Recoverable Robust Timetables on Trees*, **

Gianlorenzo D'Angelo[1], Gabriele Di Stefano[1], Alfredo Navarra[2], and Cristina M. Pinotti[2]

[1] Department of Electrical and Information Engineering, University of L'Aquila.
{gianlorenzo.dangelo,gabriele.distefano}@univaq.it
[2] Department of Mathematics and Informatics, University of Perugia.
navarra@dipmat.unipg.it, pinotti@unipg.it

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 the activities from scratch. This realizes the concept of designing so called *robust timetables*. During the planning phase, one has to consider recovery features that might be applied at runtime if disruptions occur. Such recovery capabilities are given as input along with the possible disruptions that must be considered. As in standard *delay management* problems, the main objective still remains the minimization of the overall needed time. We show that finding an optimal solution for this problem is NP-hard even though the topology of the event activity network, which models dependencies among activities, is restricted to trees. However, we manage to design a pseudo-polynomial time algorithm.

1 Introduction

When dealing with many real world applications, the design of a solution can be 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

* 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).
** Practical applications of some results contained in this paper can be found in [1].

of the two phases. A concrete example of real world systems, where this interaction is important, is the *timetable planning* in railways systems. It arises in the strategic planning phase of railway systems, and it 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 loose 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 in the literature as *delay management problem* [2–5]. Although 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 [2, 4–7]).

In order to cope with the management of delays we follow the recent *recoverable robust optimization* approach provided in [8, 9], 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 disruptions. If a disruption (delay) occurs, the designed timetable should guarantee to recover the scheduled events by means of the allowed operations represented by given recovery algorithms. Events and dependencies among events are modeled by means of an *event activity network* (see [2, 5]). 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 or changing for another train). In this paper, we assume that 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 in a fixed amount of steps, 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 activities. The objective function is then to minimize the total time required by the events in order to serve all the scheduled activities and to be robust with respect to one possible delay of time α. The challenging combinatorial problem arising by those restrictions is of its own interest. We restrict our attention to event activity networks whose topology is provided by a tree. In fact, in [2], the authors show that the described problem is NP-hard when the event activity network topology is a DAG, and they provide algorithms which

cope with the case of $\Delta = 0$. In [10], the authors provide algorithms for a generic Δ when the event activity network is a linear graph.

Our Results. 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. Then we present algorithmic results. We provide an algorithm that solves the problem in $O(\Delta^2 n)$ time where n is the number of events in the input event activity network. The result implies that the problem can be solved in pseudo-polynomial time. The algorithm exploits the tree topology in order to chose which arc must be associated with some slack time. Intuitively, on trees, 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 shows that a solution with two consecutive activities associated both with a slack time either is not optimal or it can be turned into an optimal solution without such occurrence, unless $\Delta = 0$.

Due to space restrictions, omitted proofs can be found in Appendix.

2 Recoverable Robustness Model

In this section, we summarize the model of recoverable robustness given in [8]. 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, and let $\pi \in S$ be the planned solution for I. A *disruption* is meant as a modification to the input i. Hence, given $i \in I$, $M(i)$ 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 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. *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.

Definition 3. *Let $\mathcal{P} = (P, M, \mathbb{A}) \in$ RRP. A* robust algorithm *for $\mathcal{P}$ is any algorithm A_{rob} such that, for each $i \in I$, $A_{rob}(i)$ is a robust solution for i with respect to P.*

The quality of a robust solution is measured by the *price of robustness* as in the following definition.

Definition 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: $P_{rob}(\mathcal{P}, A_{rob}) = \max_{i \in I} \left\{ \frac{f(A_{rob}(i))}{\min\{f(x) : x \in F(i)\}} \right\}$.*
- *The* price of robustness *of $\mathcal{P}$ is: $P_{rob}(\mathcal{P}) = \min\{P_{rob}(\mathcal{P}, A_{rob}) : A_{rob}$ 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: $f(\pi) = \min\{f(\pi') : \pi'$ is feasible for $\mathcal{P}\}$.*

3 Robust Timetabling Problem

In this section, we first consider a particular timetable problem and then, according to the model of Section 2, we turn it into a recoverable robustness problem, the *Robust Timetabling* problem ($\mathcal{RTT}$).

Given a DAG $G = (V, A)$, where the nodes represent events and the arcs represent the activities, 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 $a = (u, v) \in A$.

