Evaluation of recoverable-robust timetables on tree networks*

Gianlorenzo D'Angelo[1], Gabriele Di Stefano[1], and Alfredo Navarra[2]

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

Abstract. In the context of scheduling and timetabling, we study a challenging combinatorial problem which is very interesting from both a practical and a theoretical point of view. The motivation behind it is to cope with scheduled activities which might be subject to unavoidable disturbances, 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 to 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 disturbances occur. Such recovery capabilities are given as input along with the possible disturbances 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. The considered problem has been shown to be NP-hard. We propose a pseudo-polynomial time algorithm 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.

1 Introduction

In this work, we investigate an important combinatorial problem in the context of scheduling: the *timetable planning* of public transportation systems. It arises, for instance, in the planning phase of railway systems, requiring to compute a timetable for passenger trains that determines minimal passenger waiting times. However, many disturbing events might occur during the operational phase, that is when the system is running. Such events, whose main effect is the arising of delays, might make unfeasible the scheduled timetables. Hence, it is important to take them into account in advance.

A schedule that lets vehicles sit at stations for some time will not suffer from small delays of arriving vehicles, because delayed passengers can still catch potential connecting vehicles. On the other hand, big delays can cause passengers to lose vehicles and hence imply extra traveling time. The problem of deciding when to guarantee connections from a delayed vehicle to a connecting vehicle is known in the literature as *delay management problem* [3, 7–11, 14, 15]. Although the problem has a natural formalization, it turns out to be very complicated to be optimally solved. In fact, it has been shown to be NP-hard, while it is polynomial in some particular cases (see [3, 10, 11, 15]).

In order to cope with the management of delays we follow the *recoverable robust optimization* approach provided in [2, 13]. The aim is to design timetables in the planning phase in order to be "prepared" to react against possible disturbances. E.g., if a delay occurs, the designed timetable

* 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).

should guarantee the recovery of the scheduled events by means of allowed operations represented by given recovery algorithms. In [6], interesting theoretical results were presented, assuming that only one delay might occur at a generic event of the scheduled event activity network (see [3, 15]) which is a directed graph that represents the sequence and the dependencies of scheduled events. The attention was restricted to event activity networks whose topology is a tree. In this context, it is worth noting that the assumption concerning one single delay does not constitute a restriction to the problem as k delays of size at most α can be modelled as one delay of size at most $k\alpha$. An event either absorbs the entire delay or propagates it to all the subsequent events. In order to absorb the delay, an event is associated with a so called *slack time* of α time. Clearly, by associating a slack time of α to each event, every delay of duration α can be locally absorbed. However, this approach is not practical as the overall duration time of the scheduled events would increase too much. The planned timetables should be instead 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 events in the network. Namely, the propagation might involve at most Δ events. The objective function to minimize is 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 α time. That is, a timetable is robust if it can be recovered by postponing at most a fixed number of events in case of a delay.

In [4], the authors show that the described problem is NP-hard when the event activity network topology is a DAG with weights associated to nodes. In the same paper, it is shown that the problem where the weights are associated to the arcs is a subproblem. Even for this subproblem, in [3], the authors show that the problem remains NP-hard, and they provide polynomial time algorithms which cope with the case of $\Delta = 0$. In [4], the authors provide algorithms for a generic Δ when the event activity network is a linear graph with positive weights on nodes.

In this paper, we keep on investigating the complexity of the problem. In particular, we study event activity networks which have a tree topology and positive weights. In [6], this problem has been shown to be NP-hard even in this restricted scenario. We provide an algorithm that solves the problem in $O(n^{\Delta+1})$ 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 for constant Δ, i.e. when Δ is fixed a priori. The algorithm is not polynomial, since it has been shown in [6] that some instances can be represented in $O(\log n)$ space and the problem restricted to these instances remains NP-hard. Hence, when Δ is fixed, the proposed algorithm requires a time which is polynomial in n, that is pseudo-polynomial in the size of the instances. For practical contexts, $\Delta << n$ is a reasonable restriction, since we may require that in at most Δ steps (independently from the input instance) a possible delay must be absorbed. The algorithm is based on some investigated properties that specify on trees some preferences about which arcs are the best candidates to be associated with slack times. 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 cheaper solutions. Another interesting property for tree topologies is 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 $\Delta = 0$.

We evaluate the proposed algorithm both theoretically and experimentally. The latter evaluation is done by using random networks and real world instances of the problem provided by the Italian railways company, Trenitalia [16]. It turns out that, even if the algorithm is not polynomial, it is

very efficient in practice. Moreover, we show the applicability and the low costs in terms of slack times needed for making robust the considered timetables with respect to different scenarios.

Preliminary results has been presented at the 3rd International Seminar on Railway Operations Modelling and Analysis (RailZurich2009).

2 Recoverable robust timetabling problem

In railway systems, events and dependencies among events are modeled by means of an *event activity network* (see [15]). This is a directed graph where nodes represent arrival or departure events of trains and arcs represent activities occurring between events (i.e., waiting in a train, driving between stations or changing to another train). Note that, event activity networks are a particular class of direct acyclic graphs (DAGs).

