Recoverable Robust Timetables: An Algorithmic Approach on Trees

Gianlorenzo d'Angelo, Gabriele Di Stefano, Alfredo Navarra, Cristina Pinotti

Recoverable Robust Timetables: an Algorithmic Approach on Trees

Gianlorenzo D'Angelo, Gabriele Di Stefano, Alfredo Navarra, and Cristina M. Pinotti

Abstract—In the context of scheduling and timetabling, we study a challenging combinatorial problem which is very interesting for both practical and theoretical points of view. The motivation behind it is to cope with scheduled activities which might be subject to unavoidable disruptions, such as delays, occurring during the operational phase. The idea is to preventively plan some extra time for the scheduled activities in order to be "prepared" if a delay occurs, and absorb it without the necessity of re-scheduling all the activities from scratch. This realizes the concept of designing *robust timetables*. During the planning phase, one should also consider recovery features that might be applied at runtime if disruptions occur. This leads to the concept of *recoverable robust timetables*. In this new concept, it is assumed that recovery capabilities are given as input along with the possible disruptions that must be considered. The main objective is the minimization of the overall needed time. The quality of a robust timetable is measured by the *price of robustness*, i.e. the ratio between the cost of the robust timetable and that of a non-robust optimal timetable. We show that finding an optimal solution for this problem is NP-hard even though the topology of the network, which models dependencies among activities, is restricted to trees. However, we manage to design a pseudo-polynomial time algorithm based on dynamic programming and apply it on both random networks and real case scenarios provided by Italian railways. We evaluate the effect of robustness on the scheduling of the activities and provide the price of robustness with respect to different scenarios. We experimentally show the practical effectiveness and efficiency of the proposed algorithm.

Index Terms—Timetable, Scheduling Activities, Robustness, Price of Ribusteness, Combinatorial Optimization, Dynamic Programming

1 Introduction

In many real-world applications, the design of a solution is divided in two main phases: a *strategic planning* phase and an *operational planning* phase. The two planning phases differ in the time in which they are applied. The strategic planning phase aims to plan how to optimize the use of the available resources according to some objective function *before the system starts operating*. The operational planning phase aims to have immediate reaction to disturbing events that can occur *when the system is running*. In general, the objectives of strategic and operational planning might be in conflict with each other. As disturbing events are unavoidable in large and complex systems, it is fundamental to understand the interaction between the objectives of the two phases. An example of real-world systems, where this interaction is important, is the *timetable planning* in railway systems. It arises in the strategic planning phase and requires to compute a timetable for passenger trains that determines minimal passenger waiting times. However, many disturbing events might occur during the operational phase, and they might completely change the scheduled activities. The main effect of the disturbing events is the arising of delays. The conflicting objectives of strategic against operational planning are evident in timetable optimization. In fact, a train schedule that lets trains sit in stations for some time will not suffer from small delays of arriving trains, because delayed passengers can still catch potential connecting trains. On the other hand, large delays can cause passengers to lose trains and hence imply extra traveling time. The problem of deciding when to guarantee connections from a delayed train to a connecting train is known as *delay management* (see [3], [9], [11], [12], [15], [16]). Despite its natural formalization, the problem turns out to be very complicated to be optimally solved. In fact, it is NP-hard in the general case, while it is polynomial in some particular cases (see [3], [4], [11], [12], [15], [16]).

To cope with the management of delays, we follow the recent *recoverable robustness* approach provided in [2], [5] and [14], continuing the recent studying in robust optimization. Our aim is the design of timetables in the strategic planning phase in order to be "prepared" to react against possible delays. If a delay occurs, the designed timetable should guarantee to recover the scheduled events by means of allowed operations represented by given recovery algorithms. Events and dependencies among events are modeled by means of an *event activity network* (see [3], [4], [16], [17]). This is a directed graph where the nodes represent events (e.g., arrival or departure of trains) and arcs represent activities occurring between events (e.g., waiting in a train, driving between stations or changing to another train). We assume that

- *G. D'Angelo and G. Di Stefano are with the Department of Electrical and Information Engineering, University of L'Aquila, L'Aquila, Italy. Email:{gianlorenzo.dangelo, gabriele.distefano}@univaq.it*
- *A. Navarra and C. M. Pinotti are with the Department of Mathematics and Computer Science, University of Perugia, Perugia, Italy. Email:{navarra, pinotti}@dmi.unipg.it*

This work was partially supported by the Future and Emerging Technologies Unit of EC (IST priority - 6th FP), under contract no. FP6-021235-2 (project ARRIVAL).
Preliminary results contained in this paper appeared in [6], [7], [8].

only one delay of at most α time might occur at a generic activity of the scheduled event activity network. An activity may absorb the delay if it is associated with a so called *slack time*. A slack time assigned to an activity represents some extra available time that can be used to absorb limited delays. Clearly, if we associate a slack time of at least α to each activity, every delay can be locally absorbed. However, this approach is not practical as the overall duration time of the scheduled events would increase too much. We plan timetables able to absorb the possible occurring delay within a fixed amount of events, Δ. This means that, if a delay occurs, it is not required that the delay is immediately absorbed (unless $\Delta = 0$) but it can propagate to a limited number of activities in the network. Namely, the propagation might involve at most Δ events. The objective function is then to minimize the total time required by the activities in order to serve all the scheduled events and to be robust with respect to one possible delay. The challenging combinatorial problem arising by those restrictions is of its own interest. We restrict our attention to event activity networks whose topology is a tree. In fact, in [3], the authors show that the described problem is NP-hard when the event activity network topology is a directed acyclic graph, and they provide approximation algorithms which cope with the case of $\Delta = 0$. In [4], these algorithms have been extended to $\Delta \geq 0$.

In this paper, we study event activity networks which have a tree topology. Surprisingly, the described problem turns out to be NP-hard even in this restricted case. For trees, we present an algorithm that solves the problem, when $\Delta \geq 1$, in $O(\Delta^2 n)$ time and $O(\Delta n)$ space, where n is the number of events in the input event activity network. Whereas, $O(n)$ time and $O(n)$ space are required to solve the problem when $\Delta = 0$. The result implies that the problem can be solved in pseudo-polynomial time. In fact, the parameter Δ can be clearly provided using $\lceil \log \Delta \rceil$ bits. Moreover, since we prove that the size of some input event activity network instances, for which the problem remains NP-hard, can be encoded by means of $O(\log n)$ bits, the proposed algorithm is also pseudo-polynomial in n. The algorithm exploits the tree topology in order to choose which arc must be associated with some slack time. On trees, intuitively, we prove that the choice to carefully postpone the assignment of a slack time to descendent activities as much as possible leads to cheapest solutions. Another interesting property for tree topologies when $\Delta \geq 1$ shows that there is always an optimal solution without two consecutive activities both associated with a slack time. We also implement the algorithm and test its performances on both real case and random scenarios. Based on data provided by Trenitalia [18], we make use of the devised algorithm in order to obtain robust timetables with respect to the possibility of an occurring delay of duration α. We show and discuss the interesting obtained results about the applicability and the low costs in terms of slack times needed for making robust the considered timetables with respect to different scenarios. The small execution time elapsed by the algorithm on real-world instances revels its effective applicability. Moreover, experiments on random instances show that the good behavior of the algorithm does not depend on the particular structure of the input tree arising from the real-world instances.

1.1 Outline

The next section describes the model used to transform an optimization problem into a recoverable robust problem as shown in [5]. Section 3 presents the timetabling problem and its recoverable robust version. Section 4 is devoted to the study of the computational complexity required for solving the proposed recoverable robust timetabling. Section 5 presents the pseudo-polynomial time algorithm and provides its correctness. Section 6 is devoted to the experimental studies obtained by applying the algorithm to real-world and random instances. Finally, Section 7 provides conclusive remarks and useful outcomes for further investigation on the studied problem.

2 Recoverable Robustness Model

In this section, we summarize the model of recoverable robustness given in [5]. Such a model describes how an optimization problem P can be turned into a *robustness problem* $\mathcal{P}$. Hence, concepts like *robust solution*, *robust algorithm* for $\mathcal{P}$ and *price of robustness* are defined. In the remainder, an optimization problem P is characterized by the following parameters. A set I of instances of P; a function F that associates to any instance $i \in I$ the set of all feasible solutions for i; and an objective function f: $S \to \mathbb{R}$ where $S = \bigcup_{i \in I} F(i)$ is the set of all feasible solutions for P. Without loss of generality, from now on we consider minimization problems. Additional concepts to introduce robustness requirements for a minimization problem P are needed:

- $M : I \to 2^I$ – a *modification* function for instances of P. Let $i \in I$ be the considered input to the problem P. A *disruption* is meant as a modification to the input i. Hence, $M(i)$ represents the set of disruptions of the input of P that can be obtained by applying all possible modifications to i.
- $\mathbb{A}$ – a class of *recovery algorithms* for P. Algorithms in $\mathbb{A}$ represent the capability of recovering against disruptions. An element $A_{rec} \in \mathbb{A}$ works as follows: given $(i, \pi) \in I \times S$, an instance/solution pair for P, and $j \in M(i)$, a disruption of the current instance i, then $A_{rec}(i, \pi, j) = \pi'$, where $\pi' \in F(j)$ represents the recovered solution for P.

Definition 2.1: A *recoverable robustness problem* $\mathcal{P}$ is defined by the triple $(P, M, \mathbb{A})$. All the recoverable robustness problems form the class RRP.

Definition 2.2: Let $\mathcal{P} = (P, M, \mathbb{A}) \in$ RRP. Given an instance $i \in I$ for P, an element $\pi \in F(i)$ is a *feasible*

solution for i with respect to $\mathcal{P}$ if and only if the following relationship holds:

$$\exists A_{rec} \in \mathbb{A} : \forall j \in M(i),\ A_{rec}(i, \pi, j) \in F(j).$$

In other words, $\pi \in F(i)$ is feasible for i with respect to $\mathcal{P}$ if it can be *recovered* by applying some algorithm $A_{rec} \in \mathbb{A}$ for each possible disruption $j \in M(i)$. The solution π is called a *robust solution* for i with respect to problem P. The quality of a robust solution is measured by the *price of robustness*.

Definition 2.3: Let $\mathcal{P} = (P, M, \mathbb{A}) \in$ RRP. A *robust algorithm* for $\mathcal{P}$ is an algorithm A_{rob} such that, for each $i \in I$, $A_{rob}(i)$ is a robust solution for i with respect to P.

Definition 2.4: Let $\mathcal{P} \in$ RRP and let A_{rob} be a robust algorithm for $\mathcal{P}$. The *price of robustness of* A_{rob} is:

$$P_{rob}(\mathcal{P}, A_{rob}) = \max_{i \in I} \left\{ \frac{f(A_{rob}(i))}{\min\{f(\pi) : \pi \in F(i)\}} \right\}.$$

The *price of robustness of* $\mathcal{P}$ is:

$$P_{rob}(\mathcal{P}) = \min\{P_{rob}(\mathcal{P}, A_{rob}) : A_{rob} \text{ is robust for } \mathcal{P}\}.$$

A_{rob} is $\mathcal{P}$*-optimal* if $P_{rob}(\mathcal{P}, A_{rob}) = P_{rob}(\mathcal{P})$. A robust solution π for $i \in I$ is $\mathcal{P}$-optimal if

$$f(\pi) = \min\{f(\pi') \ :\ \pi' \text{ is feasible for } i \text{ w.r.t. } \mathcal{P}\}.$$

3 Robust Timetabling Problem

In this section, we turn a particular timetable problem (TT) into a recoverable robustness problem, the *Robust Timetabling* problem ($\mathcal{RTT}$).