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

TT	
GIVEN:	A DAG $G = (V, A)$, a function $L : A \to \mathbb{N}$ and a function $w : V \to \mathbb{R}_{\geq 0}$.
PROBLEM:	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: $F(i) = \{\Pi : \Pi(u) \in \mathbb{R}_{\geq 0},\ \forall u \in V$ 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* $s(a) = \Pi(v) - \Pi(u) - L(a)$ 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 we restrict to the TT problem where the DAG is an out-tree $T = (V, A)$, any feasible solution satisfies $\Pi(v) \geq \Pi(r) + \sum_{a \in P(r,v)} L(a)$, 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 fix 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 1. *Given a tree $T = (V, A)$ and an instance $i = (T, L, w)$ of TT, if $i' = (T, L', w)$ where $L'(a) = 1$, $\forall a \in A$, then for any feasible solution Π of i, there exists a feasible solution Π' of 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: $\Pi(v) = \Pi(r) + \sum_{a \in P(r,v)}(L(a) + s(a))$. We define Π' as $\Pi'(r) = \Pi(r)$ and $\Pi'(v) = \Pi'(r) + \sum_{a \in P(r,v)}(1 + s(a))$. Note that, Π' is feasible for i'. The values of the objective functions of Π and Π' are $f(\Pi) = \Pi(r)w(r) + \sum_{v \in V} \Pi(v)w(v) = \Pi(r)w(r) + \sum_{v \in V} \sum_{a \in P(r,v)}(L(a) + s(a))w(v)$ and $f(\Pi') = \Pi'(r)w(r) + \sum_{v \in V} \Pi'(v)w(v) = \Pi'(r)w(r) + \sum_{v \in V} \sum_{a \in P(r,v)}(1 + s(a))w(v)$, respectively. Hence, $f(\Pi) = f(\Pi') + \sum_{v \in V} \sum_{a \in P(r,v)} w(v)(L(a) - 1)$. □

TT can be solved in linear time by assigning the minimal possible time to each event (i.e. by using the Critical Path Method [11]). However, such a solution cannot always cope with possible delays occurring at running time to the activities. Recovery (on-line) 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 $\alpha \in G$, 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* as follows.

Definition 5. *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 $a = (u, v) \in A$ (equivalently, 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 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 are smaller than α. That is, the planned duration of the activities are not able to absorb the delay α, which cannot be hidden from x .

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

Moreover, we concentrate our work on the $\mathcal{RTT}_{\Delta}$ problem, where the DAG is an out-tree. Formally, the above optimization problem, denoted by $\mathcal{RTT}_{\Delta,opt}$, is defined as follows:

$\mathcal{RTT}_{\Delta,opt}$	
GIVEN:	A tree $T = (V, A)$, a function $w : V \to \mathbb{R}_{\geq 0}$, and two numbers α and $\Delta \in \mathbb{N}^+$.
PROBLEM:	Find 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 $s : A \to \mathbb{R}_{\geq 0}$ defined as $s(a = (i, j)) = \Pi(j) - \Pi(i) - 1$, and such that $f(\Pi) = \sum_{v \in V} \Pi(v)w(v)$ 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 $\mathcal{RTT}_{\Delta,opt}$ problem which assigns only slack times equal to α.

Lemma 2. *Given an instance i of $\mathcal{RTT}_{\Delta}$, for each solution Π for i there exists a solution Π' for i such that $f(\Pi') \leq f(\Pi)$ and, for each arc $a = (x, y)$, either $\Pi'(y) = \Pi'(x) + 1$ or $\Pi'(y) = \Pi'(x) + 1 + \alpha$.*

Next, let us analyze the complexity of the $\mathcal{RTT}_{\Delta,opt}$ problem, with $\Delta \geq 1$. In fact, observe that for the case $\Delta = 0$, the $\mathcal{RTT}_{0,opt}$ problem is optimally solved in linear time by adding slack times equal to α for each arc of the tree.

Theorem 1. *The $\mathcal{RTT}_{\Delta,opt}$ problem, with $\Delta \geq 1$, is NP-hard.*

Theorem 1 follows from the fact that the decisional version of the $\mathcal{RTT}_{\Delta,opt}$ problem, with $\Delta \geq 1$, is NP-complete. This can be found in the Appendix.

4 Pseudo-Polynomial Time Algorithm

Based on the dynamic programming techniques, in this section we devise a pseudo-polynomial time algorithm for $\mathcal{RTT}_{\Delta,opt}$, with $\Delta \geq 1$.

At first, let us introduce few notation. From now on, let $\mathcal{RIT}_{\Delta}$-optimal denote a solution for $\mathcal{RIT}_{\Delta,opt}$. Let $T = (V, A)$ be an arbitrarily ordered 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 subtree of T rooted in v and let $T_i(v)$ denote the 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, for each $v \in V$, the values $|T(v)|$ and $c(v)$ can be computed in linear time by visiting $T(r)$ where r is the radix of T.

In order to characterize a $\mathcal{RIT}_{\Delta}$-optimal solution Π, we need the following lemmata and definition.

