = d_0^{\max} = d_{n+1}^{\min} = d_{n+1}^{\max} = 0$. $\mathcal{D}$ denotes the set of possible realizations of duration vector d and is defined as follows:

$$\mathcal{D} = \{d \in \mathbb{R}^{n+2} | d^{\min} \leq d \leq d^{\max}\}.$$

2.3 Paths

A path p in G is an ordered set of activities $(i_1, i_2, \ldots, i_z)$ of V^z, $z \geq 1$ where $(i_q, i_{q+1}) \in A$ for $z \geq 2$ and $1 \leq q \leq z-1$. For ease of notation, a path made of a single activity $i \in V$ may be denoted i in place of (i). In addition, $p_{i_k \to i_l}$, with $k < l$, refers to as a subpath of p between two activities i_k and i_l both belonging to p. Then a path p may also be denoted by the concatenation of two paths $(p_{i_1 \to i_l}, p_{i_l \to i_z})$, with $2 \leq l \leq z-1$.

In the sequel, $\mathcal{P}$ refers to as the set of paths in G (not necessarily from 0 to $n+1$) and $\mathcal{P}_i$ denotes the set of paths p in $\mathcal{P}$ such that $i \in p$. We also define $\mathcal{P}_i^+$ as the set of paths starting with i and $\mathcal{P}_i^-$ as the set of paths terminating at i. $\mathcal{P}_{i,j}$ is the set of paths from i to j, i.e. $\mathcal{P}_{i,j} = \mathcal{P}_i^+ \cap \mathcal{P}_j^-$.

2.4 (Longest) path lengths, start times and floats

Consider a realization (or scenario) $d \in \mathcal{D}$. The following values can be defined as functions of scenario d.

The length of path $p = (i_1, i_2, \ldots, i_z)$ is equal to the sum of the duration of the activities located on p, except the last-one i.e.:

$$l_p(d) = \sum_{q=1}^{z-1} d_{i_q}$$

The length of the longest path from activity a to activity b for scenario d is denoted by

$$l^*_{a,b}(d) = \max_{p \in \mathcal{P}^+_a \cap \mathcal{P}^-_b} l_p(d)$$

The earliest start time of activity i, denoted est_i, is equal to the length of the longest path terminating at i.

$$est_i(d) = l^*_{0,i}(d) = \max_{p \in \mathcal{P}^-_i} l_p(d)$$

The latest start time of activity i, denoted lst_i, is equal to the length of the longest path in G (the minimum project duration) minus the length of the longest path from i to $n+1$ (minimum elapsed time from the start of i to the end of the project).

$$lst_i(d) = l^*_{0,n+1}(d) - l^*_{i,n+1}(d) = \max_{p \in \mathcal{P}} l_p(d) - \max_{p \in \mathcal{P}^+_i} l_p(d)$$

The float of activity i for realized duration vector d is equal to the difference between the latest start time and the earliest start time of the activity.

$$f_i(d) = lst_i(d) - est_i(d)$$

Equivalently, the float can be expressed as the difference between the length of the longest path in G and the length of the longest path in G traversing i.

$$f_i(d) = l^*_{0,n+1}(d) - l^*_{i,n+1}(d) - l^*_{0,i}(d) = \max_{p \in \mathcal{P}} l_p(d) - \max_{p \in \mathcal{P}_i} l_p(d)$$

Given a scenario d, all the above-defined values can be computed in polynomial time by computing the relevant longest paths.

2.5 Problem statement

The n following minimization problems are considered, which amount to computing the minimum float of each activity i over the scenario set:

$$\min_{d \in \mathcal{D}} f_i(d), \quad \forall i \in V \setminus \{0, n+1\}.$$

As already mentioned, their resolution is strongly NP-hard [2, 3].

3 Structural properties of optimal solutions

Subsections 3.1 and 3.2 both recall some already known structural properties of optimal solutions. Subsection 3.3 presents a new one.

3.1 Extreme scenarios

Let $\mathcal{D}^{\text{ext}}$ denotes the set of *extreme scenarios*, i.e.:

$$\mathcal{D}^{\text{ext}} = \left\{ d \in \mathcal{D} | \forall i \in V, d_i \in \{d_i^{\min}, d_i^{\max}\} \right\}$$

Dubois *et al* [6] established the following fundamental property

Theorem 1 ([6]). *For each $i \in V$, there always exists an optimal extreme scenario d such that:*

$$\min_{d \in \mathcal{D}} f_i(d) = \min_{d \in \mathcal{D}^{ext}} f_i(d).$$

This is an important result since the solution space, initially containing an infinite number of scenarios, can be reduced to a set of 2^n scenarios.

3.2 Path-induced extreme scenarios

Given a path p, let $d^{\text{ext}}(p) \in \mathcal{D}^{\text{ext}}$ denote the *path-induced extreme scenario* where every activity $i \in p$ is set to $d_i^{\max}$, whereas all activities $i \in V \setminus p$ are set to $d_i^{\min}$.

Under scenario d, p being the longest path in $\mathcal{P}_i \cap \mathcal{P}_{0,n+1}$, the float $f_i(d)$ is defined by $l^*_{0,n+1}(d) - l_p(d)$. Hence, for minimizing the float, one intuitively seeks to minimize the length $l^*_{0,n+1}(d)$ of the longest paths, while maximizing the duration of the activities along p. Dubois *et al* [5] show that for each candidate path $p \in \mathcal{P}_i \cap \mathcal{P}_{0,n+1}$, there exists a dominant path-induced extreme scenario. This property is stated below in Theorem 2.