Given a Directed Acyclic Graph (DAG) $G = (V, A)$, where the nodes represent events and the arcs represent the activities[1], the *timetabling problem* consists in assigning a time to each event in such a way that all the constraints provided by the set of activities are respected. Specifically, given a function $L : A \to \mathbb{N}$ that assigns the minimal duration time to each activity, a solution $\pi \in \mathbb{R}_{\geq 0}^{|V|}$ for the timetable problem on G is found by assigning a time $\pi(u)$ to each event $u \in V$ such that $\pi(v) - \pi(u) \geq L(a)$ for all $a = (u, v) \in A$.

Given a function $w : V \to \mathbb{R}_{\geq 0}$ that assigns a weight to each event, an optimal solution for the timetabling problem minimizes the total weighted time for all events. Formally, the *timetabling problem* TT is defined as follows.

TT	
GIVEN:	A DAG $G = (V, A)$, functions $L : A \to \mathbb{N}$ and $w : V \to \mathbb{R}_{\geq 0}$.
PROB.:	Find a function $\pi : V \to \mathbb{R}_{\geq 0}$ such that $\pi(v) - \pi(u) \geq L(a)$ for all $a = (u, v) \in A$ and $f(\pi) = \sum_{v \in V} w(v)\pi(v)$ is minimal.

Then, an instance i of TT is specified by a triple (G, L, w), where G is a DAG, L associates a minimal duration time to each activity, and w associates a weight to each event. The set of feasible solutions for i is: $F(i) = \{\pi : \pi(u) \in \mathbb{R}_{\geq 0},\ \forall u \in V \text{ and } \pi(v) - \pi(u) \geq L(a),\ \forall a = (u, v) \in A\}$.

A feasible solution for TT may induce a positive *slack time* $s_\pi(a) = \pi(v) - \pi(u) - L(a)$ for each $a \in A$. That is, the planned duration $\pi(v) - \pi(u)$ of an activity $a = (u, v)$ is greater than the minimal duration time $L(a)$.

When in TT the DAG is an out-tree $T = (V, A)$, any feasible solution satisfies, for each $v \in V$, $\pi(v) \geq \pi(r) + \sum_{a \in P(r,v)} L(a)$, where r is the root of T and $P(r, v)$ the directed path from r to v in T. Moreover, without loss of generality, we can focus our attention only on instances of TT with $L(a) = 1\ \forall a \in A$. Indeed, as proved below, the cost $f(\pi)$ of a feasible solution π for an instance of TT with an arbitrary function L easily derives from the cost $f(\pi')$ of a feasible solution π' for the same instance of TT with $L'(a) = 1$, $\forall a \in A$.

Lemma 3.1: Consider an out-tree $T = (V, A)$, rooted at r, and an instance $i = (T, L, w)$ of TT. Let $i' = (T, L', w)$ be an instance such that $L'(a) = 1$, $\forall a \in A$. For any feasible solution π for i, there exists a feasible solution π' for i' such that

$$f(\pi) = f(\pi') + \sum_{v \in V} \sum_{a \in P(r,v)} w(v)(L(a) - 1).$$

Proof: Any feasible solution π of i is in the form:

$$\pi(v) = \pi(r) + \sum_{a \in P(r,v)} (L(a) + s_\pi(a)),\ v \in V.$$

We define π' as $\pi'(r) = \pi(r)$ and

$$\pi'(v) = \pi'(r) + \sum_{a \in P(r,v)} (1 + s_\pi(a)).$$

Note that, π' is feasible for i'. The values of the objective functions of π and π' are

$$f(\pi) = \pi(r)w(r) + \sum_{v \in V} \pi(v)w(v) =$$

$$= \pi(r)w(r) + \sum_{v \in V} \sum_{a \in P(r,v)} (L(a) + s_\pi(a))w(v)$$

and

$$f(\pi') = \pi'(r)w(r) + \sum_{v \in V} \pi'(v)w(v) =$$

$$= \pi(r)w(r) + \sum_{v \in V} \sum_{a \in P(r,v)} (1 + s_\pi(a))w(v),$$

respectively. Hence,

$$f(\pi) = f(\pi') + \sum_{v \in V} \sum_{a \in P(r,v)} w(v)(L(a) - 1).$$

□

Problem TT can be solved in linear time by assigning the minimal possible time to each event (i.e. by using the Critical Path Method [13]). However, such a solution cannot always cope with possible delays occurring at running time to the activities. Recovery (on-line)

1. Indeed G is the so called event activity network described in the introduction.

strategies might be necessary. For this reason, let now transform TT into a recoverable robustness problem $\mathcal{RTT} = (TT, M, \mathbb{A})$, according to Section 2. Given an instance $i = (G, L, w)$ for TT and a constant $\alpha \in \mathbb{N}$, we limit the modifications on i by admitting a single delay of at most α time. We model it as an increase on the minimal duration time of the delayed activity. Formally, $M(i)$ is defined as follows: $M(i) = \{(G, L', w) : \exists\, \bar{a} \in A : L(\bar{a}) \leq L'(\bar{a}) \leq L(\bar{a}) + \alpha,\ L'(a) = L(a)\ \forall a \neq \bar{a}\}$.

We define the class of recovery algorithms $\mathbb{A}$ for TT by introducing the concept of *events affected by one delay*.

Definition 3.2: Given a DAG $G = (V, A)$, a function $s : A \to \mathbb{R}_{\geq 0}$, and a number $\alpha \in \mathbb{R}_{\geq 0}$, a node x is α-*affected* by $a = (u, v) \in A$ (or a α-*affects* x) if there exists a path $p = (u \equiv v_0, v \equiv v_1, \ldots, v_k \equiv x)$ in G such that $\sum_{i=1}^{k} s((v_{i-1}, v_i)) < \alpha$. The set of nodes α-affected by an arc $a = (u, v)$ is denoted as $\mathrm{Aff}(a)$.

In the following, given a feasible solution π for TT, we will use the slack times s_π defined by π as the function s in the previous definition. Thus, an event x is affected by a delay α occurring on the arc $a = (u, v)$ if the sum of the slack times assigned by the function π to the events on the path from u to x is smaller than α. That is, the planned durations of the activities are not able to absorb the delay α and thus x will be delayed.

We assume that the recovery capabilities allow to change the time of at most Δ events. Formally, given $\Delta \in \mathbb{N}$, each algorithm in $\mathbb{A}$ is able to compute a feasible solution π' if $|\mathrm{Aff}(a)| \leq \Delta$ for each $a \in A$. This implies that a robust solution for $\mathcal{RTT}$ must guarantee that a delay of at most α time may affect at most Δ events. Note that, $\mathcal{RTT}$ only requires to find a feasible solution. Nevertheless, it is worth to find a solution that minimizes the objective function of TT.

From now on, we restrict our attention to rooted out-trees and we define the $\mathcal{RTT}_{opt}$ optimization problem as follows:

$\mathcal{RTT}_{opt}$	
GIVEN:	A tree $T = (V, A)$, a function $w : V \to \mathbb{R}_{\geq 0}$, and $\alpha, \Delta \in \mathbb{N}$.
PROB.:	Find a function $\pi : V \to \mathbb{R}_{\geq 0}$ s. t. each arc in A α-affects at most Δ nodes, according to the function $s_\pi : A \to \mathbb{R}_{\geq 0}$ defined as $s_\pi(a = (i, j)) = \pi(j) - \pi(i) - 1$, $a \in A$, and s. t. $f(\pi) = \sum_{v \in V} \pi(v) w(v)$ is minimal

In the next lemma, we prove that, when the modifications are confined to a single delay of at most α time, there exists a solution for the $\mathcal{RTT}_{opt}$ problem which assigns only slack times equal to α.

Lemma 3.3: Given an instance i of $\mathcal{RTT}$, for each feasible solution π for i there exists a solution π' for i such that $f(\pi') \leq f(\pi)$ and either $\pi'(y) = \pi'(x) + 1$ or $\pi'(y) = \pi'(x) + 1 + \alpha$ for each arc $a = (x, y) \in A$.

Proof: Let $\pi^0 = \pi$, we define π^k as the solution obtained from π^{k-1} by applying one of following operations starting from the root, downward to the leaves until none of them can be applied.

1) For an arc $a = (x, y)$ such that $\pi^{k-1}(y) > \pi^{k-1}(x) + 1 + \alpha$, we assign $\pi^k(y) = \pi^{k-1}(x) + 1 + \alpha$;
2) For an arc $a = (x, y)$ such that $\pi^{k-1}(x) + 1 < \pi^{k-1}(y) < \pi^{k-1}(x) + 1 + \alpha$, we assign $\pi^k(y) = \pi^{k-1}(x) + 1$.

The last obtained solution is π'. By construction $\pi'(x) \leq \pi(x)$, for each $x \in V$, therefore $f(\pi') \leq f(\pi)$.

To show that π' is a feasible solution, we prove that π^k is feasible if π^{k-1} is feasible. To this end, it is sufficient to show that if a node z is α-affected by an arc b in π^k then it was also α-affected by the same arc in π^{k-1}. If a is not in the path from b to z, the statement easily follows. We then assume a in the path from b to z.

Let us suppose that π^k is obtained from π^{k-1} by operation 1) on arc a. Since z is not α-affected by b in π^k because the sum of the slack times on the path from b to z is at at least α, we don't care whether it was α-affected or not by b in π^{k-1}.

Now, let us assume that π^k is obtained from π^{k-1} by operation 2) on arc $a = (x, y)$. First of all, observe that the sum S of the slack times on the path C from $b = (u, w)$ to z is given by $S = \sum_{c \in C} s_{\pi^k}(c) = \pi^k(z) - \pi^k(u) - \sum_{c \in C} L(c)$. If a is not incident to z, then sum S is equal to the same sum computed at π^{k-1}. In fact, the durations $L(c)$ of the activities c on the path C are unchanged and the operation 2) modifies only the time assigned to y, hence obtaining $\pi^{k-1}(z) - \pi^{k-1}(u) = \pi^k(z) - \pi^k(u)$.

On the other hand, if $a = (x, y)$ is the last arc in the path C to z, i.e., $y = z$, the sum S of the slack times changes. By contradiction, assume z is not affected at π^{k-1}, but it is α-affected at π^k. Then, there must exist at least another arc c on C such that either $s_{\pi^{k-1}}(c) \geq \alpha$ or $0 < s_{\pi^{k-1}}(c) < \alpha$. In the former case, z is not α-affected by b in π^k. In the latter case, the hypothesis that the operations are applied from the root downward to the leaves is contradicted, since operation 2) is applicable to c. □

Lemma 3.3 implies that, without loss of generality we can focus on solutions that assigns only slack times of either 0 or α.

4 Complexity

Let $\mathcal{RTT}_{dec}$ be the underlying decision problem of $\mathcal{RTT}_{opt}$.

$\mathcal{RTT}_{dec}$	
GIVEN:	A tree $T = (V, A)$, function $w : V \to \mathbb{R}_{\geq 0}$, $\alpha, \Delta \in \mathbb{N}$, and $K' \in \mathbb{R}_{\geq 0}$.
PROB.:	Is there a function $\pi : V \to \mathbb{R}_{\geq 0}$ such that each arc in A α-affects at most Δ nodes, according to the function $s_\pi : A \to \mathbb{R}_{\geq 0}$ defined as $s_\pi(a = (i, j)) = \pi(j) - \pi(i) - 1$, and such that $\sum_{v \in V} \pi(v) w(v) \leq K'$?

In the next theorem, we show that $\mathcal{RTT}_{dec}$ is NP-complete by a transformation from *Knapsack* [10].