Lemma 3. *Given a tree T, consider two feasible solution Π' and Π of cost f' and f, respectively which differ only for their slack time on the arc $a = (u, v)$. Let Π' have $s(a) = 0$, and Π have $s(a) = \alpha$. Then, $f' = f - \alpha c(v)$.*

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 a. For each node $x \in T(v)$ then, it holds $\Pi'(x) = \Pi(x) - \alpha$. Hence, $f' = f - \alpha c(v)$. □

Definition 6. *Given a feasible solution Π for the RDM_{Δ} problem and a node $v \in V$, a* ball *$B_{\Pi}(v)$ is the maximal subtree rooted in v such that each node in $B_{\Pi}(v)$ has its incoming arc a with $s(a) = 0$.*

Note that if the incoming arc into v has its slack time equal to α, the ball $B_{\Pi}(v)$ is empty. In other words, the ball $B_{\Pi}(v)$ represents the set of nodes in solution Π which are affected by the delay occurring at the arc incoming to v. Due to the feasibility of Π, no more than Δ nodes can belong to $B_{\Pi}(v)$.

Lemma 4. *For each instance of $\mathcal{RIT}_{\Delta}$, there exists a $\mathcal{RIT}_{\Delta}$-optimal solution such that for each $v \in V$, $B_{\Pi}(v)$ cannot be extended by adding any node in $N_o(B_{\Pi}(v))$ while keeping feasibility.*

Proof. By contradiction, we assume that there exists an instance such that, for each $\mathcal{RIT}_{\Delta}$-optimal solution Π' there exists a non empty set of nodes $\mathcal{V}$ which contradict 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 $d(r, v)$ 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, the solution Π'' that assigns $\Pi''(y) = \Pi'(x) + 1$, $\Pi''(y_i) = \Pi''(y) + 1 + \alpha$, for $1 \leq i \leq |N_o(y)|$, and keeps the rest of the original solution is feasible. Indeed, $B_{\Pi''}(v) = B_{\Pi'}(v) + 1 \leq \Delta$ for each v such that $x \in B_{\Pi'}(v)$ and $B_{\Pi''}(y) \leq B_{\Pi'}(y) \leq \Delta$. By Lemma 3, $f(\Pi'') = f(\Pi') - \alpha c(y) + \sum_{i=1}^{|N_o(y)|} \alpha c(y_i)$.

Observing that only the slack times of the arcs incoming into y_i and y have been changed, and since $\sum_{i=1}^{|N_o(y)|} c(y_i) \leq c(y)$ because $\cup_{i=1}^{|N_o(y)|} T(y_i) \subset T(y)$, it holds $f(\Pi'') \leq f(\Pi')$. This procedure is repeated until $B_\Pi(v)$ becomes maximal, i.e., of size $\min\{\Delta, |T(v)|\}$.

At the end of this process, the obtained solution is optimal as $f(\Pi'') \leq f(\Pi')$ and none of the balls can be extended by adding nodes while keeping feasibility, a contradiction. □

Lemma 5. *For each instance of $\mathcal{RTT}_\Delta$, with $\Delta \geq 1$, there exists a $\mathcal{RTT}_\Delta$-optimal solution such that at most one of two consecutive arcs has a slack time of α.*

Proof. Suppose, by contradiction, that each $\mathcal{RTT}_\Delta$-optimal solution Π' assigns a slack time to both of two consecutive arcs, i.e. there exist $(x, y), (y, z) \in A$ such that $\Pi'(y) = \Pi'(x) + 1 + \alpha$ and $\Pi'(z) = \Pi'(y) + 1 + \alpha$. Then, we can extend the ball rooted in y by defining a solution Π'' such that $\Pi''(z) = \Pi'(y) + 1$, $\Pi''(z_i) = \Pi''(z) + 1 + \alpha$, $1 \leq i \leq |N_o(z)|$, and keeping the rest of the original solution. Π'' is feasible because $B_{\Pi''}(z) = 1 \leq \Delta$. By Lemma 3, $f(\Pi'') = f(\Pi') - \alpha c(z) + \sum_{i=1}^{|N_o(z)|} \alpha c(z_i)$. Since only the slack times of the arcs incoming into z_i and z have been changed, and since $\sum_{i=1}^{|N_o(z)|} \alpha c(z_i) \leq \alpha c(z)$, it holds $f(\Pi'') \leq f(\Pi')$. Therefore, Π'' is feasible and $f(\Pi'') \leq f(\Pi')$, a contradiction. □

Let us now consider an optimal solution Π for the RDM_Δ problem on T. Since the root r of T has no incoming edges, it cannot be affected by any delay. In other words, r can be considered as a node reached by an edge associated with a slack time of α, hence, by Lemma 5, for each arc a outgoing from the root r, $s(a) = 0$. By Lemma 4, for each $v \in N_o(r)$, Π induces a maximal ball $B_\Pi(v)$ of size $\min\{|T(v)|, \Delta\}$.