Given a DAG $G = (V, A)$, 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. Given a function $L : A \to \mathbb{N}$ that assigns the minimal duration time to each activity, a timetable $\Pi \in \mathbb{R}_{\geq 0}^{|V|}$ for G is an assignment of 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 to the timetabling problem minimizes the total weighted time for all events. Formally, TT is 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 instances for TT is denoted by I. The set of feasible solutions for $i \in I$ 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 solution Π for TT may induce a positive slack time $s(a)$ for each $a \in A$. In particular, since the planned duration of an activity $a = (u, v)$ is given by $\Pi(v) - \Pi(u)$, then $s(a) = \Pi(v) - \Pi(u) - L(a)$.

Problem TT can be solved in linear time by assigning the minimal possible time to each event (e.g. by using the Critical Path Method [3, 12]). However, in practical context, delays on the scheduled activities may occur. In this cases, an optimal solution for TT could result unfeasible and recovery (on-line) strategies become necessary.

Given an instance $i = (G, L, w)$ for TT, and a constant $\alpha \in \mathbb{R}_{\geq 0}$, we consider a single delay of at most α time. This is modeled as an increase on the minimal duration time of the delayed activity. We denote the set of instances of TT that can be obtained by applying all possible modifications to i as a function $M(i)$ which is formally defined as follows:

$$M(i) = \left\{(G, L', w) : \exists\ \bar{a} \in A : L(\bar{a}) \leq L'(\bar{a}) \leq L(\bar{a}) + \alpha,\ L'(a) = L(a)\ \forall a \neq \bar{a}\right\}.$$

Recovery capabilities against delays are modeled as a class $\mathbb{A}$ of algorithms. The class $\mathbb{A}$ is defined by introducing the concept of *events affected by one delay* as follows.

Definition 1. *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 solution Π for $i = (G, L, w)$, the slack time function induced by Π is used as the function s in the previous definition. It is assumed that the recovery capabilities allow 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 must guarantee that, if a delay of at most α time occurs, then it affects at most Δ events.

We define the recoverable robust timetabling problem $\mathcal{RTT}$ as the problem of finding a timetable that can be recovered by changing the time of at most Δ events when a delay of at most α time occurs. According to the recoverable robustness model in [2, 13], such a problem is defined as $\mathcal{RTT} = (TT, M, \mathbb{A})$.

In other words, a solution Π for an instance i is feasible for $\mathcal{RTT}$ if it can be *recovered* by applying an algorithm in $\mathbb{A}$ which changes the time of at most Δ events for each possible disturbance $j \in M(i)$. The solution Π is called a *robust solution* for i with respect to the original problem TT.

A *robust algorithm* for TT is any algorithm A_{rob} such that, for each $i \in I$, $A_{rob}(i)$ is a robust solution for i with respect to TT. The quality of a robust solution is measured by the *price of robustness*. The price of robustness of A_{rob} is the worst case ratio between the robust solution for an instance of $\mathcal{RTT}$ and the corresponding optimal solution for the underlying TT problem. Formally,

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

The *price of robustness* $P_{rob}(\mathcal{RTT})$ of problem $\mathcal{RTT}$ is defined as the minimum price of robustness among all the robust algorithms. An algorithm A_{rob} is $\mathcal{RTT}$*-optimal* if $P_{rob}(\mathcal{RTT}, A_{rob}) = P_{rob}(\mathcal{RTT})$. A robust solution Π for an instance i of $\mathcal{RTT}$ is $\mathcal{RTT}$-optimal if: $f(\Pi) = \min\{f(\Pi') : \Pi' \text{ is a feasible solution for } \mathcal{RTT}\}$.

3 Pseudo-Polynomial Algorithms for fixed Δ

In this section, we give some theoretical results, whose omitted proofs can be found in Appendix. We concentrate our attention to instances of $\mathcal{RTT}$ where the DAG is a tree. Hence, in the reminder of the paper, we will refer as $\mathcal{RTT}$ to the problem restricted to trees. We denote as $T = (V, A)$ a tree rooted in r. If $v \in V$, $deg(v)$ denotes the degree of v, T_v denotes the subtree of T rooted in $v \in V$. Given a subtree T_v, $N_o(T_v)$ denotes the set of nodes y such that $(x, y) \in A$, $x \in T_v$ and $y \notin T_v$. We denote by $w(T_v)$ the value $\sum_{x \in T_v} w(x)$ and by $|T_v|$ the number of nodes contained in T_v. Note that, in order to compute all the values $w(T_v)$ for each $v \in V$, one visit of T_r is sufficient.

We look for solutions which assign only slack times of size α, as in the next lemma we prove that for any instance, there exists a $\mathcal{RTT}$-optimal solution which fulfills this condition.

Lemma 1. *Given an instance i of $\mathcal{RTT}$, 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 $\Pi'(y) = \Pi'(x) + L(a)$ or $\Pi'(y) = \Pi'(x) + L(a) + \alpha$.*

Denoted by $\mathcal{RTT}_\Delta$ the problem $\mathcal{RTT}$ when the maximal number Δ of affected nodes allowed is fixed a priori, algorithm $\texttt{SA}_\Delta$, provided in Figure 1, is $\mathcal{RTT}_\Delta$-optimal for any fixed $\Delta \geq 1$. The computational complexity of $\texttt{SA}_\Delta$ is $O(n^{\Delta+1})$.