Theorem 2 ([5]). *For each $i \in V$, there always exists a path-induced extreme scenario $d^{ext}(p)$, with $p \in \mathcal{P}_i \cap \mathcal{P}_{0,n+1}$, such that:*

$$\min_{d \in \mathcal{D}} f_i(d) = \min_{p \in \mathcal{P}_i \cap \mathcal{P}_{0,n+1}} f_i\left(d^{ext}(p)\right).$$

Proof. Suppose that for all $p \in \mathcal{P}_i \cap \mathcal{P}_{0,n+1}$, $f_i\left(d^{\text{ext}}(p)\right) > \min_{d \in \mathcal{D}} f_i(d)$. Let d^* be the scenario such that $f_i(d^*) = \min_{d \in \mathcal{D}} f_i(d)$. Clearly, $d^* \neq d^{\text{ext}}(p)$, $\forall p \in \mathcal{P}_i \cap \mathcal{P}_{0,n+1}$. Let p' be one of the longest paths in $\mathcal{P}_i \cap \mathcal{P}_{0,n+1}$ for scenario d^*. The minimal float of i under scenario d^* is $l^*_{0,n+1}(d^*) - l_{p'}(d^*)$. Consider now scenario $d^{\text{ext}}(p')$, obtained by increasing to $d_j^{\max}$ all activities $j \in p'$ and decreasing to their minimum duration all other activities. Clearly, p' remains a longest path traversing i in scenario $d^{\text{ext}}(p')$ and its length strictly increases by $v = \sum_{j \in p'} (d_j^{\max} - d_j^*)$. Moreover, the length of any other path in $\mathcal{P}$ cannot be increased by more than v, including the longest one(s). So, the float of i cannot be increased by switching from scenario d^* to scenario $d^{\text{ext}}(p')$, which contradicts the hypothesis. □

It follows from Theorem 2 that the search for an optimal scenario can be replaced by the search for an optimal path *i.e.*, a path $p \in \mathcal{P}_i \cap \mathcal{P}_{0,n+1}$

that minimizes $f_i\left(d^{\text{ext}}(p)\right)$. Thus, Dubois *et al* [5] designed the so-called *path algorithm*, consisting in enumerating all paths $p \in \mathcal{P}_{0,n+1}$, and, for each of them, in computing $f_i\left(d^{\text{ext}}(p)\right)$, for all $i \in V$, through standard longest path computations. However, as experienced in [7], the number of paths becoming huge as the number of precedence constraints increases, the path algorithm fails in producing solutions in reasonable time for dense networks. For illustrating this drawback, let consider the network, displayed in Figure 1, which is decomposed into $\phi + 2$ levels, each level containing q activities, each activity of level $K \geq 1$ being a successor of any activity of level $K - 1$. Obviously, this graph has q^ϕ distinct paths from 0 to $n + 1$, so their enumeration becomes intractable as ϕ and q increase (even reasonably).

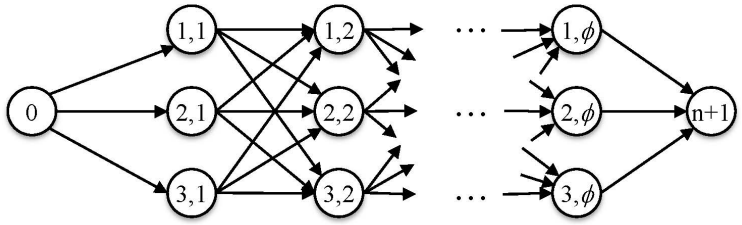

Figure 1: Intractable graph for the path algorithm with $q = 3$

3.3 Path structure in path-induced extreme scenarios

In this subsection, a structural analysis refining Theorem 2 is proposed, so as to hopefully find further search-space-reducing techniques. A corollary of Theorem 2 is first presented, which allows to exclude dominated paths. Let pay attention to the set $\hat{\mathcal{P}}_i \subseteq \mathcal{P}_i \cap \mathcal{P}_{0,n+1}$, defined as follows:

$$\hat{\mathcal{P}}_i = \{p \in \mathcal{P}_i \cap \mathcal{P}_{0,n+1} | l_p\left(d^{\text{ext}}(p)\right) = l^*_{0,i}\left(d^{\text{ext}}(p)\right) + l^*_{i,n+1}\left(d^{\text{ext}}(p)\right)\}.$$

$\hat{\mathcal{P}}_i$ is the subset of $\mathcal{P}_i$ such that each path of $\hat{\mathcal{P}}_i$ is also a longest path passing through i for its induced extreme scenario.

Corollary 1. *For each task i, there always exists a path-induced extreme scenario $d^{ext}(p)$, with $p \in \hat{\mathcal{P}}_i$ such that:*

$$\min_{d \in \mathcal{D}} f_i(d) = \min_{p \in \hat{\mathcal{P}}_i} f_i\left(d^{ext}(p)\right).$$

Proof. This property can easily be derived from the proof of Theorem 2 since path p', used in the argument, is also the longest path traversing i under scenario $d^{\text{ext}}(p')$. □

It follows that the solution space of the minimum float problem of an activity i can be further restricted to $\hat{\mathcal{P}}_i$. In the sequel of the text, any path of $\hat{\mathcal{P}}_i$ will be said *valid*.

Given a valid path p, computing the float of i requires the computation of both the length of the longest path $l^*_{0,n+1}(d^{\text{ext}}(p))$ and the one of p under scenario $d^{\text{ext}}(p)$ *i.e.*, $l_p\left(d^{\text{ext}}(p)\right)$, which equals $l^*_{0,i}\left(d^{\text{ext}}(p)\right) + l^*_{i,n+1}\left(d^{\text{ext}}(p)\right)$. Actually, these longest paths have a special structure, which makes the length computations easier.

Consider a path $p_{a\to b}$ from an activity a to an activity b. Let $g(p_{a\to b})$ denote the difference between two path lengths:

- the length $l^*_{a,b}\left(d^{\text{ext}}(a)\right)$ of longest paths from a to b under scenario $d^{\text{ext}}(a)$ (all activities are set to their minimal duration except a which is set to its maximal one),
- minus the length $l_{p_{a\to b}}(d^{\max})$ of $p_{a\to b}$ under scenario $d^{\max}$.