In order to compute the $\mathcal{RTT}_\Delta$-optimal solution Π, by applying dynamic programming techniques, we obtain an algorithm which requires $O(\Delta^2 n)$ time complexity and $O(\Delta n)$ space. Such algorithm, called *SA-DP*$_\Delta$ and illustrated in Figure 2, considers an arbitrarily ordered tree T in input. and performs a visit in post-order of T. It uses for each node $v \in T$ two matrices G_v and SOL_v of dimensions $(N_o(v) + 1) \times (\Delta + 1)$.

Let f^* denote the cost of an $\mathcal{RTT}_0$-optimal solution on T, that is, the cost of the solution which associates to each arc a slack time of α. The *SA-DP*$_\Delta$ algorithm stores in each entry $G_v[i, j]$ the maximum gain for a solution Π with respect to f^* achievable by constructing a ball $B_\Pi(v)$ of size j when considering only the first i children of v. As $G_v[i, j]$ must be the maximum gain, solution Π must be optimal with respect to $T_i(v)$.

In practice, for each node v and each integer $j \leq \Delta$, we have to decide the most profitable way of building a ball (i.e., a maximal subtree) of size j, rooted at v. Since any node v belonging to the ball gains a profit of $\alpha c(v)$, independent of the shape of the ball itself, each ball can be easily decomposed in maximal sub-balls, possibly empty, rooted at the children $v_1, v_2, \ldots, v_{|N_o(v)|}$ of v. Moreover, note that, by Lemma 4, if $j < |T(v)|$, the maximum gain is obtained by setting $B_\Pi(v) \equiv T(v)$.

For a generic internal node v of T, in Figure 1 shows the possible configurations to consider when evaluating $G_v[i, j]$.

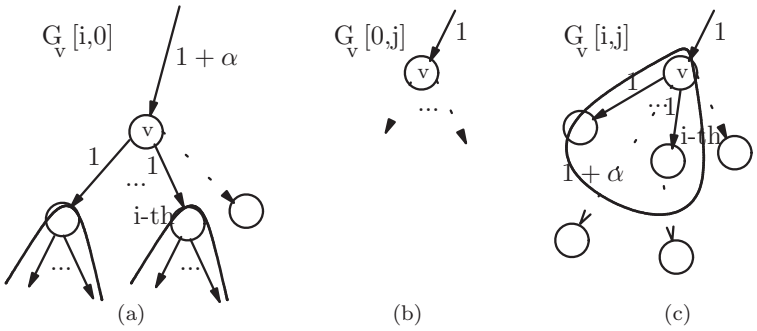

Fig. 1. The three possible configurations that must be considered when computing matrix G_v.

If $j = 0$ (see Fig. 1(a) and Line 5 of the algorithm), then the entry $G_v[i, 0]$ is defined as the maximum gain of a solution Π with respect to f^* achievable by an empty ball $B_\Pi(v)$. Thus, solution Π must have the slack time on the arc incoming into v set to α. Therefore, by Lemma 5, all the arcs outgoing from v have a slack time equal to 0, and so the optimal solution Π with respect to $T_i(v)$ in each subtree rooted at the child v_ℓ, $1 \leq \ell \leq i$, of v has a ball $B_\Pi(v_\ell)$ of maximum possible size, as proved in Lemma 4. Specifically, $B_\Pi(v_\ell)$ has size Δ if $\Delta < |T(v_\ell)|$, and size $|T(v_\ell)|$ otherwise. Hence, the overall gain $G_v[i, 0]$ with respect to f^* is recursively computed as the sum of the gains achieved at the first i children of v, that is, $G_v[i, 0] = \sum_{\ell=1}^{i} G_{v_\ell}[|N_o(v_\ell)|, \Delta]$.

If $i = 0$ and $j > 0$ (see Fig. 1(b) and Line 7 of the algorithm), then the entry $G_v[0, j]$ is defined as the maximum gain for a solution Π with respect to f^* achievable by constructing a ball $B_\Pi(v)$ of size j without considering the children of v, that is $|B_\Pi(v)| = 1$ independent of the specified value j. By Lemma 3, $G_v[0, j] = \alpha c(v)$.