In order to characterize a solution Π, we need the following definition and lemma.

Definition 2. *Given a solution Π of $\mathcal{RTT}_\Delta$ and a node $v \in V$, a* ball *is the maximal subtree $B_\Pi(v)$ rooted in v such that for each arc $a = (x, y)$ in $B_\Pi(v)$, $s(a) = 0$.*

Lemma 2. *For each instance of $\mathcal{RTT}_\Delta$, there exists a $\mathcal{RTT}_\Delta$-optimal solution Π such that for each $v \in V$, $B_\Pi(v)$ cannot be extended by adding any node from $N_o(B_\Pi(v))$ while keeping feasibility and, unless $\Delta = 0$, at most one of two consecutive arcs has a slack time of α.*

Then, for any $\Delta \geq 1$, there exists a $\mathcal{RTT}_\Delta$-optimal solution Π with the following structure. By Lemma 2, for each arc a outgoing from the root r, $s(a) = 0$. Then, for each $v \in N_o(r)$, Π induces a ball $B_\Pi(v)$ such that $|B_\Pi(v)| \leq \Delta$. In particular, $|B_\Pi(v)| < \Delta$ only if $|T_v| < \Delta$. As a consequence, $|B_\Pi(r)| \leq 1 + \Delta \cdot deg(r)$. For each arc $a = (x, y)$ such that $x \in B_\Pi(r)$ and $y \notin B_\Pi(r)$, $s(a) = \alpha$. By Lemma 2, for each arc a outgoing from y, $s(a) = 0$ and the same arguments used for $B_\Pi(r)$ can be used to characterize $B_\Pi(y)$.

A possible approach can be that of enumerating all the solutions with the above structure and choosing the cheapest one. Note that, such an approach has a computational time which is exponential in n. In what follows, we show a recursive approach which avoids to consider a large number of solutions and thus reducing the computational time to a polynomial in n. The algorithm $\texttt{SA}_\Delta$ works as follows. It assigns $\Pi(r) = 0$ and no slack times to arcs outgoing from r. Then, for each $v \in N_o(r)$ it has to decide which subtree of T_v belongs to $B_\Pi(r)$. To do this, it evaluates the cost, in terms of the value of the objective function, of any possible subtree B of T_v rooted at v of size at most Δ and then chooses the subtree which implies the cheapest solution.

For each already defined ball B_Π, this procedure is then repeated for each node $v \in N_o(B_\Pi)$ which does not belong to an already defined ball by using v as the root.

The cost of a subtree B rooted at v is computed as the value of the objective function when B is chosen as a ball rooted in v. That is, for each arc $a \in B$, $s(a) = 0$; for each $a = (x, y) \in A$ such that $x \in B$ and $y \notin B$, $s(a) = \alpha$; and for each node in T_y, an optimal solution is chosen. Computing this cost requires to know the optimal solution of a subtree, this is done by using recursively $\texttt{SA}_\Delta$.

Formally, $\texttt{SA}_\Delta$ is given in Figure 1. It takes a node v as input and it returns a pair $(\Pi, f(\Pi))$ where Π is a $\mathcal{RTT}$-optimal timetable for T_v and $f(\Pi)$ is its value of the objective function. A solution for $\mathcal{RTT}_\Delta$ is computed by calling $\texttt{SA}_\Delta(r)$.

In detail, Lines 1 assigns $\Pi(v)$ and initializes f_Π. For each $v_i \in N_o(v)$, Lines 3–21 compute Π for the subtree T_{v_i}. Lines 3–14 compute a subtree $B_{\min}$ of T_{v_i} rooted at v_i of size at most Δ which implies the cheapest solution of cost $f_{\min}$. To do this, Line 4 enumerates all the possible subtree B rooted at v_i of T_{v_i} of size at most Δ. Then, Lines 5–8 compute the cost f_B of nodes in B and Lines 9–11 compute the cost of nodes not in B by summing for each $(u, z) \in A$, such that $u \in B$ and $z \notin B$ the cost f_{T_z} of a solution of T_z computed by recursively calling $\texttt{SA}_\Delta(z)$. If f_{T_z} is the cost of a solution of a subtree T_z, then the contribution to f_B of all nodes in T_z is $(\Pi_B(u) + L((u, z)) + \alpha) \cdot w(T_z)$ where $\Pi_B(u)$ is the time assigned to u if B is chosen as a ball rooted in v_i. In fact, each time assigned to such nodes have been shifted of $\Pi_B(u) + L((u, z)) + \alpha$. Finally, Lines 15–21 assign Π for the subtree T_{v_i} by choosing $B_{\min}$ as a ball rooted in v_i.

The following theorems give the theoretical results concerning the performances of $\texttt{SA}_\Delta$.