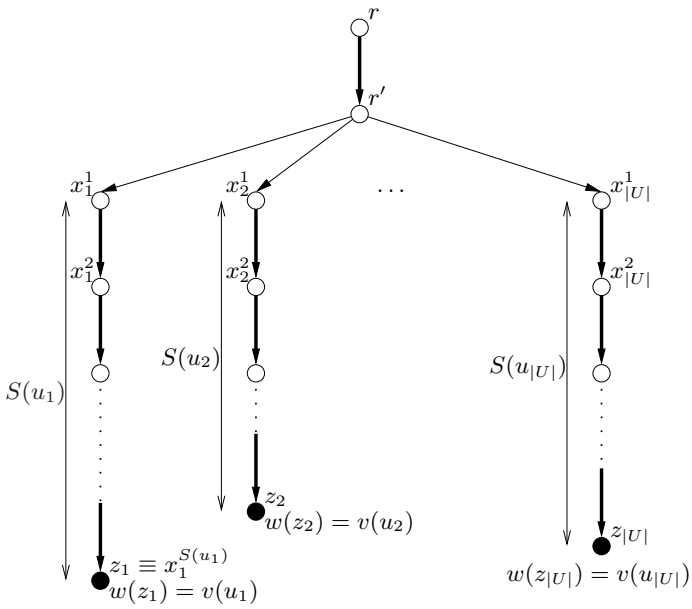

Fig. 1. Instance of $\mathcal{RTT}$ used for the transformation from an instance of knapsack.

Knapsack	
GIVEN:	A finite set U, for each $u \in U$ a size $S(u) \in \mathbb{Z}_{>0}$, a value $v(u) \in \mathbb{Z}_{>0}$, and positive integers $B, K \in \mathbb{Z}_{>0}$.
PROB.:	Is there a subset $U' \subseteq U$ s. t. $\sum_{u \in U'} S(u) \leq B$ and $\sum_{u \in U'} v(u) \geq K$?

Theorem 4.1: $\mathcal{RTT}_{dec}$ is NP-complete.

Proof: A solution for $\mathcal{RTT}_{dec}$ can be verified in polynomial time, as we only need to find out whether there exists a subtree of size bigger than Δ where no slack time α has been added. This can be performed by counting for each node v, how many descending nodes are affected if a delay of α is assumed to occur at the in-arc of x. Starting from the leaves of the tree and moving up until the root, the procedure needs a simple visit of the tree, hence $\mathcal{RTT}_{dec}$ is in NP.

Given an instance $I^{Knapsack}$ of *Knapsack* where $U = \{u_1, u_2, \ldots, u_{|U|}\}$, we define an instance $I^{\mathcal{RTT}}$ of $\mathcal{RTT}_{dec}$. See Fig. 1 for a visualization of $I^{\mathcal{RTT}}$. Without loss of generality, we assume that $S(u_i) \leq B$, for each $u_i \in U$. The set of nodes is made of: nodes r, r' and the sets of nodes $X_i = \{x_i^1, x_i^2, \ldots, x_i^{S(u_i)} \equiv z_i\}$, for each $u_i \in U$. The set of arcs A is made of: arc (r, r'); arcs (r', x_i^1), for each $i = 1, 2, \ldots, |U|$; and arcs (x_i^j, x_i^{j+1}), for each $i = 1, 2, \ldots, |U|$ and for each $j = 1, 2, \ldots, S(u_i) - 1$.

The weight of each node in V is 0 except for nodes z_i, $i = 1, 2, \ldots, |U|$, where $w(z_i) = v(u_i)$. Finally, $\Delta = B + 1$, $\alpha = 1$ and $K' = \sum_{u_i \in U} v(u_i)(S(u_i) + 2) - K$. It is worth noting that the particular tree construction preserves the size of the *Knapsack* instance, as each path of nodes of the same weight w can be compacted and implicitly be represented by two numbers, namely, the number of nodes of the path and the weight w of each node. This implies that each path $(x_i^1, x_i^2, \ldots, x_i^{S(u_i)-1})$ of our construction will be represented by the pair $(S(u_i)-1, 0)$. Also the operations performed on such paths do not need the explicit representation of the tree. In particular, the check needed to verify whether a solution is feasible can be done efficiently with respect to the compacted instance.

Now we show that, if there exists a "yes" solution for *Knapsack*, then there exists a "yes" solution for $\mathcal{RTT}_{dec}$.

Given a "yes" solution $U' \subseteq U$ for $I^{Knapsack}$, we define the following solution π for $I^{\mathcal{RTT}}$: $\pi(r) = 0$; $\pi(r') = 1$; $\pi(x_i^1) = 2$ for each $u_i \in U'$ and $\pi(x_i^1) = 3$ for each $u_i \notin U'$; $\pi(x_i^j) = \pi(x_i^1) + j - 1$, for each $i = 1, 2, \ldots, |U|$ and for each $j = 2, 3, \ldots, S(u_i)$.

Note that, for each $u_i \in U'$, $\pi(z_i) = S(u_i) + 1$ while for each $u_i \notin U'$, $\pi(z_i) = S(u_i) + 2$. As the weight of each node in V is 0 except for nodes z_i, then $f(\pi) = \sum_{i=1}^{|U|} \pi(z_i) \cdot w(z_i) = \sum_{u_i \in U'} (S(u_i) + 1) \cdot v(u_i) + \sum_{u_i \notin U'} (S(u_i) + 2) \cdot v(u_i) = \sum_{u_i \in U} (S(u_i) + 2) \cdot v(u_i) - \sum_{u_i \in U'} v(u_i) \leq \sum_{u_i \in U} (S(u_i) + 2) \cdot v(u_i) - K = K'$. We have to show that each arc in A α-affects at most Δ nodes. As $\Delta = B + 1 > S(u_i)$, for each $u_i \in U$, then all the arcs but (r, r') do not α-affect more than Δ nodes. Moreover, for each $u_i \notin U'$, $\pi(x_i^1) - \pi(r') = 1 + \alpha$. Hence, the arc (r, r') α-affects r' and nodes x_i^j for each $u_i \in U'$ and for each $j = 1, 2, \ldots, S(u_i)$. Then, the overall number of affected nodes is $1 + \sum_{u_i \in U'} S(u_i) \leq 1 + B = \Delta$.

Now we show that, if there exists a "yes" solution for $\mathcal{RTT}_{dec}$, then there exists a "yes" solution for *Knapsack*. Given a "yes" solution π' for $I^{\mathcal{RTT}}$, we define a "yes" solution π that assigns slack times only to arcs (r', x_i^1), $i = 1, 2, \ldots, |U|$ as follows: $\pi(r) = 0$; $\pi(r') = 1$; $\pi(x_i^1) = 3$ for each i such that π' assigns at least a slack time in the path from r' to z_i, i.e. $\pi'(z_i) - \pi'(r') \geq S(u_i) + \alpha$; $\pi(x_i^1) = 2$ for each i such that $\pi'(z_i) - \pi'(r') < S(u_i) + \alpha$; $\pi(x_i^j) = \pi(x_i^1) + j - 1$, for each $i = 1, 2, \ldots, |U|$ and for each $j = 2, 3, \ldots, S(u_i)$. Note that, if π' is a "yes" solution for $I^{\mathcal{RTT}}$ then π is also a "yes" solution for $I^{\mathcal{RTT}}$. In fact, $\pi(z_i) \leq \pi'(z_i)$ and then $f(\pi) \leq f(\pi') \leq K'$. Moreover, the number of nodes α-affected by (r, r') in π is less than or equal to the number of nodes α-affected by (r, r') in π'.

We define a "yes" solution for $I^{Knapsack}$ as $U' = \{u_i : \pi(x_i^1) = 2\}$. We have to show that $\sum_{u \in U'} S(u) \leq B$ and $\sum_{u \in U'} v(u) \geq K$. As the number of nodes α-affected by (r, r') in the solution π is less than or equal to Δ, then $\Delta \geq 1 + \sum_{i:\pi(x_i^1)=2} |X_i| = 1 + \sum_{u_i \in U'} S(u_i)$. Hence $\sum_{u_i \in U'} S(u_i) \leq \Delta - 1 = B$. As the weight of each node in V is 0 except for nodes z_i, then $f(\pi) = \sum_{i=1}^{|U|} \pi(z_i) \cdot w(z_i) = \sum_{i:\pi(x_i^1)=2} (S(u_i)+1) \cdot w(z_i) + \sum_{i:\pi(x_i^1)=3} (S(u_i)+2) \cdot w(z_i) = \sum_{i:\pi(x_i^1)=2} (S(u_i)+1) \cdot v(u_i) + \sum_{i:\pi(x_i^1)=3} (S(u_i)+2) \cdot v(u_i) = \sum_{i=1}^{|U|} (S(u_i) + 2) \cdot v(u_i) - \sum_{i:\pi(x_i^1)=2} v(u_i) = \sum_{u_i \in U} (S(u_i) + 2) \cdot v(u_i) - \sum_{u_i \in U'} v(u_i) \leq K' = \sum_{u_i \in U} (S(u_i) + 2) \cdot v(u_i) - K$. Hence $\sum_{u_i \in U'} v(u_i) \geq K$. □

Corollary 4.2: $\mathcal{RTT}_{opt}$ and computing $P_{rob}(\mathcal{RTT})$ are NP-hard.

5 Pseudo-Polynomial Time Algorithm

Based on the dynamic programming techniques, in this section we devise a pseudo-polynomial time algorithm for $\mathcal{RTT}_{opt}$.

Let us introduce further notation. Note that, a solution for $\mathcal{RTT}_{opt}$ is $\mathcal{RTT}$-optimal. Let $T = (V, A)$ be an arbitrarily ordered rooted tree, i.e. for each node v we can distinguish its children, denoted as $N_o(v)$, as an arbitrarily ordered set $\{v_1, v_2, \ldots, v_{|N_o(v)|}\}$. For an arbitrary subtree $S(v)$ rooted at $v \in V$, let $N_o(S(v))$ denote the set of nodes y such that $(x, y) \in A$, $x \in S(v)$ and $y \notin S(v)$. Clearly, when $S(v) = \{v\}$, $N_o(S(v)) \equiv N_o(v)$. In addition, for each node $v \in T$, let $T(v)$ be the full subtree of T rooted in v and let $T_i(v)$ denote the full subtree of T rooted at v limited to the first (according to the initially chosen order) i children of v. Moreover, recalling that T is a weighted tree, let $c(v) = \sum_{x \in T(v)} w(x)$ be the sum of the weights of the nodes in $T(v)$ and let $|T(v)|$ be the number of nodes belonging to $T(v)$. Note that, the values $|T(v)|$ and $c(v)$ for all $v \in V$ can be computed in linear time by visiting $T(r)$ where r is the root of T. Finally, given a node v, let us denote as $d(r, v)$ the number of arcs on the path from the root r of T and v.

In the next lemma, we prove that when the slack time of an arc (u, v) changes, all the times associated with the events in $T(v)$ changes accordingly.

Lemma 5.1: Given a tree T, consider two feasible solutions π' and π such that, for an arc $\bar{a} = (u, v)$, $s_{\pi'}(\bar{a}) = s_\pi(\bar{a}) - \alpha$ and $s_\pi(a) = s_{\pi'}(a)$, for each $a \neq \bar{a}$. Then, $f(\pi') = f(\pi) - \alpha c(v)$.

Proof: By construction, all the events contained in the subtree $T(v)$ are delayed in π of α time with respect to π' due to the slack time associated with arc $\bar{a}$. For each node $x \in T(v)$ then, it holds $\pi'(x) = \pi(x) - \alpha$. Hence, $f(\pi') = f(\pi) - \alpha c(v)$. □

Definition 5.2: Given a feasible solution π for the $\mathcal{RTT}$ problem and a node $v \in V$, the *ball* $B_\pi(v)$ is the largest subtree rooted at v such that each node in $B_\pi(v)$ (including v) has its incoming arc a with $s_\pi(a) = 0$.