Since $l^*_{a,b}\left(d^{\text{ext}}(a)\right) = l^*_{a,b}(d^{\min}) + d^{\max}_a - d^{\min}_a$, we have:

$$g(p_{a\to b}) = l^*_{a,b}(d^{\min}) + d^{\max}_a - d^{\min}_a - l_{p_{a\to b}}(d^{\max}) \tag{1}$$

Lemma 1. *For any valid path $p \in \hat{\mathcal{P}}_i$, there is a subpath $p_{a\to b}$ of p such that $i \in p_{a\to b}$ and*

$$f_i\left(d^{ext}(p)\right) = g(p_{a\to b}).$$

Proof. Let p' be a longest path from 0 to $n+1$ in scenario $d^{\text{ext}}(p)$. As illustrated in Figure 2, at first glance, three divergence possibilities can be distinguished between p and p'.

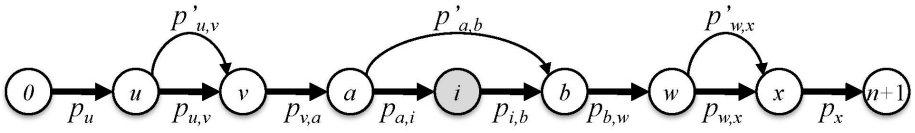

Figure 2: Impossible path structure

A first one considers a divergence of p' before activity i which converges back to p before i, as between u and v in Figure 2. A second symetrical possibility corresponds to the case where the divergence occurs after i, as between w and x in Figure 2. Nevertheless, from the Bellman principle, both of them contradict the longest path property since if p' is a longest path then p cannot be a longest path in $\mathcal{P}_i$), and vice-versa. Therefore, the only type of divergences that has to be considered, provided p and p' diverge, is the one that starts at a predecessor a of i along p, and ends at a successor b of i along p, as illustrated in Figure 3.

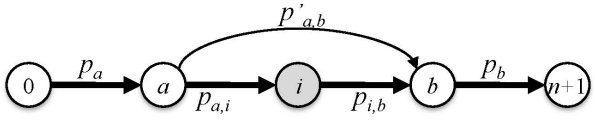

Figure 3: Structure of a valid path and of the longest path

Hence, the difference between p' and p lengths in scenario $d^{\text{ext}}(p)$ can be measured only between a and b, which yields precisely the $g(p_{a\to b})$ value. The case where p' and p are identical corresponds to a elementary sub-path reduced to i and the float is given by $g(p_{i\to i}) = 0$. □

Now Lemma 2 states that, considering an activity i and a valid path $p \in \hat{\mathcal{P}}_i$, the float f_i is given by the predecessor/successor pair (a, b) that maximizes g over $\hat{\mathcal{P}}_i$.

Lemma 2. *For any activity i and any valid path $p = (p_{0\to i}, p_{i\to n+1}) \in \hat{\mathcal{P}}_i$,*

$$f_i\left(d^{ext}(p)\right) = \max_{a\in p_{0\to i}, b\in p_{i\to n+1}} g(p_{a\to b}).$$

Proof. From Lemma 1, we know that it exists a pair of activities (a, b) of p which satisfies the previous equality (with possibly $a = b = i$).

Now for showing that $f_i\left(d^{\text{ext}}(p)\right) \geq g(p_{a\to b})$, for any pair (a, b) with $a \in p_{0\to i}$ and $b \in p_{i\to n+1}$, let rewrite $g(p_{a\to b})$ as follows:

$$\begin{aligned} g(p_{a\to b}) = \\ \left[l^*_{0,a}\left(d^{\text{ext}}(p)\right) + l^*_{a,b}\left(d^{\text{ext}}(a)\right) + l^*_{b,n+1}\left(d^{\text{ext}}(p)\right)\right] \\ - \left[l^*_{0,a}\left(d^{\text{ext}}(p)\right) + l_{p_{a\to b}}\left(d^{\text{ext}}(p)\right) + l^*_{b,n+1}\left(d^{\text{ext}}(p)\right)\right]. \end{aligned}$$

The first term can not be larger than the length of the longest path for scenario d^{ext} (as $d^{\text{ext}}(a) \leq d^{\text{ext}}(p)$). The second term is larger or equal to the total length of p in scenario $d^{\text{ext}}(p)$. Consequently, $g(p_{a\to b})$ can not be larger than the difference between the lengths of the longest path and p. □

Theorem 3 allows to obtain a new formulation of the minimum float problem.

Theorem 3. *For each task i,*

$$\min_{d\in\mathcal{D}} f_i(d) = \min_{p\in\hat{\mathcal{P}}_i} \max_{a\in p_{0\to i}, b\in p_{i\to n+1}} g(p_{a\to b})$$

Proof. This follows directly from Lemmas 1 and 2. □

3.4 Float lower bound given a valid partial path

Considering an activity i and a valid *partial* path $p_{x\to y}$ with $i \in p_{x\to y}$, a major consequence of Theorem 3 is the opportunity to obtain a new lower bound on the float f_i (see Figure 4 where $p_{x\to i} = (p_{x\to a}, p_{a\to i})$ and $p_{i\to y} = (p_{i\to b}, p_{b\to y})$). We highlight that $p_{x\to y}$ is said valid in the sense that $p_{x\to i}$ is the longest path in $d^{\text{ext}}(p_{x\to y})$ between x and i and $p_{i\to y}$ is the longest path in $d^{\text{ext}}(p_{x\to y})$ between i and y. Then, according to Theorem 3, if there exists a valid path $p_{0\to n+1}$ extending $p_{x\to y}$ towards activities 0 and $n+1$, it satisfies:

$$f_i\left(d^{\text{ext}}(p_{0\to n+1})\right) \geq \max_{a\in p_{x\to i}, b\in p_{i\to y}} g(p_{a\to b}).$$