Theorem 1. *$\texttt{SA}_\Delta$ is $\mathcal{RTT}_\Delta$-optimal.*

Theorem 2. *The price of robustness of $\texttt{SA}_\Delta$ is bounded by $P_{rob}(\mathcal{RTT}_\Delta, \texttt{SA}_\Delta) \leq 1 + \frac{\alpha}{2}$.*

Theorem 3. *The price of robustness of $\mathcal{RTT}_\Delta$ is bounded by $P_{rob}(\mathcal{RTT}_\Delta) \geq 1 + \frac{\alpha}{\Delta+1}$.*

Theorem 4. *$\texttt{SA}_\Delta$ requires $O(n^{\Delta+1})$ time and $O(n^2)$ space, where n is the number of nodes.*

By Theorem 4, $\texttt{SA}_\Delta$ requires pseudo-polynomial time for fixed Δ. However, it is worth investigating the performances of $\texttt{SA}_\Delta$ in practical scenarios.

Algorithm $\texttt{SA}_\Delta$

Input: $v \in V$
Output: $(\Pi, f(\Pi))$, times assigned at nodes in T_v with $\Pi(v) = 0$ and $f(\Pi)$

1. $\Pi(v) = 0$, $f_\Pi = 0$
2. **for each** $v_i \in N_o(v)$
3. $\quad f_{\min} = +\infty$
4. $\quad$ **for each** maximal subtree B rooted at v_i of T_{v_i}, such that $|B| \leq \Delta$
5. $\quad\quad \Pi_B(v_i) = L((v, v_i))$, $f_B = L((v, v_i)) \cdot w(v_i)$
6. $\quad\quad$ **for each** $(x, y) \in B$
7. $\quad\quad\quad \Pi_B(y) = \Pi_B(x) + L((x, y))$
8. $\quad\quad\quad f_B = f_B + \Pi_B(y) \cdot w(y)$
9. $\quad\quad$ **for each** $(u, z) \in A$, such that $u \in B$ and $z \notin B$
10. $\quad\quad\quad (\Pi_{T_z}, f_{T_z}) = \texttt{SA}_\Delta(z)$
11. $\quad\quad\quad f_B = f_B + f_{T_z} + (\Pi_B(u) + L((u, z)) + \alpha) \cdot w(T_z)$
12. $\quad\quad$ **if** $f_B < f_{\min}$ **then**
13. $\quad\quad\quad f_{\min} = f_B$
14. $\quad\quad\quad B_{\min} = B$
15. $\quad \Pi(v_i) = L((v, v_i))$
16. $\quad$ **for each** $(x, y) \in B_{\min}$
17. $\quad\quad \Pi(y) = \Pi(x) + L((x, y))$
18. $\quad$ **for each** $(u, z) \in A$, such that $u \in B_{\min}$ and $z \notin B_{\min}$
19. $\quad\quad$ **for each** $x \in T_z$
20. $\quad\quad\quad \Pi(x) = \Pi_{T_z}(x) + \Pi(u) + L((u, z)) + \alpha$
21. $\quad f_\Pi = f_\Pi + f_{\min}$
22. **return** (Π, f_Π)

Fig. 1. Recursive algorithm to compute a robust timetable on a tree.

4 Experimental Study

We present the experimental results first on the real world data provided by Trenitalia [16] and, then, on the randomly generated data.

4.1 Real world data

We consider real case scenarios of *Single-Line Corridors*. A corridor is a sequence of stations represented by a line. The stations are linked by multiple tracks and 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 2. 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 [5].

Let us consider the real world example provided in Figure 2 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

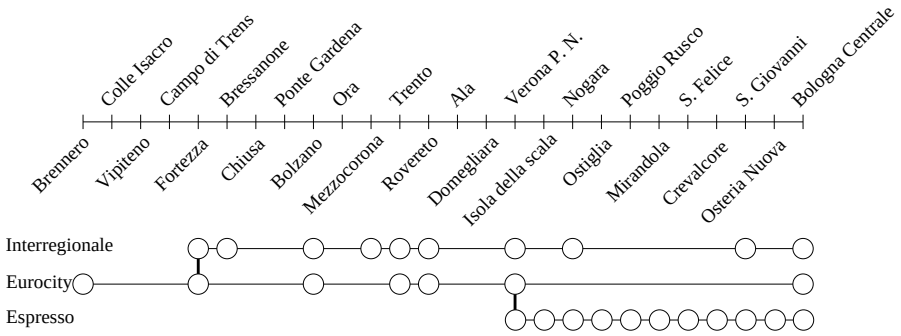

Fig. 2. Example of three trains serving a same line. For the sake of clarity, for each station and for each train, we represented only one circle which indeed corresponds to an arrival and a departure event.

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 2. 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 (for details, see [5]). 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.

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

Corridor	**Line**	**N. of Stations**	**N. of 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.

activity networks having tree topologies whose sizes are reported in Table 2. We then apply the $\texttt{SA}_{\Delta}$ algorithm on different scenarios, comparing the obtained robust timetables with the optimal non-robust ones.

Corridor	**N. of Nodes**	**Maximum Traveling Time**	**Average Activity Time**	**Maximum Number 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.