If $i > 0$ and $j > 0$ (see Fig. 1(c) and Line 10 of the algorithm), then the entry $G_v[i, j]$ takes its most general meaning. Since $j > 0$, the considered solution Π sets the slack time incoming into v to 0. Exploiting the sub-optimality property, $B_\Pi(v)$ is built recursively on the subtrees $T_{i-1}(v)$ and $T(v_i)$ where two balls of complementary sizes (i.e., whose sizes sum up to j) are considered. Observe that for a chosen size s of $B_\Pi(v_i)$, the gain is recursively computed as $G_v[i - 1, j - s] + G_{v_i}[|N_o(v_i)|, s]$. Thus, $G_v[i, j]$ is determined by considering all the possible sizes s, with $0 \leq s \leq j - 1$, of $B_\Pi(v_i)$ when considering only the first i children of v. Note that since $j > 0$, the incoming arc into v has the slack time set to 0, and hence $B_\Pi(v_i)$ cannot have size larger than $j - 1$. Formally, $G_v[i, j] = \max_{0 \leq s < j} \{G_v[i - 1, j - s] + G_{v_i}[|N_o(v_i)|, s]\}$.

Algorithm $\texttt{SA-DP}_\Delta$

Input: $v \in V$

let $N_o(v) = \{v_1, \ldots, v_k\}$,

1. **for** $i = 1$ to k
2. $\quad \texttt{SA-DP}_\Delta(v_i)$
3. **for** $i = 0$ **to** k
4. $\quad$ **for** $j = 0$ **to** Δ
5. $\quad\quad$ **if** $j == 0$ **then** $G_v[i,0] = \sum_{\ell=1}^{i} G_{v_\ell}[|N_o(v_\ell)|, \Delta]$
6. $\quad\quad$ **else**
7. $\quad\quad\quad$ **if** $i == 0$ **then** $G_v[i,j] = \alpha c(v)$;
8. $\quad\quad\quad$ **else**
9. $\quad\quad\quad\quad$ **if** $j \leq \sum_{\ell=1}^{i} |T(v_\ell)| + 1$ **then**
10. $\quad\quad\quad\quad\quad G_v[i,j] = \max_{0 \leq s < j} \{G_v[i-1, j-s] + G_{v_i}[|N_o(v_i)|, s]\}$
11. $\quad\quad\quad\quad\quad$ let s^* be the index determining the maximum at Line 10, $SOL_v[i,j] = s^*$
12. $\quad\quad\quad\quad$ **else** $G_v[i,j] = G_v[i,j-1]$

Fig. 2.

For each node v, the $\texttt{SA-DP}_\Delta$ algorithm also memorizes in each matrix SOL_v the choices that lead to the optimal gains computed in the corresponding matrix G_v (see Line 11 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 $SOL_v[i,j] \neq 0$, it means that the arc (v, v_i) has no slack time, that is, it belongs to the ball $B_\Pi(v)$ of the optimal solution Π. Viceversa, if $SOL_v[i,j] = 0$, either the arc (v, v_i) has a slack time of α (i.e., $|B_\Pi(v_i)| = 0$) or such an arc does not exist (i.e., $i = 0$ or $j > |T(v)|$).

A call to Procedure $\texttt{SA-DP}_\Delta$ from a node v evaluates all the entries of the matrices associated to each node belonging to $T(v)$. As by Lemma 5, we know that the children of the root r must be all at distance 1 from r, we need $|N_o(r)|$ separate calls of the procedure in order to evaluate all the entries of the matrices, each one involving the subtree rooted at a child of r. Alternatively, the optimal gain for the entire tree T is computed in $G_r[|N_o(r)|, 0]$ by calling Procedure $\texttt{SA-DP}_\Delta$ on the root r of T.

Once all the matrices G_v and Sol_v have been computed, we have to show how to construct the optimal solution Π, i.e. how to assign to each $v \in T$ the value $\Pi(v)$. Recall first that, by Lemma 5, $\Pi(r) = 0$ and for each $v_\ell \in N_o(r)$, $1 \leq \ell \leq |N_o(r)|$, $\Pi(v_\ell) = \Pi(r) + 1 = 1$.

Moreover, a ball $B_\Pi(v_\ell)$ of size $\min = \{|T(v_\ell)|, \Delta\}$ is rooted at each $v_\ell \in N_o(r)$. Then, we need $|N_o(r)|$ separate calls of Procedure `BUILD`, each one involving the subtree rooted at a child of r, to find the value Π for each node in T. Specifically, for each child $v_\ell \in N_o(r)$, the procedure $\texttt{BUILD}(v_\ell, |N_o(v_\ell)|, \Delta)$, depicted in Figure 3, is invoked to build the corresponding $B_\Pi(v_\ell)$.

Procedure $\texttt{BUILD}(v, i, j)$ 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 of the value of $SOL_v[i,j]$.