Given a feasible solution π, $B_\pi(v)$ represents the set of nodes in π which are surely affected by each possible delay occurring at $a = (u, v)$. Due to the feasibility of π, no more than Δ nodes can belong to $B_\pi(v)$, that is $|B_\pi(v)| \leq \Delta$. Note that, if $s_\pi(a) = \alpha$, then $B_\pi(v)$ is empty. We say that $B_\pi(v)$ can be *extended* if there exists an arc (x, y) such that $x \in B_\pi(v)$, $y \notin B_\pi(v)$, $w(y) > 0$, and for each $a \in A$ such that $x \in \mathrm{Aff}(a)$, $|\mathrm{Aff}(a)| < \Delta$. A ball is said *maximal* if it cannot be extended.

Lemma 5.3: For each instance of $\mathcal{RTT}$, with $\Delta \geq 1$, there exists an $\mathcal{RTT}$-optimal solution such that at most one of two consecutive arcs has a slack time of α.

Proof: Suppose that there is some $\mathcal{RTT}$-optimal solution π that assigns a slack time to both of two consecutive arcs, i.e. there exist $(x, y), (y, z) \in A$ such that $s_\pi(x, y) = \alpha$, $s_pi(y, z) = \alpha$, and $s_\pi(w) = 0$ for each $w \in N_o(z)$. Then, we can define a solution π' such that $s_{\pi'}(x, y) = \alpha$, $s_{\pi'}(y, z) = 0$, and $s_{pi'}(z, w) = \alpha$ for each $w \in N_o(z)$. Note that π' is feasible because $|B_{\pi'}(z)| = 1 \leq \Delta$, $|B_{\pi'}(w)| \leq |B_{\pi'}(w)|$ for $w \in N_o(z)$, and the remaining balls are not changed. Moreover, π' differs from π only in z. Specifically, $\pi'(z) = \pi(x) + 2 + \alpha$, while $\pi(z) = \pi(x) + 2 + 2\alpha$, whereas, for every node $w \in N_o(z)$, $\pi'(w) = \pi'(z) + 1 + \alpha = \pi'(y) + 2 + \alpha = \pi(x) + 3 + 2\alpha = \pi(w)$. Clearly, π' remains unchanged for every other node of T. Thus, $f(\pi') = f(\pi) - \alpha w(z) \leq f(\pi)$ because $w(z) \geq 0$. Therefore, π' is a feasible solution, with cost no greater than the optimal one, with no two consecutive arcs having a slack time α. □

The above proof implies a stronger result: no $\mathcal{RTT}$-optimal solution can have two consecutive arcs with a slack time α when all the events have positive weights. For arbitrary weights, there is an optimal solution that satisfies such a property. Thus, from now on, without loss of generality we can focus on optimal solutions that do not have two consecutive arcs with a slack time α.

Lemma 5.4: There exists an $\mathcal{RTT}$-optimal solution π such that $B_\pi(v)$ is maximal for each $v \in V$.

Proof: By contradiction, we assume that there exists an instance such that, for an $\mathcal{RTT}$-optimal solution π there exists a non empty set of nodes $\mathcal{V}$ which contradicts the thesis: for each $v \in \mathcal{V}$, $B_\pi(v)$ can be extended by adding a node from $N_o(B_\pi(v))$. Let $v \in \mathcal{V}$ be a node such that the distance $d(r, v)$ from r to v is minimal. As $B_\pi(v)$ can be extended, then there exists an arc (x, y) such that $x \in B_\pi(v)$, $y \notin B_\pi(v)$ and for each $a \in A$ such that $x \in \mathrm{Aff}(a)$, $|\mathrm{Aff}(a)| < \Delta$. It follows that $\pi(y) = \pi(x) + 1 + \alpha$. Then, let us define π' as the solution derived by setting $s_{\pi'}(x, y) = 0$ and $s_{\pi'}(y, w) = \alpha$ for each $w \in N_o(y)$. Specifically, π' assigns $\pi'(y) = \pi(x) + 1$, $\pi'(w) = \pi'(y) + 1 + \alpha$, for $w \in N_o(y)$. Clearly, π' is feasible. Indeed, $|B_{\pi'}(u)| = |B_\pi(u)| + 1 \leq \Delta$ for each u such that $x \in B_\pi(u)$ and $|B_{\pi'}(y)| \leq |B_\pi(y)| \leq \Delta$. Finally, since by Lemma 5.3 $s_\pi(y, w) = 0$ for each $w \in N_o(y)$, it holds $\pi'(u) = \pi(u)$ for each $u \in V \setminus \{y\}$. In fact, for each $w \in N_o(y)$, $\pi'(w) = \pi(x) + 2 + \alpha = \pi(w)$. Thus, $f(\pi') = f(\pi) - \alpha w(y)$ and thus $f(\pi') \leq f(\pi)$ because $w(y) \geq 0$. Repeating the above construction for each node in $\mathcal{V}$, a feasible solution is built, with cost no greater than the optimal one, such that $B_\pi(v)$ is maximal for each $v \in V$. □

From now on, without loss of generality, we focus on optimal solutions with maximal balls.

As regards to compute an optimal solution for the $\mathcal{RTT}_{opt}$ problem on T, it is easy to see that, when $\Delta = 0$, there is a trivial optimal solution $\hat{\pi}$ which associates to each arc $a \in T$ a slack time equal to α. From now on, let $\hat{f} = \sum_{v \in T} \hat{\pi}(v) w(v)$ denote the cost of $\hat{\pi}$ on T.

When $\Delta > 0$, we are now in position to describe a dynamic programming algorithm to compute an $\mathcal{RTT}$-optimal solution π^*. Let us start deriving the slack time on the arcs outgoing from the root r of T. Since r has no incoming arcs, it cannot be affected by any delay. Thus, r can be considered as a node having an incoming arc associated with a slack time of α. Therefore, by Lemma 5.3, for each arc a outgoing from the root r it holds $s_{\pi^*}(a) = 0$. Moreover, according to Lemma 5.4,

for each $v \in N_o(r)$, π^* must have a ball $B_{\pi^*}(v)$ of size $\min\{|T(v)|, \Delta\}$.

In order to complete π^*, let us introduce the following notations. For any $\mathcal{RTT}$ solution π, let $f_i^v(\pi)$ denote the value of the objective function $\sum_{u \in T_i(v)} \pi(u)w(u)$ computed only on the nodes of $T_i(v)$. Moreover, let $f^v(\pi)$ be the objective function $\sum_{u \in T(v)} \pi(u)w(u)$ evaluated on $T(v)$.

With $\mathcal{G}_v[i,j]$, with $i,j \neq 0$, we mean the maximum gain with respect to the solution $\hat{\pi}$ achievable with any solution π that has a ball $B_\pi(v)$ of size at most j in the subtree limited to its first i children $T_i(v)$ rooted at v, i.e. $|B_\pi(v) \cap T_i(v)| \leq j$. As $\mathcal{G}_v[i,j]$ must be the maximum gain, solution π must be optimal with respect to $T_i(v)$, and thus $|B_\pi(v) \cap T_i(v)| = \min\{j, |T_i(v)|\}$. Formally, when $i,j \neq 0$, $\mathcal{G}_v[i,j] = \max_\pi \{f_i^v(\hat{\pi}) - f_i^v(\pi) \ : \ |B_\pi(v) \cap T_i(v)| \leq j\} = \{f_i^v(\hat{\pi}) - f_i^v(\pi^*) \ : \ |B_{\pi^*}(v) \cap T_i(v)| = \min\{j, |T_i(v)|\}\}$.

Moreover, with $\mathcal{G}_v[0,j]$, for $1 \leq j \leq \Delta$, we designate the maximum gain with respect to the solution $\hat{\pi}$ achievable with any solution π that has a ball $B_\pi(v)$ of size at most j in the subtree limited to node v, i.e. $|B_\pi(v) \cap v| \leq j$. As $\mathcal{G}_v[i,j]$ must be the maximum gain, solution π must be optimal with respect to $T_0(v)$, and thus $|B_\pi(v) \cap T_0(v)| = \min\{j, |T_0(v)|\} = 1$. Thus, $\mathcal{G}_v[0,j]$, for $1 \leq j \leq \Delta$ denotes the gain of a solution π^* that has a ball of size 1 in node v, that is π^* differs from $\hat{\pi}$ only for the slack time on the incoming arc a to v, i.e. $s_\pi(a) = 0$. As proved in Lemma 5.1, we can directly set $\mathcal{G}_v[0,j] = f^v(\hat{\pi}) - f^v(\pi^*) = \alpha c(v)$ for $1 \leq j \leq \Delta$.

Finally, we denote with $\mathcal{G}_v[i,0]$, for $0 \leq i \leq |N_o(v)|$, the maximum gain with respect to $\hat{\pi}$ achievable with an optimal solution π^* that, having a slack time of α on the incoming arc in v, must set (by Lemma 5.3) the slack time equal to 0 for each arc (v, v_ℓ), for $1 \leq \ell \leq i$. Thus, π^* must have a ball of size $\min\{\Delta, |T(v_\ell)|\}$ in each subtree rooted at the first i children $v_1, \ldots, v_i$ of v. Formally, $\mathcal{G}_v[i,0] = \sum_{\ell=1}^{i} \{f^{v_\ell}(\hat{\pi}) - f^{v_\ell}(\pi^*) \ : \ |B_{\pi^*}(v_\ell)| \leq \min\{\Delta, |T(v_\ell)|\}\}$.

Lemma 5.5: When $i,j \neq 0$, the gains $\mathcal{G}_v[i,j]$ can be recursively computed as $\mathcal{G}_v[i,j] = \max_{0 \leq s < j}\{\mathcal{G}_v[i-1, j-s] + \mathcal{G}_{v_i}[|N_o(v_i)|, s]\}$.

Proof: By definition, let π^* be an optimal solution such that $\mathcal{G}_v[i,j] = \{f_i^v(\hat{\pi}) - f_i^v(\pi^*)\}$ and let the ball $B_{\pi^*}(v)$, associated with the optimal solution π^*, be of size $t = \min\{j, |T_i(v)|\}$. As trees $T(v_i)$ and $T_{i-1}(v)$ are node-disjoint, $B_{\pi^*}(v) \cap T_{i-1}(v)$ and $B_{\pi^*}(v) \cap T(v_i)$ are two disjoint subtrees, which represent the balls of π^* restricted to $T_{i-1}(v)$ and to $T(v_i)$, respectively. Let s be the size of the latter ball (which can be empty), and $t - s$ the size of the former one (which cannot be empty because contains at least the node v).

Therefore, $\mathcal{G}_v[i,j]$ can be rewritten as $\mathcal{G}_v[i,j] = \{f_i^v(\hat{\pi}) - f_i^v(\pi')\} + \{f^{v_i}(\hat{\pi}) - f^{v_i}(\pi'')\}$, where

$$\pi'(u) = \begin{cases} \pi^*(u) & \text{if } u \in T_{i-1}(v) \\ \hat{\pi}(u) & \text{otherwise,} \end{cases}$$

and

$$\pi''(u) = \begin{cases} \pi^*(u) & \text{if } u \in T(v_i) \\ \hat{\pi}(u) & \text{otherwise.} \end{cases}$$

Note that both π' and π'' must be optimal, otherwise one can contradict the optimality of π^* by using a cut-and-paste argument. Indeed, suppose, by contradiction, that $\pi'(u)$ is not optimal. Then, there is another solution in $T_{i-1}(v)$, say $\overline{\pi}$, with $|B_{\overline{\pi}}(v)| = t - s$, whose gain $\{f_i^v(\hat{\pi}) - f_i^v(\overline{\pi})\}$ is greater than that of π'. Substituting π' with $\overline{\pi}$, one can build a new solution

$$\pi(u) = \begin{cases} \overline{\pi}(u) & \text{if } u \in T_{i-1}(v) \\ \pi''(u) & \text{if } u \in T(v_i) \\ \hat{\pi}(u) & \text{otherwise,} \end{cases}$$

which contradicts the optimality of the solution π^*. Thus, $\mathcal{G}_v[i,j] = \mathcal{G}_v[i-1, j-s] + \mathcal{G}_{v_i}[|N_o(v_i)|, s]$, for some $s \in [0, j)$.