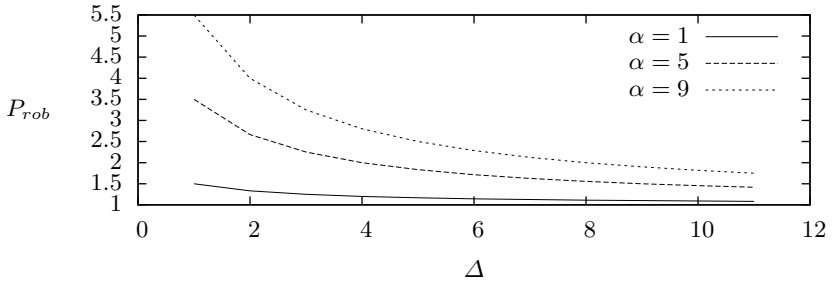

Fig. 3. Theoretical lower bounds to P_{rob}.

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 $\Delta \in \{1, 2, \ldots 11\}$ 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 $\texttt{SA}_\Delta$, and the computational time needed by $\texttt{SA}_\Delta$ 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 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 3, we provide Figure 3 which shows the values of function $1 + \frac{\alpha}{\Delta+1}$ for $\alpha \in \{1, 5, 9\}$ and $\Delta \in \{1, 2, \ldots, 11\}$. 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. However, the exponential growth of the curves as Δ increases is already evident. Surprisingly, for practical purposes, our experiments show that algorithm $\texttt{SA}_\Delta$ can be safely applied without requiring ages of computation.

Corridor `BrBo` (see Figure 4, left). 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. Then, 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, as shown in Figure 4.

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 between 1.00754 and 1.06785, when $\Delta = 11$. This implies that adding robustness reflects an increasing in the costs of just $0.7 - 6.7\%$.

Regarding the computational time, we can see that it increases with Δ but it is less than 30 milliseconds in the worst case. In detail, in the worst case, for $\Delta = 11$ we need about 28 milliseconds to achieve a price of robustness of at most 1.06785.