Algorithm BUILD

Input: $v \in V$, $i \in [0, |N_o(v)|]$, $j \in [0, \Delta]$

1. **if** v is not a leaf **and** $i > 0$ **then**
2. **if** $SOL_v[i,j] > 0$ **then**
3. $s := SOL_v[i,j]$
4. $\Pi(v_i) := \Pi(v) + 1$
5. BUILD$(v_i, |N_o(v_i)|, s)$
6. BUILD$(v, i-1, j-s)$
7. **else**
8. **if** $i \leq |N_o(v)|$ **then**
9. $\Pi(v_i) := \Pi(v) + 1 + \alpha$
10. $w := v_i$
11. **for** $k = 1$ **to** $|N_o(w)|$
12. $\Pi(w_k) := \Pi(w) + 1$
13. BUILD$(w_k, |N_o(w_k)|, \Delta)$
14. BUILD$(v, i-1, j)$

Fig. 3.

If $SOL_v[i,j] > 0$, it means that 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 extend over $T(v_i)$ and it completely belongs to $T_{i-1}(v)$. Hoverer, since the slack time on the arc (v, v_i) is equal to α, by Lemma 5, all the arcs outgoing from v_i, say (v_i, w_k) for $1 \leq k \leq |N_o(v_i)|$, have slack time equal to 0. The balls of size at most Δ rooted at the nodes w_k, with $1 \leq k \leq |N_o(v_i)|$, are recursively built invoking $|N_o(v_i)|$ times Procedure BUILD, i.e. one for each child of v_i.

Theorem 2. *For $\Delta \geq 1$, SA-DP$_\Delta$ is correct.*

The proof of Theorem 2 follows by induction on Δ and by Lemmata 3, 4, and 5.

Theorem 3. $Prob(\mathcal{RIT}_\Delta, \text{SA-DP}_\Delta) \leq 1 + \frac{\alpha}{2}$.

Theorem 4. $Prob(\mathcal{RIT}_\Delta) \geq 1 + \frac{\alpha}{\Delta+1}$.

Theorem 5. *SA-DP$_\Delta$ requires $O(\Delta^2 n)$ time and $O(\Delta n)$ space. BUILD requires $O(n)$ time.*

5 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 in general, while it is pseudo-polynomially solvable.

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. D'Angelo, G., Di Stefano, G., Navarra, A.: Recoverable-robust timetables for trains on single-line corridors. In: Proceedings of the 3rd International Seminar on Railway Operations Modelling and Analysis - Engineering and Optimisation Approaches (RailZurich). (2009)
2. Cicerone, S., D'Angelo, G., Di Stefano, G., Frigioni, D., Navarra, A.: Delay Management Problem: Complexity Results and Robust Algorithms. In: Proc. of 2nd Annual International Conference on Combinatorial Optimization and Applications (COCOA). Volume 5165 of LNCS., Springer (2008) 458–468
3. De Giovanni, L., Heilporn, G., Labbé, M.: Optimization models for the delay management problem in public transportation. European Journal of Operational Research **189**(3) (2007) 762–774
4. Schöbel, A.: A model for the delay management problem based on mixed integer programming. Electronic Notes in Theoretical Computer Science **50**(1) (2004) 1–10
5. Schöbel, A.: Integer programming approaches for solving the delay management problem. In: Proc. of the Algorithmic Methods for Railway Optimization (ATMOS04). Volume 4359 of LNCS. (2007) 145–170
6. Gatto, M., Jacob, R., Peeters, L., Widmayer, P.: Online Delay Management on a Single Train Line. In: Proc. of the Algorithmic Methods for Railway Optimization (ATMOS04). Volume 4359 of LNCS. (2007) 306–320
7. Ginkel, A., Schöbel, A.: The bicriteria delay management problem. Transportation Science **41**(4) (2007) 527–538
8. Cicerone, S., D'Angelo, G., Di Stefano, G., Frigioni, D., Navarra, A.: Robust Algorithms and Price of Robustness in Shunting Problems. In: Proc. of the 7th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS). (2007) 175–190
9. Liebchen, C., Lübbecke, M., Möhring, R.H., Stiller, S.: Recoverable robustness. Technical Report ARRIVAL-TR-0066, ARRIVAL Project (2007)
10. Cicerone, S., Di Stefano, G., Schachtebeck, M., Schöbel, A.: Dynamic Algorithms for Recoverable Robustness Problems. In: Proc. of the 8th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS). (2008)
11. Levy, F., Thompson, G., Wies, J.: The ABCs of the Critical Path Method. Graduate School of Business Administration. Harvard University (1963)
12. Garey, M.R., Johnson, D.S.: Computers and Intractability, A Guide to the Theory of NP-Completeness. W.H. Freeman and Company, New York (1979)

Appendix

Proof (of Lemma 2). Given Π, we define $\Pi' = \Pi$ and then we perform iteratively the next two operations starting from the root, downward to the leaves until none of them can be applied.

1. For each arc $a = (x, y)$ such that $\Pi'(y) > \Pi'(x) + 1 + \alpha$, we assign $\Pi'(y) = \Pi'(x) + 1 + \alpha$;
2. For each arc $a = (x, y)$ such that $\Pi'(x) + 1 < \Pi'(y) < \Pi'(x) + 1 + \alpha$, we assign $\Pi'(y) = \Pi'(x) + 1$.