Finally, since it is not known in advance how the optimal ball $B_{\pi^*(v)}$ is partitioned between $T_{i-1}(v)$ and $T(v_i)$, one has to check all the feasible combinations, that is: $\mathcal{G}_v[i,j] = \max_{0 \leq s < j} \{\mathcal{G}_v[i-1, j-s] + \mathcal{G}_{v_i}[|N_o(v_i)|, s]\}$. □

In order to compute an $\mathcal{RTT}$-optimal solution, we propose a dynamic programming algorithm which requires $O(\Delta^2 n)$ time complexity and $O(\Delta n)$ space, when $\Delta \geq 1$, whereas it requires $O(n)$ time and $O(n)$ space when $\Delta = 0$. The algorithm makes use of the two procedures `SA-DP` and `BUILD`, given below. Algorithm `SA-DP` (which stands for Slack Assignment with Dynamic Programming) considers an arbitrarily ordered tree T in input and performs a visit of T. It uses for each node $v \in T$ two matrices G_v and SOL_v, both of size $(N_o(v) + 1) \times (\Delta + 1)$. Concerning a generic node v in T and the matrix G, the `SA-DP` algorithm stores in each entry $G_v[i,j]$ the corresponding value $\mathcal{G}_v[i,j]$.

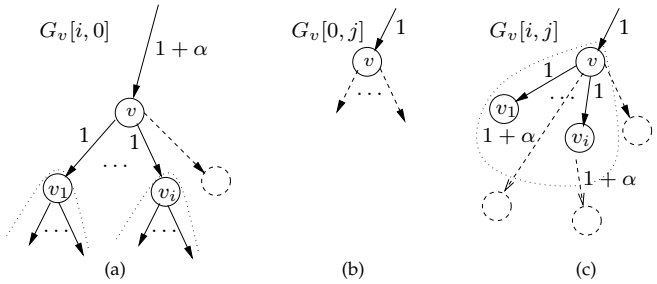

Fig. 2. The three configurations that must be considered when computing matrix G_v.

Fig. 2 shows the possible configurations to consider when evaluating $G_v[i,j]$.

Note that when $j > |T_i(v)|$, we set $G_v[i, |T_i(v)|] = G_v[i, |T_i(v)| + 1] = \ldots = G_v[i,j]$.

If $j = 0$ (Fig. 2(a)), since as discussed above $\mathcal{G}_v[i,0] = \sum_{\ell=1}^{i} \{f^{v_\ell}(\hat{\pi}) - f^{v_\ell}(\pi^*) \ : \ |B_{\pi^*}(v_\ell)| \leq \min\{\Delta, |T(v_\ell)|\}\}$, we compute $\mathcal{G}_v[i,0] = \sum_{\ell=1}^{i} G_{v_\ell}[|N_o(v_\ell)|, \Delta]$.

If $i = 0$ and $j > 0$ (Fig. 2(b)), by the above discussion, $G_v[0,j] = \alpha c(v)$, for each $1 \leq j \leq \Delta$.

Finally, if $i > 0$ and $j > 0$ (Fig. 2(c), by Lemma 5.5), $G_v[i,j] = \max_{0 \leq s < j} \{G_v[i-1, j-s] + G_{v_i}[|N_o(v_i)|, s]\}$.

Concerning a generic node v in T and the matrix SOL_v, the `SA-DP` algorithm memorizes in SOL_v the

Algorithm SA-DP
Input: $v \in V$
Output: SOL_u, for each $u \in T(v)$
1. **for** $i = 1$ to k
2. SA-DP(v_i)
3. $G_v[i,0] = \sum_{\ell=1}^{i} G_{v_\ell}[|N_o(v_\ell)|, \Delta]$, for $0 \le i \le N_o(v)$
4. $G_v[0,j] = \alpha c(v)$, for $1 \le j \le \Delta$
5. **for** $i = 1$ **to** k
6. **for** $j = 1$ **to** Δ
7. **if** $j \le |T_i(v)|$ **then**
8. $G_v[i,j] = \max_{0 \le s < j} \{G_v[i-1, j-s] + G_{v_i}[|N_o(v_i)|, s]\}$
9. let s^* be the index giving the maximum at Line 8
10. $SOL_v[i,j] = s^*$
11. **else** $G_v[i,j] = G_v[i,j-1]$
12. $SOL_v[i,j] = SOL_v[i,j-1]$

choices that lead to the optimal gains computed in the corresponding matrix G_v (Lines 10 and 12 of the algorithm). All matrices SOL_v, $v \in T$, are first initialized by assigning 0 to each entry. Then, if $i, j > 0$, the entry $SOL_v[i,j]$ stores the size s of the ball $B_\pi(v_i)$ rooted at the i-th child v_i of v which gives the maximum gain when evaluating $G_v[i,j]$. Basically, if $SOL_v[i,j] \neq 0$, then the arc (v, v_i) has no slack time. Viceversa, if $SOL_v[i,j] = 0$, either the arc (v, v_i) has a slack time of α (i.e., $|B_\pi(v_i)| = 0$ that is $j = 0$) or such an arc does not exist (i.e., $i = 0$).

Now, we have to show how to construct the optimal solution π from matrices SOL_v, i.e. how to assign to each $v \in T$ the value $\pi(v)$. Recall first that, by the discussion after Lemma 5.3, $\pi(r) = 0$ and, for each $v_\ell \in N_o(r)$, $\pi(v_\ell) = \pi(r) + 1 = 1$. Moreover, a ball $B_\pi(v_\ell)$ of size $\min = \{|T(v_\ell)|, \Delta\}$ is rooted at each $v_\ell \in N_o(r)$. An optimal solution is computed by calling, for each child $v_\ell \in N_o(r)$, the procedure BUILD$(v_\ell, |N_o(v_\ell)|, \Delta)$.

Algorithm BUILD
Input: $v \in V$, $i \in [0, |N_o(v)|]$, $j \in [0, \Delta]$
1. **if** $i > 0$ **then**
2. **if** $SOL_v[i,j] > 0$ **then**
3. $s := SOL_v[i,j]$; $\pi(v_i) := \pi(v) + 1$
4. BUILD$(v_i, |N_o(v_i)|, s)$
5. BUILD$(v, i-1, j-s)$
6. **else**
7. $\pi(v_i) := \pi(v) + 1 + \alpha$
8. **for each** $w \in N_o(v_i)$
9. $\pi(w) := \pi(v_i) + 1$
10. BUILD$(w, |N_o(w)|, \Delta)$
11. BUILD$(v, i-1, j)$

Algorithm BUILD(v, i, j) recursively builds the most profitable ball $B_\pi(v)$ of size j with respect to $T_i(v)$. At first, it determines the slack time on the arc (v, v_i) depending on $SOL_v[i,j]$. If $SOL_v[i,j] > 0$, then the slack time on the arc (v, v_i) is equal to 0, and the ball $B_\pi(v)$ consists of two sub-balls: one of size $SOL_v[i,j]$ in $T(v_i)$ and one of size $j - SOL_v[i,j]$ in $T_{i-1}(v)$. Whereas, if $SOL_v[i,j] = 0$, the ball of size j in the subtree $T(v)$ does not contain nodes in $T(v_i)$ but only in $T_{i-1}(v)$. However, since the slack time on the arc (v, v_i) is equal to α, by Lemma 5.3, all the arcs outgoing from v_i, say (v_i, w_k) for $1 \le k \le |N_o(v_i)|$, have slack time equal to 0. The balls of size at most Δ rooted at the nodes $w \in N_o(v_i)$, are recursively built by invoking $|N_o(v_i)|$ times Algorithm BUILD, one for each child of v_i.

Finally, we define Algorithm SA-DP_BUILD which outputs a robust timetable π by performing first SA-DP(r) and BUILD$(v_\ell, |N_o(v_\ell)|, \Delta)$, for each $v_\ell \in N_o(r)$. By construction, the following theorem can be stated.

Theorem 5.6: For any $\Delta \ge 0$, SA-DP_BUILD is $\mathcal{RTT}$-optimal.

Moreover:

Theorem 5.7: For $\Delta \ge 1$, SA-DP_BUILD requires $O(\Delta^2 n)$ time and $O(\Delta n)$ space. For $\Delta = 0$, SA-DP_BUILD requires $O(n)$ time and $O(n)$ space.

Proof: The time complexity of the SA-DP procedure follows by observing that, for a given $v \in T$, to fill the entry of $G_v[i,j]$ requires $O(i)$ time if $j = 0$ (Line 3), $O(1)$ if $i = 0$ and $j > 0$ (Line 4), and $O(j)$ if $i, j > 0$ (see the recurrence defined by Lines 5–12), and hence the matrices G_v and SOL_v are filled in $O(|N_o(v)|\Delta^2)$ time. Thus, to fill all the matrices in T, it costs $\sum_{v \in T} O(|N_o(v)|\Delta^2) = O(n\Delta^2)$. On the other side, procedure BUILD performs a visit of T to compute the assignment $\pi(v)$ for each node $v \in T$ and thus, it takes $O(n)$ time.

If $\Delta = 0$, Lines 6–12 of procedure SA-DP are not executed and hence SA-DP only performs a visit of T requiring $O(n)$ time. □

Theorem 5.8: For $\Delta \ge 1$, $P_{rob}(\mathcal{RTT}, \text{SA-DP\_BUILD}) \le 1 + \frac{\alpha}{2}$ and $P_{rob}(\mathcal{RTT}) \ge 1 + \frac{\alpha}{\Delta+1}$.

Proof: For the first part of the theorem, as $\Delta \ge 1$, SA-DP does not leave slack times associated to two consecutive arcs. In fact, for any pair of consecutive arcs $a = (x,y), b = (y,z) \in A$ either $\pi(z) = \pi(x) + 2$ or $\pi(z) = \pi(x) + 2 + \alpha$. Then, for each $v \in V$, $\pi(v) \le d(r,v) + \alpha \lfloor \frac{d(r,v)}{2} \rfloor \le d(r,v)\left(1 + \frac{\alpha}{2}\right)$. For any optimal solution π' of TT, $\pi'(v) \ge d(r,v)$. When $\Delta = 0$, $\pi'(v) \ge d(r,v)\alpha$ and this is equal to the solution provided by SA-DP_BUILD, as it does not remove any slack time. Therefore, the statement holds.

Regarding the second part of the theorem, it is sufficient to give an instance such that any robust solution π implies $f(\pi) \ge (1 + \frac{\alpha}{\Delta+1}) \cdot f(\pi')$, where π' is an optimal solution for TT. Let us consider a tree consisting of a single path of $\Delta+1$ arcs (x_i, x_{i+1}), $i = 0, 1, \ldots, \Delta$, $x_0 \equiv r$. For each $i = 0, 1, \ldots, \Delta$, $w(x_i) = 0$ and $w(x_{\Delta+1}) > 0$. Each solution π of $\mathcal{RTT}$ is such that $\pi(x_{\Delta+1}) \ge d(x_0, x_{\Delta+1}) + \alpha = \Delta + 1 + \alpha$. An optimal solution π' for TT is such that $\pi'(x_{\Delta+1}) = d(x_0, x_{\Delta+1}) = \Delta + 1$. Hence, $f(\pi) = (\Delta+1+\alpha)w(x_{\Delta+1}) = (1+\frac{\alpha}{\Delta+1}) \cdot (\Delta+1)w(x_{\Delta+1}) = (1 + \frac{\alpha}{\Delta+1}) \cdot f(\pi')$. □

6 Experimental Study

We present the experimental results first on real-world data provided by Trenitalia [18] and, then, on randomly

generated data.