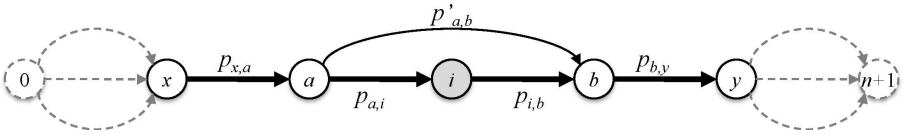

Figure 4: Partial path structure for float computation

4 Algorithms

4.1 Path Algorithm and improved variant

In this section, the Path algorithm proposed in [5] is described (see Algorithm 1). An improved variant of this algorithm is also proposed (see Algorithm 2). Note that for both algorithms a topological sort is run as a preprocessing for computing the longest paths $l^*_{0,i}(d^{\min})$ for all $i \in V$.

Algorithm 1 of Dubois *et al.* [5] visits all paths $p \in \mathcal{P}_{0,n+1}$ using a depth-first search strategy and, for scenario $d^{\text{ext}}(p)$, determines the earliest and latest starting times of all the activities in V, and new activity floats, using a standard PERT-CPM algorithm.

While Algorithm 2 is similar in the way the various $p \in \mathcal{P}_{0,n+1}$ are visited, it works differently in the way floats are updated. Indeed, it appears that the float of any activity having its duration set to $d^{\min}$ in scenario $d^{\text{ext}}(p)$ cannot be minimal, without contradicting Theorem 2. Therefore, only the floats of activities on p need to be updated with the upper bound $l^*_{0,n+1}\left(d^{\text{ext}}(p)\right) - l_p\left(d^{\text{ext}}(p)\right)$.

In the original version of the algorithm, for each path $p \in \mathcal{P}_{0,n+1}$, $2m$ operations at the most are required for computing all earliest and latest starting times and n float updating operations are performed. Our variant only requires m operations to compute the longest path (since the network is a DAG) and $|p|$ operations to update the floats of activities along p.

Algorithm 1 original_path($p_{0\to i}$)

if $i = n+1$ **then**
  compute all earliest starting times est in $d^{\text{ext}}(p_{0\to n+1})$;
  compute all latest starting times lst in $d^{\text{ext}}(p_{0\to n+1})$;
  for all $j \in V$ **do**
    $f_j := \min\{f_j, lst_j - est_j\}$;
  end for
else
  for all $j \in \Gamma(i)$ **do**
    original_path($(p_{0\to i}, j)$);
  end for
end if

Algorithm 2 improved_path($p_{0\to i}$)

if $i = n+1$ **then**
  compute $l^*_{0,n+1}\left(d^{\text{ext}}(p_{0\to n+1})\right)$;
  for all $j \in p_{0\to n+1}$ **do**
    $f_j := \min\{f_j, l^*_{0,n+1}\left(d^{\text{ext}}(p_{0\to n+1})\right) - l_{p_{0\to n+1}}(d^{\text{ext}}(p_{0\to n+1}))\}$;
  end for
else
  for all $j \in \Gamma(i)$ **do**
    improved_path($(p_{0\to i}, j)$);
  end for
end if

4.2 Specific branch-and-bound method

Corollary 1 and Theorem 3 allow to design an efficient branch-and-bound procedure for the computation of the minimum floats. In this procedure, the nodes of the search tree correspond to valid partial paths, related to a given activity i, and are stored inside a stack $\mathcal{Q}$ for depth-first search. Given a valid partial path $p_{x\to y}$ with $i \in p_{x\to y}$, the branching scheme consists in alternatively extending the path to the left or to the right, by considering either all the immediate predecessors of x, or all the immediate successors of y, discarding the non-valid path extensions (with respect to Corollary 1). Thus, considering a given path-extension direction δ, a node has always as many children as immediate valid path-extensions. Each node of the search tree also memorizes the direction δ ($\delta \in \{\text{left, right}\}$) to consider for the next path extension. A leaf of the search tree corresponds to a valid path $p \in \hat{\mathcal{P}}_i$ from 0 to $n+1$ and is evaluated by $f_i\left(d^{\text{ext}}(p)\right)$. Classically, a partial path $p_{x\to y}$ is deleted only if its current evaluation, $\max_{a\in p_{x\to i}, b\in p_{i\to y}} g(p_{a\to b})$, (see Theorem 3), is lower than the best float f_i already found.

Let us comment on Algorithm 3. Let remark first that the longest path values $l^*_{i,j}(d^{\min})$ are precomputed using the variant of Bellman-Ford's algorithm for DAGs. Computing the minimum float of all activities requires running the branch-and-bound procedure n times (see step 1). At the be-

ginning of each run (steps 2-3), f_i is set to ∞, stack $\mathcal{Q}$ only contains a single partial path $p_{x\to y} = i$, and the path-extension direction is set to left by default. The branch-and-bound procedure is implemented in steps 4-38. While stack $\mathcal{Q}$ is not empty, a partial path $p_{x\to y}$ is taken from the stack, with its path-extension direction δ (step 5). Preliminarily, with respect to Theorem 3, the activities (a, b) for which the longest path from x to y differs from $p_{x\to y}$ are updated (see step 6). Note that this can be done incrementally in linear time: if x (y) is the last activity toward which the path has been extended, only pairs (u, v) such that $u = x$ ($v = y$) and $v \in p_{i\to y}$ ($u \in p_{x\to i}$) are considered, respectively.