Clearly, at the end of the procedure, for each arc $a = (x, y)$, either $\Pi'(y) = \Pi'(x) + 1$ or $\Pi'(y) = \Pi'(x) + 1 + \alpha$. The feasibility of applying Rule 1 follows directly by the fact that any modification does not introduce delays larger than α. The feasibility of applying Rule 2 follows by noting that it only shifts downward positive slack times smaller than α. As the process is started from the root, at each step, if an arc $a = (x, y)$ is such that $\Pi'(x) + 1 < \Pi'(y) < \Pi'(x) + 1 + \alpha$, then either 1) it is used to absorb a portion of a possible delay as a last edge, hence successive nodes were not affected by the same delay; or 2) it is used to absorb a portion of a possible delay but not as a last edge, hence successive nodes are affected by the same delay; or 3) it is used for both cases 1) and 2) but then we consider it as just for case 1); or 4) it is not necessary. In case 1), by applying Rule 2 downward from the root, eventually a will be associated with a slack time of at least α (and Rule 1 might be applied on it), hence it is still able to absorb the original delay for which it was assigned and successive nodes are still not affected by the same delay. In case 2), shifting downward the slack time originally assigned to a implies that successive nodes are still affected by the original delay for which a was designed to absorb a portion, but such a portion is now absorbed by the successive edges. In case 4), the shifting downward of the slack time (or its removal when there are no successive edges) does not change affection properties. Hence, in all cases the feasibility of the solution is maintained. □

Proof (of Theorem 1). The result immediately follows from the fact that the decisional version of the $\mathcal{RIT}_{\Delta,opt}$ problem, with $\Delta \geq 1$, is NP-complete. Formally let the $\mathcal{RIT}_{\Delta,dec}$ be defined as follows.

$\mathcal{RIT}_{\Delta,dec}$	
GIVEN:	A tree $T = (V, A)$, a function $w : V \to \mathbb{R}_{\geq 0}$, and three numbers $\alpha, \Delta \in \mathbb{N}^+, K' \in \mathbb{R}_{\geq 0}$.
PROBLEM:	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 $s : A \to \mathbb{R}_{\geq 0}$ defined as $s(a = (i, j)) = \Pi(j) - \Pi(i) - 1$, and such that $\sum_{v \in V} \Pi(v)w(v) \leq K'$?

We now show that $\mathcal{RTT}_{\Delta,dec}$ (briefly, $\mathcal{RTT}_{dec}$), is NP-complete by a transformation from *Knapsack* [12].

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-edge 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 we can conclude that $\mathcal{RTT}_{dec}$ is in NP.

First, let us recall the *Knapsack* problem.

Knapsack	
GIVEN:	A finite set U, for each $u \in U$ a size $S(u) \in Z^+$, a value $v(u) \in Z^+$, and positive integers $B, K \in Z^+$.
PROBLEM:	Is there a subset $U' \subseteq U$ such that $\sum_{u \in U'} S(u) \leq B$ and $\sum_{u \in U'} v(u) \geq K$?

Given an instance I of *Knapsack* where $U = \{u_1, u_2, \ldots, u_{|U|}\}$, we define an instance I' of $\mathcal{RTT}_{dec}$. See Figure 4 for a visualization of I'. 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 for each u_i such that $S(u_i) > 1$, 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. The representation provided in Figure 4 is just for better explain the transformation, which indeed can be maintained polynomial.

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

Given a satisfying set $U' \subseteq U$ for I, we define the following solution Π for I':

- $\Pi(r) = 0$, $\Pi(r') = 1$;
- $\Pi(x_i^1) = 2$ 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 $j = 2, 3, \ldots, S(u_i)$.