6.1 Real-world data

We consider real case scenarios of *Single-Line Corridors*. A corridor is a sequence of stations linked by multiple tracks. Each station is served by many trains of different types. Types of trains mostly concern the locations that each train serves and its maximal speed. For an example, see Figure 3. In these systems, it is a practical evidence that slow trains wait for faster trains in order to serve passengers to small stations. This situation is modelled with the only assumption that the changes of passengers from one train to another at a station must be guaranteed only when the second train is starting its journey from the current station. In practice, the only restriction is that we do not require as a constraint the possibility for passengers to change for a train which has already started its journey. This does not mean that passengers cannot change train at some station in the middle of a train journey, but only that this is not considered as a constraint. Further motivations for this model can be found in [6], [7].

Let us consider the real-world example provided in Figure 3 where three trains serve the same line. The slowest train, the Espresso, goes from Verona to Bologna, the Interregionale goes from Fortezza to Bologna, and the fastest one, the Euro-City, goes from Brennero to Bologna.

The Euro-City starts its journey before all the other trains, and it arrives at Fortezza station before the departure event of the Interregionale. At Verona Station, the Espresso is scheduled to start its journey after the arrival event of the Euro-City. Hence, there is an arc between the Euro-City and the starting event corresponding to the Interregionale at Fortezza station, and another arc connecting the Euro-City to the starting event of the Espresso at Verona station. As described above, an arc which represents a changing activity can only connect one node to the head of a branch. The DAG obtained by this procedure is a tree, as shown in Figure 3. In general, the result of this procedure is a forest and we link the roots of the trees in this forest to a unique root event. The weights on the events are assigned according to the relevance of the trains which they belong to, and the weight of the root is 0. The rationale behind such a choice is given by the priority that faster trains have with respect to the slower ones, and to the fact that they also serve more passengers. Another interesting weighting function could be to associate different weights to different events according to the number of involved passengers. Unfortunately, this would require more precise input data than that we could retrieve.

Table 1 shows the data used in the experiments referring to 4 corridors provided by Trenitalia. Starting from the provided data and according to the described requirements, we derived event activity networks having tree topologies whose sizes are reported in Table 2. We then apply the `SA-DP_BUILD` algorithm on different scenarios, comparing the obtained robust timetables with the optimal non-robust ones.

Corridor	Line	Stations	Trains
`BrBo`	Brennero–Bologna	48	68
`MdMi`	Modane–Milano	54	291
`BzVr`	Bolzano–Verona	27	65
`PzBo`	Piacenza–Bologna	17	25

TABLE 1
Data used in the experiments.

Corridor	N. of nodes	Max. time of traveling	Avg activity time	Max. N. of hops
`BrBo`	1103	516	9	66
`MdMi`	4358	318	8	27
`BzVr`	648	197	5	37
`PzBo`	163	187	10	14

TABLE 2
Sizes of the trees.

We now show and discuss interesting results about the applicability and the low costs in terms of slack times needed for making robust the considered timetables with respect to different scenarios.

Our experiments are based on three main parameters. Namely, we vary on the maximum number Δ of events that can be affected by an occurring delay, the maximum time delay α, and the case of average or real times L needed to perform the scheduled activities. In what follows, all the activities times and the delays are expressed in minutes.

In order to obtain $\mathcal{RTT}$ instances, for each corridor among `BrBo`, `MdMi`, `BzVr` and `PzBo`, we vary $\Delta \in [1, 2, \ldots, 1000]$ and $\alpha \in \{1, 5, 9, 13, 17\}$. Moreover, we use two different functions L: the first one is based on the real values obtained by available data; the second one is the constant average function which assigns to each activity the same duration time obtained as the average among all real values of each instance. This second function is used to test the behavior of the algorithm based only on the tree network topology, in order to understand the dependability with respect to real values. The average activity times for each instance are shown in Table 2.

For each corridor we show three diagrams concerning the objective function f, the price of robustness P_{rob} of `SA-DP_BUILD`, and the computational time t needed by `SA-DP_BUILD` in the mentioned cases. In each diagram, we show three curves which represent the results obtained by setting L to real values and $\alpha \in \{1, 5, 9\}$. Results obtained by assigning $\alpha \in \{13, 17\}$ are not shown as they are less significant being α too large compared with the average activity time. Furthermore, for the instance `BrBo`, we give the three diagrams obtained by setting L to the corresponding average activity time. For any other instance we do not give these diagrams as

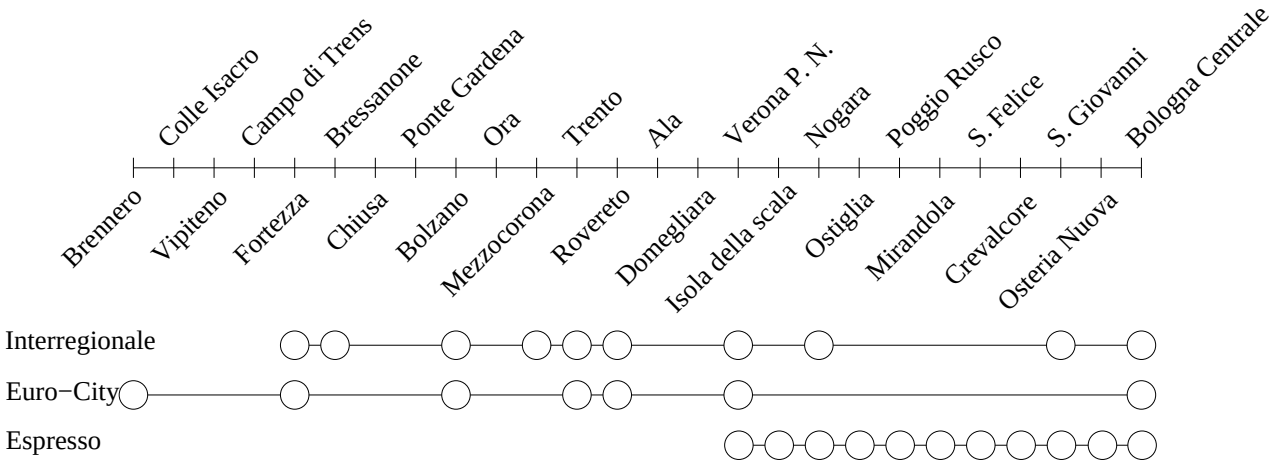

Fig. 3. Example of three trains serving a same line. For the sake of clarity, for each station and for each train, we represented only one circle which indeed corresponds to an arrival and a departure event.

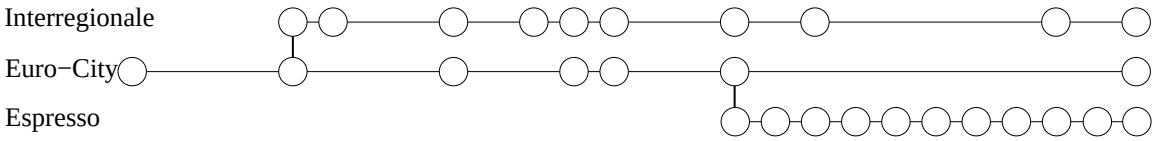

Fig. 4. A tree obtainable from the example provided by Figure 3.

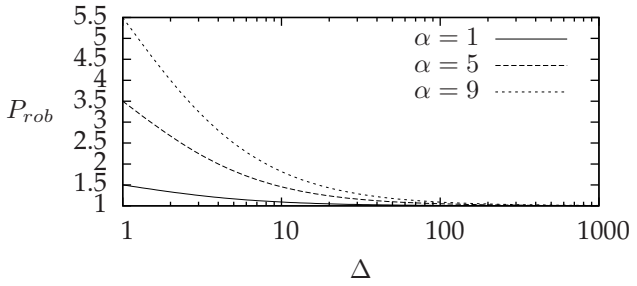

Fig. 5. Theoretical lower bounds to P_{rob}.

the inferred properties do not change. The full set of results can be found in [1]. All the experiments have been carried out on a workstation equipped with a 2,66 GHz Intel Core2 processor, 8Gb RAM, Linux (kernel 2.6.27) and gcc 4.3.3 compiler.

In the obtained diagrams, the values of the objective function f of the robust problem are compared to the optimum, i.e. the value of f given by the non-robust problem. As Δ increases, the curves tend rapidly to the optimum. For small values of α, the price of robustness is very low.

In order to compare the experimentally computed values of P_{rob} with the theoretical lower bounds given by Theorem 5.8, we provide Figure 5 which shows the values of function $1 + \frac{\alpha}{\Delta+1}$ for $\alpha \in \{1, 5, 9\}$ and $\Delta \in [1, \ldots, 1000]$. Note that, the computed values of P_{rob} are always smaller than the theoretical lower bounds as the latter are given for the worst case instances.

Concerning the diagrams representing the computational times, we can see that our tests required a very small amount of time. The linear growth of the curves as Δ increases is evident. For practical purposes, our experiments show that algorithm `SA-DP_BUILD` can be safely applied without requiring ages of computation.

6.1.1 *Corridor* `BrBo` *(see Figure 6)*

This corridor is quite large in terms of served stations and passing trains as shown in Table 1. We can see that the price of robustness is very close to 1 when $\alpha = 1$ while it is almost 1.5 when considering big delays of $\alpha = 9$ and $\Delta = 1$.

When $\Delta = 1$, the algorithm adds one slack time for each pair of consecutive arcs. As shown in Figure 6, the value of P_{rob}, when $\Delta = 1$, is about $\frac{2L_{avg}+\alpha}{2L_{avg}} = 1 + \frac{\alpha}{2L_{avg}}$, where L_{avg} is the average activity time.

It is also interesting to note how the values of f and P_{rob} decrease quickly with Δ. In particular, the price of robustness is 1 when $\Delta = 136$ and $\alpha = 9$, that is that with $\Delta = 136$ we do not have to pay for introducing robustness. As a special case, we mention that the price of robustness is between 1.00754 and 1.06785, when $\Delta = 11$. This implies that adding robustness reflects an increasing in the costs of just $0.7 - 6.7\%$ even for a small value of Δ.

Regarding the computational time, we can see that it increases with Δ but it is less than 5 milliseconds in the worst case (i.e. when $\alpha = 9$ and $\Delta = 136$). In detail, in the worst case, for $\Delta = 136$ we need about 5.01 milliseconds to achieve a price of robustness of 1.