If the evaluation of $p_{x\to y}$ fails (*i.e.*, $g(p_{a\to b}) \geq f_i$), the path is deleted (see step 7). Otherwise, if $p_{x\to y} \in \mathcal{P}_{0,n+1}$, the new f_i value is memorized (see steps 8-9). If $p_{x\to y} \notin \mathcal{P}_{0,n+1}$, it is extended with respect to direction δ. Below, only the case δ=left is commented on (see steps 10-22), the case δ =right (see steps 23-36) being symmetrical.

All the immediate predecessors x' of x are first considered for possible path extension $(x', p_{x\to y})$ (see step 11). Step 12 checks that the path obtained by the concatenation of the longest path from 0 to x' and $p_{x'\to i}$ is longer than $l^*_{0,i}\left(d^{\text{ext}}(x', p_{x\to y})\right)$, otherwise it is not considered according to Lemma 1. Still in accordance with this Lemma, step 13 verifies that it does not exist any activity $u \in p_{x\to i}$ such that the path $(x', p_{x\to u})$ has a smaller length than the one of the longest path between x' and u. Once a new left-path-extension is made, the next extension direction is set to right unless $y = n + 1$ (see steps 14-19). Let us highlight that the evaluation of $l^*_{0,i}\left(d^{\text{ext}}(x', p_{x\to y})\right)$, used in step 12, can be done in linear time since this value can be memorized on each considered vertex i (it initially equals $l^*_{0,i}\left(d^{\min}\right)$), then updated at each path extension (to the value $\max\left[l^*_{0,i}\left(d^{\text{ext}}(x', p_{x\to y})\right), l^*_{0,x'}(d^{\min}) + l_{p_{x',i}}(d^{\max})\right]$ in case of a left path extension).

4.3 Mixed integer programming

In this section, two mixed integer programming (MIP) formulations are proposed, also exploiting Theorem 2 and the concept of a path-induced extreme scenario. The former corresponds to an activity independent formulation, while the latter is activity dependent. In both formulations, a binary variable x_i indicates for each activity i if it is located on the optimal path ($x_i = 1$) or not ($x_i = 0$). The formulations also involve earliest starting time continuous variables S_i, whose value is the length of the longest path from 0 to i in the scenario that minimizes the float.

Algorithm 3 branch-and-bound

for all $i \in V \setminus \{0, n+1\}$ **do**
  $f_i \leftarrow \infty$;
  push$(\mathcal{P}, (i, \text{left}))$;
  while $\mathcal{P} \neq \emptyset$ **do**
    $(p_{x\to y}, \delta) \leftarrow$pop$(\mathcal{P})$;
    $(a, b) \leftarrow \underset{\{u\in p_{x\to i}, v\in p_{i\to y}\}}{\text{argmax}} g(p_{u\to v})$;
    if $g(p_{a\to b}) < f_i$ **then**
      if $x = 0$ AND $y = n+1$ **then**
        $f_i \leftarrow g(p_{a\to b})$;
      else if $\delta =$left **then**
        for all $x' \in \Gamma^{-1}(x)$ **do**
          if $l^*_{0,x'}(d^{\max}) + d^{\max}_{x'} - d^{\min}_{x'} + l_{p_{x\to i}}(d^{\max}) \geq l^*_{0,i}\left(d^{\text{ext}}(x', p_{x\to y})\right)$ **then**
            if $d^{\max}_{x'} + l_{p_{x\to u}}(d^{\max}) \geq l^*_{x',u}(d^{\min}) + d^{\max}_{x'} - d^{\min}_{x'}$, $\forall u \in p_{x\to i}$ **then**
              if $y \neq n+1$ **then**
                $\delta \leftarrow$right;
              else
                $\delta \leftarrow$left;
              end if
              push$(\mathcal{P}, ((x', p_{x\to y}), \delta))$;
            end if
          end if
        end for
      else if $\delta =$right **then**
        for all $y' \in \Gamma(y)$ **do**
          if $l^*_{y',n+1}(d^{\max}) + l_{p_{i,y'}}(d^{\max}) \geq l^*_{i,n+1}\left(d^{\text{ext}}(p_{x\to y}, y')\right)$ **then**
            if $l_{p_{v\to y}}(d^{\max}) + d^{\max}_{y} \geq l^*_{v,y'}(d^{\min}) + d^{\max}_{v} - d^{\min}_{v}$, $\forall v \in p_{x\to i}$ **then**
              if $x \neq 0$ **then**
                $\delta \leftarrow$left;
              else
                $\delta \leftarrow$right;
              end if
              push$(\mathcal{P}, ((p_{x\to y}, y'), \delta))$;
            end if
          end if
        end for
      end if
    end if
  end while
end for

4.3.1 Activity-independent MIP

The activity-independent formulation is solved iteratively. At iteration k, a set U_k of activities with unknown floats is considered (at iteration 0, $U_0 = V \setminus \{0, n+1\}$). MIP Q($k$) finds the minimal float for a set $F_k \subset U_k$ of activities, with its related path-induced extreme scenario $d^{\text{ext}}(p)$. F_k is the set of activities with unknown floats belonging to the computed path p (*i.e.*, such that $x_i = 1$). F_k is not empty because of constraint (7). At the next iteration, U_{k+1} is set to $U_k \setminus F_k$. Due to the minimization objective, note that, if $i \in U_k \setminus F_k$ and if $j \in F_0 \cap F_1 \cap \cdots \cap F_k$, then $\min_{d \in \mathcal{D}} f_i(d) \geq \min_{d \in \mathcal{D}} f_j(d)$. It follows that at iteration 0, a path of activities having a null float will be issued. Then, floats are issued in non-decreasing order along the iterations, until all activities get their minimal float computed. This mechanism ensures that at most n iterations will be needed.