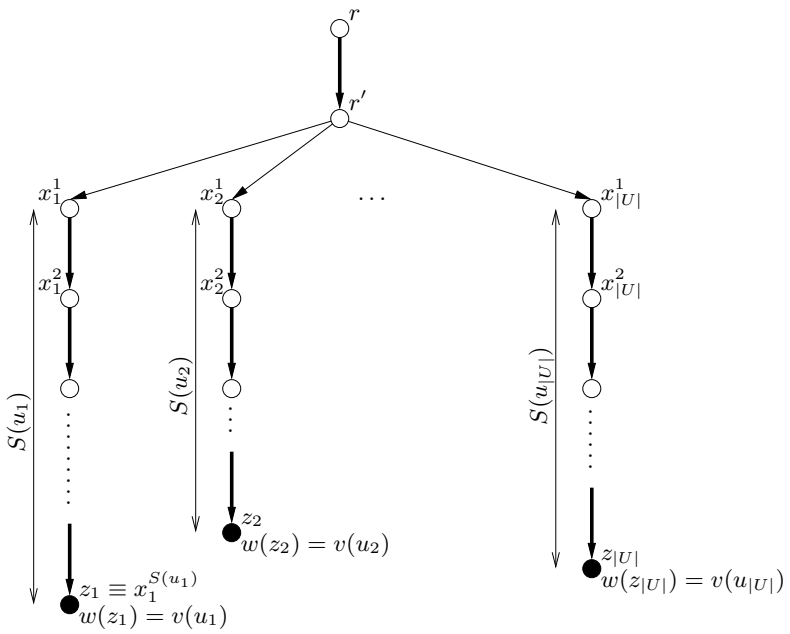

Fig. 4. Instance of $\mathcal{RTT}$ used for the transformation from an instance of knapsack.

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 satisfying solution for $\mathcal{RTT}_{dec}$, then there exists a satisfying solution for *Knapsack*.

Given a satisfying solution Π' for I', we define a satisfying 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 satisfying solution for I' then Π is also a satisfying solution for I'. 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 the solution Π is less than or equal to the number of nodes α-affected by (r, r') in the solution Π'.

We define a solution for I 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 $\sum_{u_i \in U'} v(u_i) \geq K$. □

Proof (Sketch for Theorem 2). The proof is by induction on Δ.
Inductive basis. We prove the statement for $\Delta = 0$. In this case, only Line 5 is executed as $j = 0$, and assigns $G[i, j] = 0$ to each entry. Matrix SOL remains as it was initialized, with all entries equal to 0. This is $\mathcal{RTT}_\Delta$-optimal as there is no possibility to gain anything when $\Delta = 0$.
Inductive step. We prove that if $\texttt{SA-DP}_\Delta$ builds the correct matrices for any $\delta \leq \Delta - 1$, then it does the same for $\delta = \Delta$. When constructing a ball of size $\Delta > 0$ rooted at some node v, first of all we have to consider the gain with respect to $f*$ rising from removing the slack time of α from the incoming edge leading to v, that is $\alpha c(v)$ (Line 7). Then we have to consider all the possibilities for constructing the ball. These are determined by the combination of the number of children of v that the ball takes into account and for each considered child, the size of the ball rooted at such child (size 0 means that the child is not part of the ball). All such combinations are optimally evaluated in G_v and SOL_v at Lines 10-11 as each entry is determined by means of balls of size $\delta < \Delta$ which by inductive hypothesis are correctly found. Hence, by choosing the maximum obtainable value among all such combinations gives us an $\mathcal{RTT}_\Delta$-optimal solution. □

Proof (of Theorem 3). Unless $\Delta = 0$, by construction, $\texttt{SA-DP}_\Delta$ 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 $v \in V$, $\Pi(v) \leq d(r, v) + \alpha \lfloor \frac{d(r,v)}{2} \rfloor \leq d(r, v) \left(1 + \frac{\alpha}{2}\right)$. For any optimal solution Π' of TT, $\Pi'(v) \geq d(r, v)$. When $\Delta = 0$, $\Pi'(v) \geq d(r, v)\alpha$ and

this is equal to the solution provided by $\texttt{SA-DP}_\Delta$, as it does not remove any slack time. Therefore, the statement holds. □

Proof (of Theorem 4). It is sufficient to give an instance such that any robust solution Π implies $f(\Pi) \geq (1 + \frac{\alpha}{\Delta+1}) \cdot f(\Pi')$, 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}), $i = 0, 1, \ldots, \Delta$. For each $i = 0, 1, \ldots, \Delta$, $w(x_i) = 0$ and $w(x_{\Delta+1}) > 0$. Each solution Π of $\mathcal{RTT}_\Delta$ is such that $\Pi(x_{\Delta+1}) \geq d(x_0, x_{\Delta+1}) + \alpha = \Delta + 1 + \alpha$. An optimal solution Π' for TT is such that $\Pi'(x_{\Delta+1}) = d(x_0, x_{\Delta+1}) = \Delta+1$. Hence, $f(\Pi) = (\Delta+1+\alpha)w(x_{\Delta+1}) = (1+\frac{\alpha}{\Delta+1}) \cdot (\Delta+1)w(x_{\Delta+1}) = (1+\frac{\alpha}{\Delta+1}) \cdot f(\Pi')$. □

Proof (of Theorem 5).

The time complexity of the $\texttt{SA-DP}_\Delta$ 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 $j = 0$, $O(1)$ if $i = 0$ and $j > 0$, and $O(j)$ if $i, j > 0$ (see the recurrence defined by Lines 3-10), 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 $\sum_{v \in T} O(|N_o(v)|\Delta^2) = n\Delta^2$. 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. □