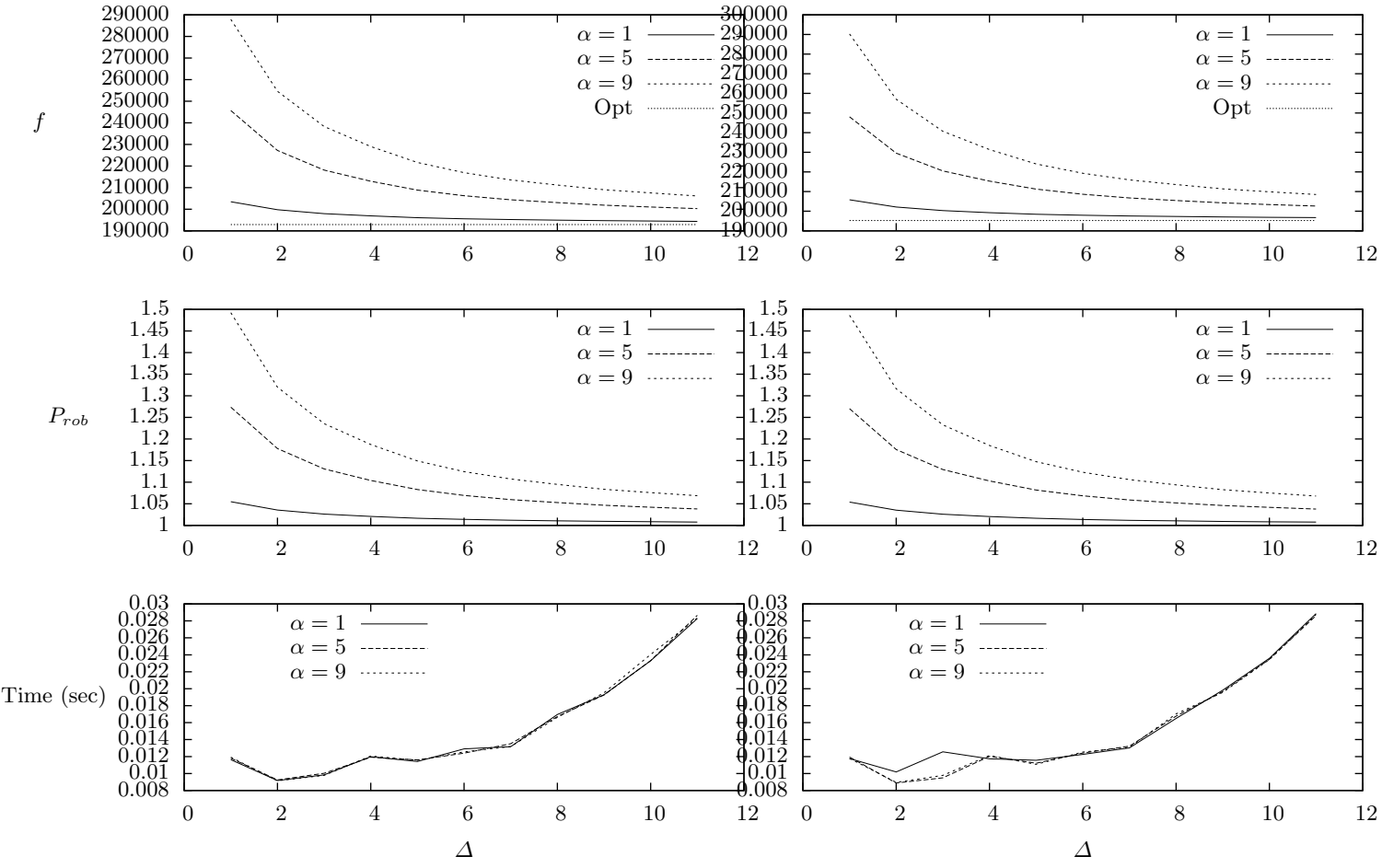

Fig. 4. Corridor `BrBo` with real values of minimum activity times (left) and average activity times (right)

Corridor `BrBo` (average activity times) (see Figure 4, right). 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 4. The same happens for $\alpha = 5$, where the expected value is about 1.27.

Corridor `MdMi` (see Figure 5). 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 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 96 seconds in the worst case. Results regarding corridors `BzVr` and `PzBo` are reported in Appendix.

4.2 Randomly generated data

By analyzing the results in the previous section, it is worth noting that time required by the algorithm on the real world instances is negligible with respect to the theoretical bound. This suggests that those instances have some hidden properties. One cause might 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 a set of five randomly generated trees. Each tree contains 1000 nodes and 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, $\Delta \in \{1, 2, \dots 10\}$ and $\alpha \in \{1, 5, 9\}$. For each pair (Δ, α), we performed one test for each randomly generated tree.

In Figure 6, we summarize the obtained results. In particular, we show the average values of price of robustness and 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 computational times. In fact, in this case the time elapsed is about 10000 times worse than that of the corridor `BrBo` which have comparable size, but different structure. However, the price of robustness is kept low as in the previous instances.

5 Conclusion

We have presented algorithm $\texttt{SA}_\Delta$ for solving the problem of planning robust timetables when the input event activity network topology is a tree. The delivered timetables can cope with delays that might occur at runtime among the scheduled activities. In particular, the algorithm ensures that, if a delay occurs, no more than Δ activities are influenced by the propagation of such a delay. We have shown the performances of $\texttt{SA}_\Delta$ both theoretically and experimentally. Despite the problem is

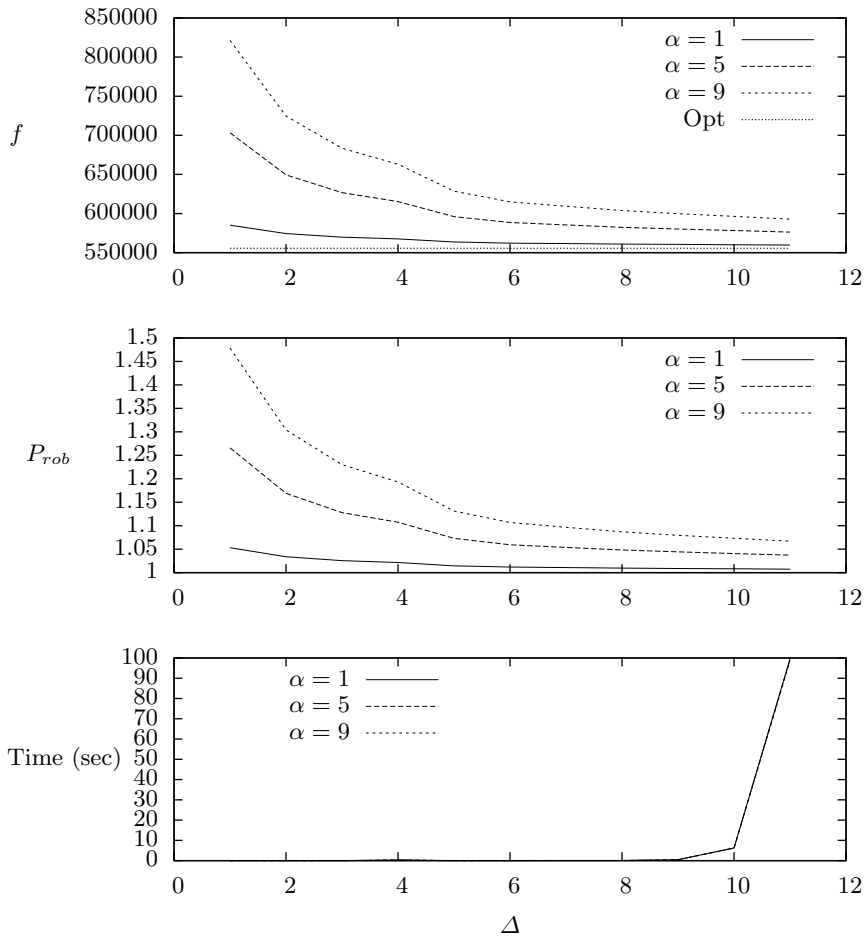

Fig. 5. Corridor `MdMi`

proved to be NP-hard, the obtained results show the applicability of the algorithm to ensure robust timetables with respect to bounded delays. This suggests the practical applicability of $\texttt{SA}_\Delta$ to the planning phase of timetables in order to prevent overlong passengers traveling times. Moreover, the experimental results suggest further investigation about the structure of the input instances. In particular, it has been highlighted that the algorithm behaves well on instances which have an almost linear structure.