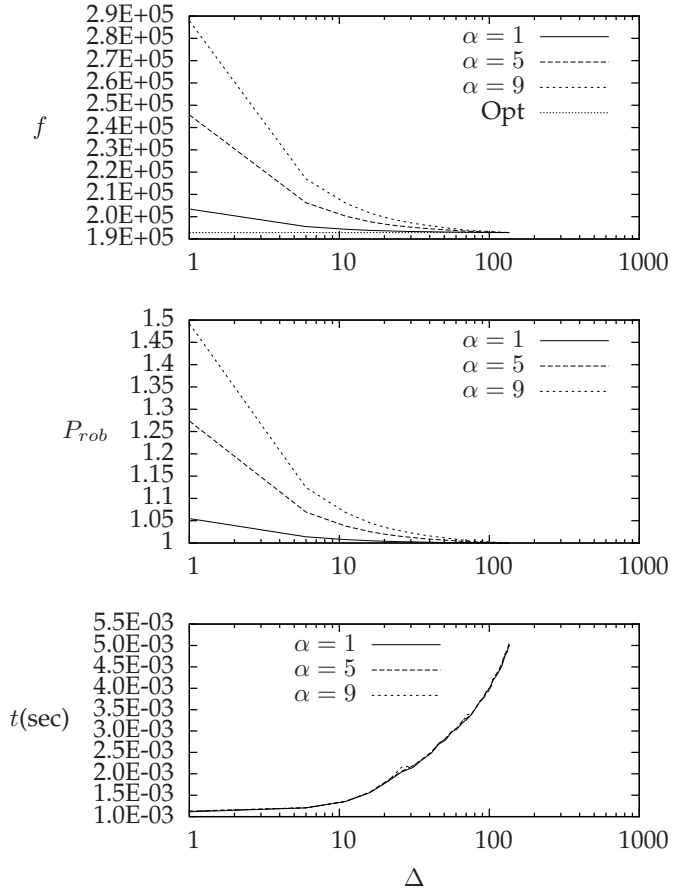

Fig. 6. Corridor BrBo with real values of minimum activity times

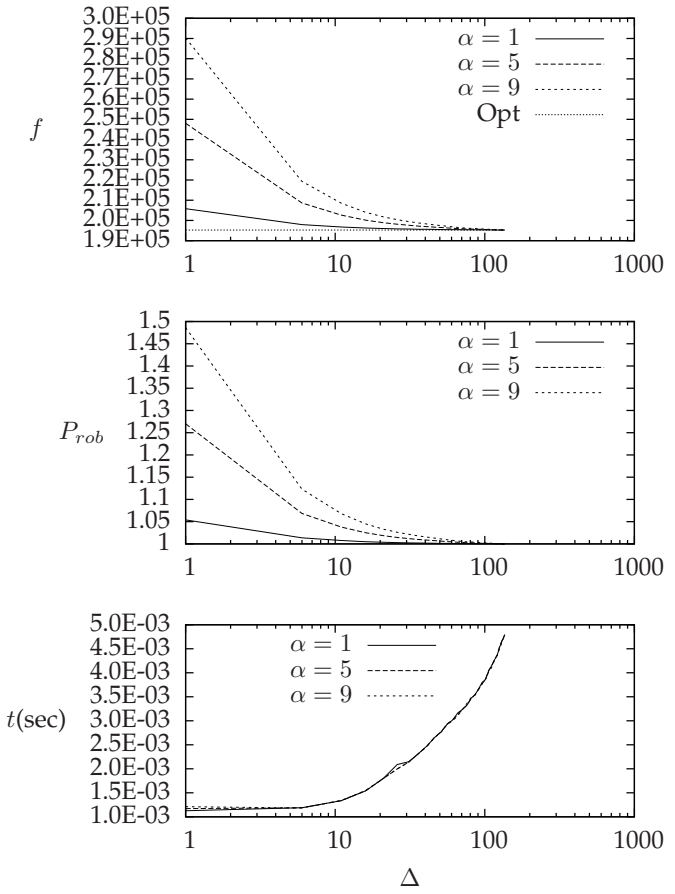

Fig. 7. Corridor BrBo with average activity times

6.1.2 *Corridor* BrBo *(average activity times) (see Figure 7)*

In this case, the objective function assumes almost identical values with respect to the previous case. As we expect, the value of the objective function does not depend on the value of L, but only on the structure of the tree and on the size of the delay.

Regarding P_{rob}, its value strictly depends on α/L_{avg} which can be considered a parameter for evaluating the magnitude of a delay. It is worth noting that for $\Delta = 1$ an optimal robust timetable has to assign exactly one slack time of size α for each pair of consecutive activities. It follows that, when $\alpha = 9$ and the average activity time is equal to 9, the price of robustness is 1.5, as it can be verified in Figure 7. The same happens for $\alpha = 5$, where the expected value is about 1.27.

6.1.3 *Corridor* MdMi *(see Figure 8)*

This corridor is the biggest in terms of served stations and passing trains. As shown in Table 1, the number of considered trains is more than four times the one in BrBo, while the number of stations is slightly more. Still, we can see comparable performances with respect to the price of robustness (it is 1 for each $\Delta \geq 966$) even though the incidence of the required computational time becomes more evident. However, as the timetables are calculated at the planning phase and not at runtime, the required time is still of an acceptable order being about 192.88 milliseconds in the worst case.

6.1.4 *Corridors* BzVr *and* PzBo *(see Figures 9 and 10)*

As expected, for the small corridors BzVr and PzBo, the price of robustness tends to the optimum much faster than for the other cases. Moreover the time required for the computations is negligible (in the worst cases, it is 1.41 milliseconds for BzVr and 0.27 milliseconds for PzBo).

For corridor BzVr, the price of robustness for small values of Δ is high, whereas, it is small for corridor PzBo. This is due to the fact that in the former case the average activity time is much smaller than the value of α, while in the latter case the average activity time is always greater than α.

6.2 Randomly generated data

By analyzing the results in the previous section, it is worth noting that the price of robustness tends to 1 faster than one would expect by the theoretical bounds (see Theorem 5.8 and Figure 5). This suggests that those instances have some hidden properties. One cause might

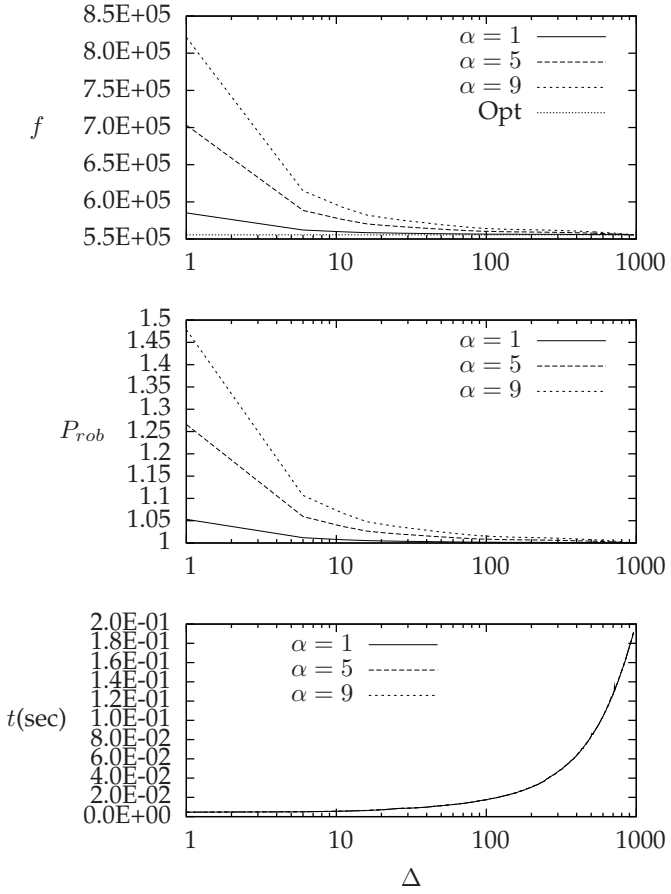

Fig. 8. Corridor `MdMi`

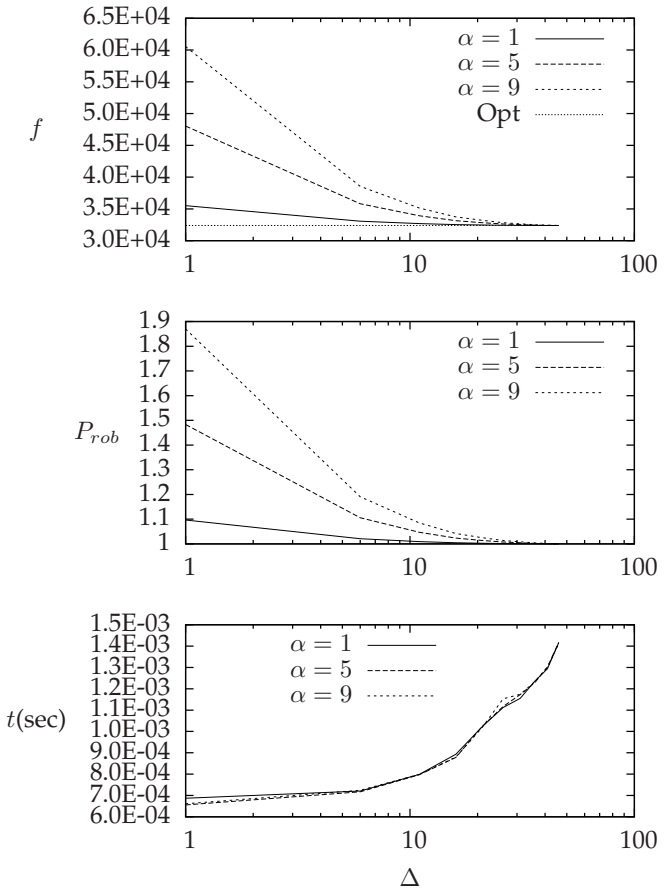

Fig. 9. Corridor `BzVr`

be the almost linearity of the tree structure, that is, the trees are made of long paths and the nodes have low outdegree.

In order to investigate on this matter, we test the behavior of the algorithm on three sets of five randomly generated trees: `Random1000`, `Random3000` and `Random5000`, containing five trees of 1000, 3000 and 5000 nodes, respectively. Each tree is generated starting from a single node and then by linking a new generated node to an existing one extracted uniformly at random. The node weights randomly rank between 1 and 10, and the minimum duration time for each activity randomly ranks between 1 and 18. In this way, the average activity duration time is comparable with that of the presented real-world instances. Finally, $\Delta \in [1, 2, \ldots, 10000]$ and $\alpha \in \{1, 5, 9\}$. For each pair (Δ, α), we performed one test for each randomly generated tree.

In Figures 11, 12, and 13, we summarize the obtained results. In particular, we show the average values of the price of robustness and the computational time, and the standard deviation of the price of robustness. The obtained results confirm our intuition that the almost linear structure of the real-world data heavily influences the curve of the price of robustness, while the computational time is not affected.

This is evident if we compare the results of graphs in `Random1000` (Figure 11) with those of corridor `BrBo` (Figures 6) which have comparable size. In fact, we can see that the computational time is almost identical (when $\Delta = 136$, it is 3.53 milliseconds for `Random1000` and 5.01 milliseconds for `BrBo`) but the price of robustness of `Random1000` is 1 only if $\Delta \geq 876$, while for `BrBo` the price of robustness is 1 for each $\Delta \geq 136$.

The same observation can be done by comparing `Random5000` with corridor `MdMi`. In this case, the price of robustness of `MdMi` is 1 for each $\Delta \geq 966$ while the price of robustness of `Random5000` is 1 only if $\Delta \geq 3746$. The computational times are still very close being 192.88 milliseconds for `MdMi` and 162.04 milliseconds for `Random5000`, when $\Delta = 966$.

7 Conclusion

We have presented the problem of planning robust timetables when the input event activity network topology is a tree. The delivered timetables can cope with one possible delay that might occur at runtime among the scheduled activities. In particular, our algorithms ensure that if a delay occurs, no more than Δ activities are affected by the propagation of such a delay. We have proved that the problem is NP-hard but not in the strong sense as it is pseudo-polynomially solvable.