Q(k) :

$$\min z(k) = S_{n+1} - \sum_i d_i^{max} x_i \tag{2}$$

subject to

$$z(k) \geq z(k-1), \tag{3}$$

$$S_j \geq S_i + d_i^{min} + (d_i^{max} - d_i^{min}) x_i \quad \forall (i,j) \in A, \tag{4}$$

$$x_i \leq \sum_{j \in \Gamma_i^{-1}} x_j \leq 1 \quad \forall i \neq 0, n+1, \tag{5}$$

$$x_i \leq \sum_{j \in \Gamma_i} x_j \leq 1 \quad \forall i \neq 0, n+1, \tag{6}$$

$$x_0 = x_{n+1} = 1, \tag{7}$$

$$\sum_{i \in U_k} x_i \geq 1, \tag{8}$$

$$x_i \in \{0,1\} \qquad \forall i \in V. \tag{9}$$

Objective function (2) measures the difference between the longest path in G, given by S_{n+1}, and the length of path p represented by x_i values, on which all activities are set to their maximal duration. Constraint (3) states that the float is non-decreasing between two successive iterations (this constraint is redundant). Constraints (4) implement precedence relations between activities $(i,j) \in A$, namely $S_j - S_i \geq d_i^{\max}$ if i is on p (*i.e.*, $x_i = 1$), $S_j - S_i \geq d_i^{\min}$ otherwise (*i.e.*, $x_i = 0$). Constraints (5-7) enforce the x_i values to define a path from 0 to $n+1$. An activity can be located on p only if exactly one of its predecessors (successors) belongs also to p, respectively. Constraints (7) state that 0 and $n+1$ are the extremities of path p. Constraint (8) enforces the selected path to traverse at least one activity with unknown float (set U_k).

4.3.2 Activity-dependent MIP

The activity-dependent MIP P(k) below is just a particular case of Q(k), where at each iteration k, the set U_k is reduced to the single activity k. Hence, for each activity k, P(k) minimizes float f_k. To obtain all minimal floats, exactly n iterations are needed. Giving this definition, the MIP can be rewritten as follows. The only difference with Q(k) lies in constraint (15) stating that k must be located on path p.

P(k) :

$$\min f_k = S_{n+1} - \sum_i d_i^{max} x_i \tag{10}$$

subject to

$$S_j \geq S_i + d_i^{min} + (d_i^{max} - d_i^{min})x_i \quad \forall (i,j) \in A, \tag{11}$$

$$x_i \leq \sum_{j \in \Gamma_i^{-1}} x_j \leq 1 \quad \forall i \neq 0, n+1, \tag{12}$$

$$x_i \leq \sum_{j \in \Gamma_i} x_j \leq 1 \quad \forall i \neq 0, n+1, \tag{13}$$

$$x_0 = x_{n+1} = 1, \tag{14}$$

$$x_k = 1, \tag{15}$$

$$x_i \in \{0,1\} \qquad \forall i \in V. \tag{16}$$

Note that computing all minimum floats with Q(k) may need less iterations than with P(k). However, the search space for P(k) is reduced since only ancestors and descendants of k may belong to p. Section 5 experimentally evaluates the two formulations together, in comparison with the proposed branch-and-bound method and the path algorithms.

5 Computational experiments

This section presents computational results to evaluating the efficiencies of the Path algorithm (Path) of Dubois *et al.* [5] and Fortin *et al.* [7] (original_path), its improved variant (improved_path), the activity dependent (MIPd) and independent (MIPi) mixed-integer programs and our specific branch-and-bound procedure (BB). All the experiments were performed on the same Intel 2.6 GHz processor having 3 Go RAM running on a Linux XUbuntu operating system. The MIP approaches were implemented using Cplex 12.0.

5.1 Randomly generated problem instances

Two sets of robust-PERT problem instances were used. The first set, also considered in [7], is built up from the 120 problem instances of the PSPLIB,

that was originally designed for comparing solving approaches for the Resource-Constrained Project Scheduling Problems (RCPSP). Each problem instance is defined by an activity network, each activity i being characterized by a deterministic duration d_i and a resource consumption vector. As in [7], for building up robust-PERT instances, resource characteristics were omitted and the interval uncertainty of the activity duration were set to $[d_i, d_i \times 1.2]$. These benchmarks are named $PSPLIB-$instances.

As in [7], using RanGen1 software ([4]), a set of nine series of 100 activities networks called RG-instances was generated, each of them being identified by its *order strength* (OS). The order strength measures the density of precedence constraints in a project network. It is computed as the size of the transitive closure of the network divided by the size of the complete (undirected) underlying graph. Therefore, networks with the same number of activities and a higher order strength tend to have more paths. An order strength of 0 means a full parallelism while an order strength of 1 gives a total order. RanGen1 was parametrized for OS ranging from 0.1 to 0.9 by steps of 0.1. For determining the uncertainty interval $[d_i^{\min}, d_i^{\max}]$ of each activity, the benchmarking procedure described in [7] was used to generate the RG-instances. It suggests to randomly setting a duration $d_i^{\min} \in [10, 50]$, then to set $d_1^{\max}$ to $d_i^{\min} \times 1.2$ (as done with the $PSPLIB-$instances).

Nevertheless, we point out that this interval generation scheme introduces a bias since it yields the property that for any path p and p', $l_p(d^{\min}) > l_{p'}(d^{\min}) \implies l_p(d^{\max}) > l_{p'}(d^{\max})$. Consequently, we also use a more general interval generation scheme for an activity j, using two parameters α and β, such that

$$\left[d_j^{\min} \in \{10, \alpha 50\}, d_j^{\max} \in \left\{d_j^{\min}, (1+\beta) d_j^{\min}\right\}\right].$$

Parameters α and β have been picked in the set $\{0.3, 0.6, 0.9\}$, the possible combinations defining, for each order strength, nine sets of 100-activities instances denoted $\alpha - \beta-$instances.