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 *Proc. of the 7th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS)*, 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 *Proc. of 2nd Annual International Conference on Combinatorial Optimization and Applications (COCOA)*, volume 5165 of *LNCS*, pages 458–468. Springer, 2008.

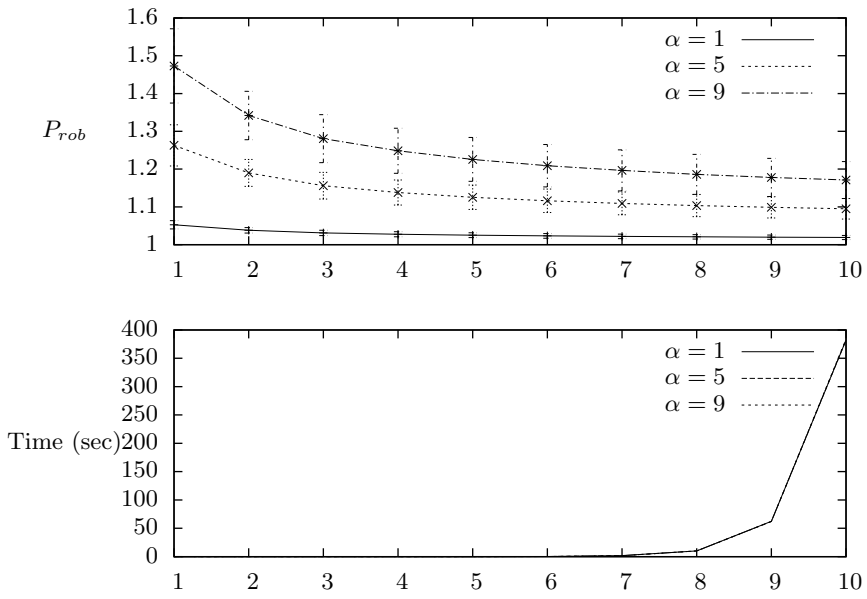

Fig. 6. Randomly generated trees

4. S. Cicerone, G. Di Stefano, M. Schachtebeck, and A. Schöbel. Dynamic Algorithms for Recoverable Robustness Problems. In *Proc. of the 8th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS)*, 2008.
5. G. D'Angelo, G. Di Stefano, and A. Navarra. Recoverable-robust timetables for trains on single-line corridors. Technical Report ARRIVAL-TR-0180, ARRIVAL project, 2008. Presented at 3rd International Seminar on Railway Operations Modelling and Analysis (RailZurich2009).
6. G. D'Angelo, G. Di Stefano, and A. Navarra. Recoverable robust timetables on trees. Technical Report ARRIVAL-TR-0163, ARRIVAL project, 2008.
7. 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.
8. M. Gatto, B. Glaus, R. Jacob, L. Peeters, and P. Widmayer. Railway delay management: Exploring its algorithmic complexity. In *Proc. of the 9th Scandinavian Workshop on Algorithm Theory (SWAT)*, volume 3111 of *LNCS*, pages 199–211, 2004.
9. M. Gatto, R. Jacob, L. Peeters, and A. Schöbel. The Computational Complexity of Delay Management. In *Proc. of the 31st International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, volume 3787 of *LNCS*, pages 227–238, 2005.
10. M. Gatto, R. Jacob, L. Peeters, and P. Widmayer. Online Delay Management on a Single Train Line. In *Proc. of the Algorithmic Methods for Railway Optimization (ATMOS04)*, volume 4359 of *LNCS*, pages 306–320, 2007.
11. A. Ginkel and A. Schöbel. The bicriteria delay management problem. *Transportation Science*, 41(4):527–538, 2007.
12. 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.
13. C. Liebchen, M. Lübbecke, R. H. Möhring, and S. Stiller. Recoverable robustness. Technical Report ARRIVAL-TR-0066, ARRIVAL Project, 2007.
14. A. Schöbel. A model for the delay management problem based on mixed integer programming. *Electronic Notes in Theoretical Computer Science*, 50(1):1–10, 2004.
15. A. Schöbel. Integer programming approaches for solving the delay management problem. In *Proc. of the Algorithmic Methods for Railway Optimization (ATMOS04)*, volume 4359 of *LNCS*, pages 145–170, 2007.
16. Trenitalia. http://www.trenitalia.com/.

A Omitted proofs

Proof (of Lemma 1). Given Π, we define $\Pi' = \Pi$ and then we perform iteratively the next two operations until none of them can be applied.

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

By construction, at the end of the procedure, Π' is feasible.

Now we show that, at the end of the procedure, for each arc $a = (x, y)$, either $\Pi'(y) = \Pi'(x) + L(x, y)$ or $\Pi'(y) = \Pi'(x) + L(x, y) + \alpha$. There cannot exist an arc $a = (x, y)$ such that $\Pi'(y) > \Pi'(x) + L(x, y) + \alpha$, as otherwise Rule 1 can still be applied.