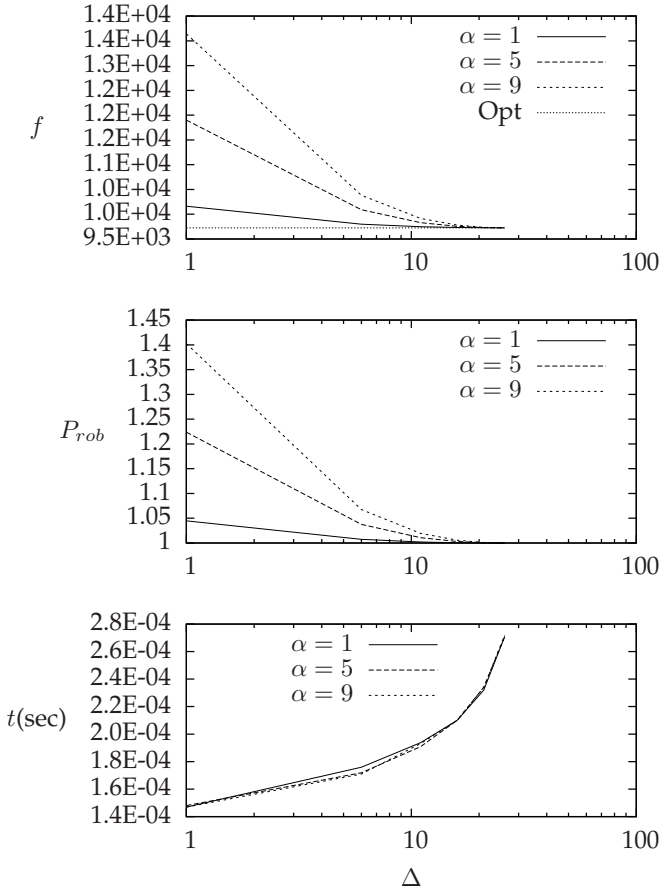

Fig. 10. Corridor PzBo

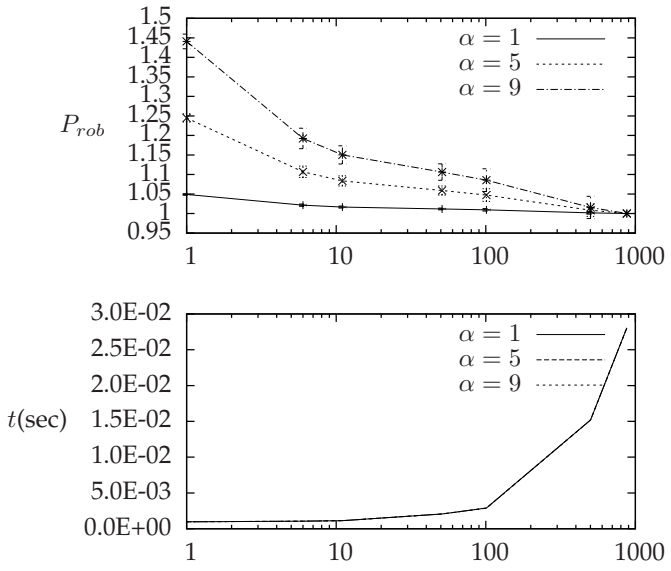

Fig. 11. Random1000: randomly generated trees with 1000 nodes

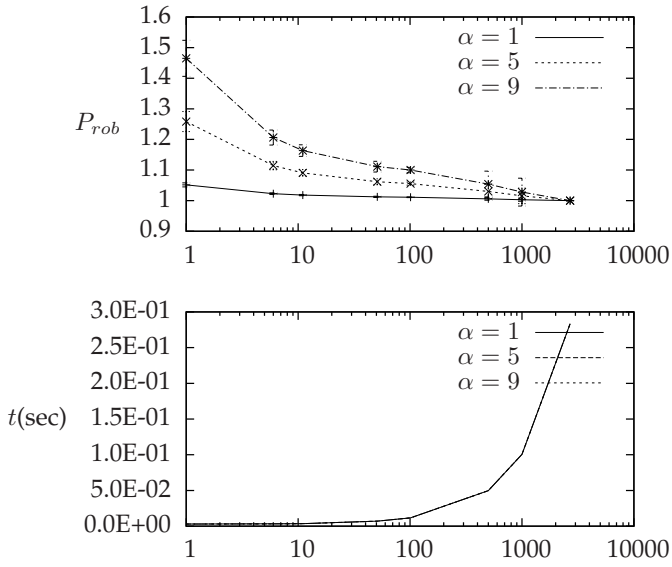

Fig. 12. Random3000: randomly generated trees with 3000 nodes

Although the combinatorial optimization problem arisen by our study is of its own interest, the paper continues the recent study on robust optimization theory. This is an important rising field led by the necessity of managing unpredictable limited disruptions with limited resources.

Several directions for future works deserve investigation such as the analysis of different recovery strategies, the application of other modification functions to the expected input and the enforcement of the recoverable robustness to other fundamental optimization problems.

REFERENCES

[1] http://informatica.ing.univaq.it/misc/TimetablingTree2009/.

[2] S. Cicerone, G. D'Angelo, G. Di Stefano, D. Frigioni, and A. Navarra. Robust Algorithms and Price of Robustness in Shunting Problems. In *Proceedings of the 7th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS07)*, pages 175–190, 2007.

[3] S. Cicerone, G. D'Angelo, G. Di Stefano, D. Frigioni, and A. Navarra. Delay Management Problem: Complexity Results and Robust Algorithms. In *Proceedings of the 2nd Annual International Conference on Combinatorial Optimization and Applications (COCOA)*, volume 5165 of *Lecture Notes in Computer Science*, pages 458–468. Springer, 2008.

[4] S. Cicerone, G. D'Angelo, G. Di Stefano, D. Frigioni, and A. Navarra. Recoverable robust timetabling: Complexity results and algorithms. *Journal of Combinatorial Optimization*, 18(3):229–257, 2009.

[5] S. Cicerone, G. D'Angelo, G. Di Stefano, D. Frigioni, A. Navarra, M. Schachtebeck, and A. Schöbel. *Robust and Online Large-Scale Optimization – Models and Techniques for Transportation Systems*, volume 5868 of *Lecture Notes in Computer Science*, chapter Recoverable Robustness in Shunting and Timetabling, pages 28–60. Springer Berlin, 2009.

[6] G. D'Angelo, G. Di Stefano, and A. Navarra. Evaluation of recoverable-robust timetables on tree networks. In *Proceedings of the 20th International Workshop on Combinatorial Algorithms (IWOCA)*, volume 5874 of *Lecture Notes in Computer Science*, pages 24–35. Springer, 2009.

[7] G. D'Angelo, G. Di Stefano, and A. Navarra. Recoverable-robust timetables for trains on single-line corridors. In *Proceedings of the 3rd International Seminar on Railway Operations Modelling and Analysis (RailZurich)*, 2009.

[8] G. D'Angelo, G. Di Stefano, A. Navarra, and Cristina M. Pinotti. Recoverable Robust Timetabling on Trees. In *Proceedings of the 3rd*

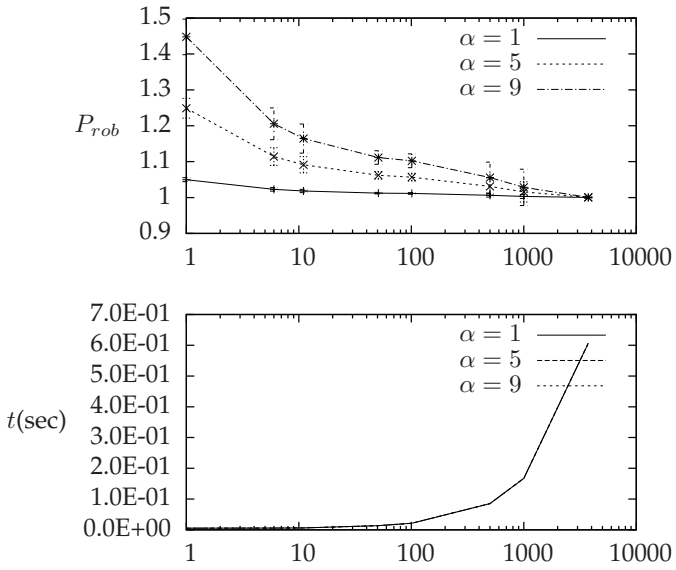

Fig. 13. `Random5000`: randomly generated trees with 5000 nodes

Annual International Conference on Combinatorial Optimization and Applications (COCOA), volume 5573 of *Lecture Notes in Computer Science*, pages 451–462. Springer, 2009.

[9] L. De Giovanni, G. Heilporn, and M. Labbé. Optimization models for the delay management problem in public transportation. *European Journal of Operational Research*, 189(3):762–774, 2007.

[10] M. R. Garey and D. S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York, 1979.

[11] M. Gatto, R. Jacob, L. Peeters, and P. Widmayer. Online Delay Management on a Single Train Line. In Frank Geraets, Leo Kroon, Anita Schöbel, Dorothea Wagner, and Christos D. Zaroliagis, editors, *Proceedings of the 4th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS04)*, volume 4359 of *Lecture Notes in Computer Science*, pages 306–320, 2007.

[12] A. Ginkel and A. Schöbel. The bicriteria delay management problem. *Transportation Science*, 41(4):527–538, 2007.

[13] F.K. Levy, G.L. Thompson, and J.D. Wies. *The ABCs of the Critical Path Method*. Graduate School of Business Administration. Harvard University, 1963.

[14] C. Liebchen, M. Lüebbecke, R. H. Möhring, and S. Stiller. *Robust and Online Large-Scale Optimization – Models and Techniques for Transportation Systems*, volume 5868 of *Lecture Notes in Computer Science*, chapter The Concept of Recoverable Robustness, Linear Programming Recovery, and Railway Applications, pages 1–27. Springer, 2009.

[15] A. Schöbel. A model for the delay management problem based on mixed integer programming. *Electronic Notes on Theoretical Computer Science*, 50(1):1–10, 2004.

[16] A. Schöbel. Integer programming approaches for solving the delay management problem. In Frank Geraets, Leo Kroon, Anita Schöbel, Dorothea Wagner, and Christos D. Zaroliagis, editors, *Proceedings of the 4th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS04)*, volume 4359 of *Lecture Notes in Computer Science*, pages 145–170, 2007.

[17] P. Serafini and W. Ukovich. A Mathematical Model for Periodic Scheduling Problems. *SIAM J. Discrete Math.*, 2(4):550–581, 1989.

[18] Trenitalia. http://www.trenitalia.com/.

Gianlorenzo D'Angelo Gianlorenzo D'Angelo is currently a Ph.D. student in the Department of Electrical and Information Engineering of the University of L'Aquila. His research interests are in the area of combinatorial optimization, robust optimization, and graphs algorithms.

Gabriele Di Stefano Prof. Dr. Gabriele Di Stefano obtained his Ph.D. at University "La Sapienza" of Rome in 1992. Currently he is associate professor for computer science at the University of L'Aquila; his current research interests include network algorithms, combinatorial optimization, algorithmic graph theory; he is (co-)author of more than 60 publications in journals and international conferences. He had key-participations in several EU funded projects. Among them: MILORD (AIM 2024), COLUMBUS (IST 2001-38314), AMORE (HPRN-CT-1999-00104), and, recently, ARRIVAL (IST FP6-021235-2).

Alfredo Navarra Alfredo Navarra is an Assistant Professor at the University of Perugia, Italy. He received his Ph.D. in Computer Science in 2004 from "Sapienza" University of Rome. In 2003 and 2004 he has joint the MASCOTTE project team at the INRIA institute of Sophia Antipolis. In 2005, he was with the Dept of Computer Science at the Univ. of L'Aquila. In 2006, he has joint the LaBRI, Univ. of Bordeaux. In 2007, he was with the Dept of Electrical and Information Engineering at Univ. of L'Aquila. His research interests include algorithms, computational complexity, networking and distributed computing.

M. Cristina Pinotti M. Cristina Pinotti received the Dr. degree cum laude in Computer Science from the University of Pisa, Italy, in 1986. From 1987-1999, she was a researcher with the National Council of Research in Pisa. From 2000-2003, she was an associate professor at the University of Trento. Since 2004, she is a full professor at the University of Perugia. Her current research interests include sensor networks, wireless networks, design and analysis of combinatorial algorithms. Since August 2009, she serves as an associate editor for IEEE Transactions on Parallel and Distributed Systems.