5.2 Algorithm comparison

We first present a comparison between the two path algorithms, the original and the improved ones on the $PSPLIB-$ and $RG-$ instances. The computing times (in seconds) compiled in Table 1 confirm the theoretical dominance of our variant of the Path algorithm. Computing times are divided by almost 2 on every problem instance. Furthermore, for the instances with $OS = 0.9$, the improved variant is able to obtain solutions in less than 2 hours while the original variant does not. For the remainder of the tests, only the improved variant of the path algorithm has been kept.

The two MIP formulations are compared in Table 2 on the $PSPLIB-$ and $RG-$instances. The activity-dependent MIP runs faster on all tested instances. On the most difficult instances (with $0S = 0.8$), MIPd needs

problem set	average time sec.		maximum time	
	original	improved	original	improved
$PSPLIB$				
$OS \in [0.1, 0.5]$	0.01	< 0.005	0.03	0.01
RG				
$OS = 0.1$	0.00	0.00	0.01	0.01
$OS = 0.2$	0.01	0.00	0.02	0.01
$OS = 0.3$	0.03	0.02	0.04	0.02
$OS = 0.4$	0.07	0.04	0.11	0.05
$OS = 0.5$	0.21	0.12	0.32	0.19
$OS = 0.6$	0.72	0.43	1.02	0.62
$OS = 0.7$	4.02	2.50	6.03	3.79
$OS = 0.8$	52.71	33.96	90.98	55.99
$OS = 0.9$	5320.00	2843.00	13332.00	6726.00

Table 1: Comparison of the two path algorithms

23.12 seconds on average and 98.80 seconds in the worst case to compute all the floats, whilst MIPi needs 197.49 on average and 671.45 in the worst case. The high number of iterations of MIPi – at least 50 – is the main reason of the MIPd dominance, since each iteration of MIPi is expected to be harder to solve. In the following, MIPi results are discarded and only MIPd is compared to other algorithms.

Table 3 reports, for each algorithm (improved_path, MIPd, BB), the average and maximal computing times, according to the type of instances. CPU times were limited to one hour. The $\alpha - \beta$-instances are grouped with respect to their $OS$and β values (since α parameter does not seem significant for any of the algorithms).

The MIPd algorithm is dominated by the Path algorithm on all problem sets but the $0S = 0.9$ series. Actually, the Path algorithm fails to compute minimal floats after one hour. Results obtained on the $PSPLIB-$instances reveal that the MIPd suffers a lot from the increase of the number of activities. The BB procedure considerably outperforms the other algorithms , its dominance increasing with the OS value.

It also appears that $RG-$instances can be quickly solved by the MIPd algorithm, and even more quickly by the BB algorithm. The MIPd computational effort increases as the OS and β values get larger since, at the worst case, 78.14, 590.17 and 1776.73 seconds are required to solve $\alpha - \beta-$instances with $\beta = 0.3$, 0.6 and 0.9, respectively. MIPd has similar results on $PSPLIB-$, $RG-$ and $\alpha - \beta = 0.3-$instances where the ratio between d^{max} and d^{min} is close to 1.2.

I DON'T UNDERSTAND THIS SENTENCE !!!!: In those sets of instances, the dominance between paths can easily overpass scenarios because

of their similar duration increase from d^{min} to d^{max}. We assume that this property can strengthen the branch-and-bound procedure in the MIP solver.

The $\alpha-\beta-$instances are derived from the $RG-$instances and their transformation keeps the underlying network unchanged. The Path algorithm therefore has essentially the same performance on every duration interval pattern of each instance. Regarding the number of nodes explored, it turns out that the results are similar for the BB algorithm on every duration interval pattern of each instance. For those reasons, the results concerning different problem sets are merged in the Table 4, which gives the average and maximum number of paths and the number of nodes explored by the Path and BB algorithms, respectively. As expected, the Path algorithm is able to quickly solve instances with low-to-medium order strength (up to 0.7). Considering the order strength definition and the Table 4, it is clear that a high OS value induced a high number of paths, which is the critical determinant of the effectiveness of the Path algorithm. We remark that the maximum number of nodes explored in BB is not so sensitive to OS, smoothly varying from 1331 to 82832. It means that the dominance rules implemented in BB are strong enough to prune many scenarios. Computing times of BB slightly increase with OS – it runs from 0.01 to 0.08 seconds – since, in the worst case, all the paths of the network are visited.

Though the BB procedure seems to be sensitive to the OS value, parameters α and β do not seem to have any influence on its performance. Also remark that, in our implementation of BB, there is no specific rule for guiding the selection of activities (the list $\mathcal{P}$ is unsorted). Designing such rules could speed-up the procedure by exploiting the regularity of instances with low α and β.

6 Conclusion

In this paper, a very fast method for computing the minimum float of activities in activity-on-node networks, under interval duration uncertainty, has been proposed. Thanks to newly established structural properties of optimal solutions and lower bounds on partial solutions, the method consistently outperforms the methods it has been compared with: new mixed integer programming formulations solved by a commercial solver and the path enumeration algorithm of Dubois *et al.* [5] and Fortin *et al.* [7].

References

[1] F. Ballestín. When it is worthwhile to work with the stochastic rcpsp? *Journal of Scheduling*, 10(3):153–166, 2007.

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

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

[4] E. Demeulemeester, M. Vanhoucke, and W. Herroelen. Rangen: a random network generator for activity-on-the-node networks. *Journal of Scheduling*, 6(1):17–38, 2003.

[5] 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.

[6] 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.

[7] J. Fortin, P. Zieliński, D. Dubois, and H. Fargier. Criticality analysis of activity networks under interval uncertainty. *Journal of Scheduling*, 2010. DOI: 10.1007/s10951-010-0163-3.

[8] W. Herroelen and R. Leus. Project scheduling under uncertainty: Survey and research potentials. *European Journal of Operational Research*, 165(2):289–306, 2005.

[9] Adam Kasperski. *Discrete Optimization with Interval Data*. Springer, 2008.

[10] P. Kouvelis and G. Yu. *Robust discrete optimization and its applications*. Springer, 1997.

[11] R. H. Möhring, F. J. Radermacher, and G. Weiss. Stochastic scheduling problems i – general strategies. *Mathematical Methods of Operations Research*, 28(7):193–260, 1984.

problem set	average time sec.		maximum time		average iterations	
	MIPi	MIPd	MIPi	MIPd	MIPi	MIPd
$PSPLIB$						
$OS \in [0.1, 0.5]$	22.34	4.55	130.84	23.02	50	120
RG						
$OS = 0.1$	4.65	0.25	8.89	0.33	83	100
$OS = 0.2$	12.28	0.54	17.93	0.69	81	100
$OS = 0.3$	19.37	1.08	26.85	1.61	80	100
$OS = 0.4$	28.21	2.29	43.52	3.40	77	100
$OS = 0.5$	44.62	4.99	110.74	7.71	75	100
$OS = 0.6$	70.42	9.22	169.90	16.51	73	100
$OS = 0.7$	140.19	14.95	255.91	21.13	70	100
$OS = 0.8$	197.49	23.12	671.45	98.80	66	100
$OS = 0.9$	99.52	17.43	430.94	111.45	58	100

Table 2: Comparison of MIP formulations

problem set	average time sec.			maximum time		
	Path	MIPd	BB	Path	MIPd	BB
PSPLIB						
$OS \in [0.1.0.5]$	< 0.005	4.55	< 0.005	0.01	23.02	0.01
RG						
$OS = 0.1$	< 0.005	0.25	< 0.005	0.01	0.33	< 0.005
$OS = 0.2$	< 0.005	0.54	< 0.005	0.01	0.69	< 0.005
$OS = 0.3$	0.02	1.08	< 0.005	0.02	1.61	< 0.005
$OS = 0.4$	0.04	2.29	< 0.005	0.05	3.4	< 0.005
$OS = 0.5$	0.12	4.99	< 0.005	0.19	7.71	< 0.005
$OS = 0.6$	0.43	9.22	< 0.005	0.62	16.51	< 0.005
$OS = 0.7$	2.5	14.95	< 0.005	3.79	21.13	0.01
$OS = 0.8$	33.96	23.12	< 0.005	55.99	98.8	0.01
$OS = 0.9$	> 1800	17.43	0.01	> 1800	111.45	0.02
$\alpha - \beta = 0.3$						
$OS = 0.1$	< 0.005	0.24	< 0.005	0.01	0.33	< 0.005
$OS = 0.2$	< 0.005	0.51	< 0.005	0.01	0.67	< 0.005
$OS = 0.3$	0.02	1.03	< 0.005	0.02	1.58	0.01
$OS = 0.4$	0.04	2.17	< 0.005	0.06	3.6	0.01
$OS = 0.5$	0.12	4.86	< 0.005	0.22	8.64	0.01
$OS = 0.6$	0.44	9.04	< 0.005	0.62	14.07	0.01
$OS = 0.7$	2.51	15.65	0.01	3.83	24.78	0.02
$OS = 0.8$	34.37	24.62	0.01	58.01	73.29	0.03
$OS = 0.9$	> 1800	17.12	0.01	> 1800	78.14	0.04
$\alpha - \beta = 0.6$						
$OS = 0.1$	< 0.005	0.26	< 0.005	0.01	0.35	< 0.005
$OS = 0.2$	< 0.005	0.56	< 0.005	0.01	0.72	0.01
$OS = 0.3$	0.02	1.15	< 0.005	0.02	1.71	< 0.005
$OS = 0.4$	0.04	2.37	< 0.005	0.05	3.67	< 0.005
$OS = 0.5$	0.12	5.34	< 0.005	0.17	8.08	0.01
$OS = 0.6$	0.43	10.71	< 0.005	0.65	17.49	0.01
$OS = 0.7$	2.5	23.43	0.01	4.03	52.17	0.01
$OS = 0.8$	34.35	63.89	0.01	56.45	157.07	0.02
$OS = 0.9$	> 1800	94.61	0.01	> 1800	590.17	0.03
$\alpha - \beta = 0.9$						
$OS = 0.1$	< 0.005	0.27	< 0.005	< 0.005	0.34	0.01
$OS = 0.2$	< 0.005	0.6	< 0.005	0.01	0.82	< 0.005
$OS = 0.3$	0.02	1.25	< 0.005	0.02	1.85	< 0.005
$OS = 0.4$	0.04	2.55	< 0.005	0.05	3.82	0.01
$OS = 0.5$	0.12	5.77	< 0.005	0.22	9.49	0.01
$OS = 0.6$	0.43	12.12	< 0.005	0.66	23.89	0.01
$OS = 0.7$	2.52	34.76	0.01	3.87	84.68	0.02
$OS = 0.8$	34.38	124.22	0.01	57.71	267.88	0.03
$OS = 0.9$	> 1800	338.98	0.01	> 1800	1776.73	0.08

Table 3: Computing times comparison on all instances

problem set order strength	Paths explored by Path avg.	 max.	Nodes explored by BB avg.	 max.
$PSPLIB$				
$OS \in [0.1, 0.5]$	326	1277	2755	5715
RG and $\alpha - \beta$				
$OS = 0.1$	404	438	839	1331
$OS = 0.2$	804	890	1207	2249
$OS = 0.3$	1644	1929	1667	3498
$OS = 0.4$	3793	4783	2305	5118
$OS = 0.5$	10527	14208	3209	8641
$OS = 0.6$	38024	50672	4567	10398
$OS = 0.7$	228018	346082	6838	17988
$OS = 0.8$	3325121	5595599	10176	34572
$OS = 0.9$	298930149	707330531	12491	82832

Table 4: Number of paths and nodes explored by Path and BB algorithms, respectively