By contradiction, we assume that there exists a non empty set of arcs $\mathcal{A} = \{a = (x, y) : \Pi'(x) + L(x, y) < \Pi'(y) < \Pi'(x) + L(x, y) + \alpha\}$. Let $a = (x, y) \in \mathcal{A}$ be an arc for which $d(r, y)$ is minimal. Then, for each arc b in $P(r, x)$, if $x \in \mathrm{Aff}(b)$, then $y \in \mathrm{Aff}(b)$. It follows that, if we assign $\Pi'(y) = \Pi'(x) + L(x, y)$, Π' remains feasible, a contradiction with respect to Rule 2. □

Proof (of Lemma 2). 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 $\Pi'(y) = \Pi'(x) + L(x, y) + \alpha$. Then, the solution Π'' that assigns $\Pi''(y) = \Pi'(x) + L(x, y)$ while keeping the rest of the original solution is robust and $f(\Pi'') \leq f(\Pi')$. This procedure is repeated until $B_\Pi(v)$ becomes maximal. The obtained solution is optimal as $f(\Pi'') \leq f(\Pi')$ and cannot be extended by adding a node while keeping feasibility, a contradiction.

Suppose that $\Delta > 0$, by contradiction, we assume that each $\mathcal{RIT}_\Delta$-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 $\Pi'(y) = \Pi'(x) + L(x, y) + \alpha$ and $\Pi'(z) = \Pi'(y) + L(y, z) + \alpha$. Then, we can extend the ball rooted in y by defining a solution Π'' such that $\Pi''(z) = \Pi'(y) + L(y, z)$ while keeping the rest of the original solution. Therefore, Π'' is robust and $f(\Pi'') \leq f(\Pi')$. The obtained solution is optimal as $f(\Pi'') \leq f(\Pi')$, a contradiction with respect to the first part of the proof. □

Proof (of Theorem 1). The proof is by induction on the height $h(T)$ of the tree T.
Inductive basis. We prove the statement for trees T such that $h(T) = 0$. In this case, T consists of a single node. Hence, $\mathtt{SA}_\Delta$ performs only Lines 1 and 23 as $N_o(v) = \emptyset$. Then, $\mathtt{SA}_\Delta(r)$ returns $\Pi(r) = 0$, $f(\Pi) = 0$.
Inductive step. We prove that if $\mathtt{SA}_\Delta$ is $\mathcal{RIT}_\Delta$-optimal for any tree T' of height $h(T') \leq h - 1$, then it is $\mathcal{RIT}_\Delta$-optimal for any tree T such that $h(T) = h$.

Let us denote as $Opt_\Delta(x)$ the value of the objective function of an optimal robust solution of the instance given by the subtree T_x rooted at x.

By Lemma 2, an optimal solution Π assigns $\Pi(v_i) = L(v, v_i)$, for each $v_i \in N_o(v)$. Moreover, for each $v_i \in N_o(v)$, Π implies a maximal ball $B_{\min}$ of minimum cost. That is, one has to find $B_{\min}$ among subtrees B_Π of T_{v_i} of size at most Δ such that:

$$B_{\min} = \operatorname*{argmin}_{B_\Pi} \left\{ w(v_i) + \sum_{(x,y)\in B_\Pi} (\Pi(x) + L(x,y)) \cdot w(y) + \right.$$

$$\left. \sum_{(u,z):u\in B_\Pi, z\notin B_\Pi} (Opt_\Delta(z) + (\Pi(u) + L(u,z) + \alpha) w(T_z)) \right\},$$

where for each $(x, y) \in B_\Pi$, $\Pi(y) = \Pi(x) + L(x, y)$. To this aim, $\texttt{SA}_\Delta$ enumerates any possible maximal ball B of size at most Δ (Line 4), then it assigns $\Pi_B(v_i) = L(v, v_i)$ and computes the cost of choosing a ball as follows. For each arc $(x, y) \in B$ it assigns $\Pi_B(y) = \Pi_B(x) + L(x, y)$ (Lines 6–8). For each arc $(u, z) \in A$, such that $u \in B$ and $z \notin B$ it computes (Π_{T_z}, f_{T_z}) by running $\texttt{SA}_\Delta(z)$. The cost of choosing a ball B is then computed at Lines 5, 8 and 11 as:

$$w(v_i) + \sum_{(x,y)\in B} (\Pi_B(x) + L(x,y)) \cdot w(y) +$$

$$\sum_{(u,z):u\in B, z\notin B} (f_{T_z} + (\Pi_B(u) + L(u,z) + \alpha) \cdot w(T_z)) .$$

As $h(T_z) \leq h - 1$, by inductive hypothesis, Π_{T_z} is a solution for T_z which minimizes the price of robustness and $f_{T_z} = Opt_\Delta(z)$. Hence, a ball which gives the minimum cost is chosen (Lines 12–14) as $B_{\min}$.

Finally, $\texttt{SA}_\Delta$ assigns Π and $f(\Pi)$ according to B_{min} and to Π_{T_z}, for each $(u, z) \in A$, such that $u \in B_{\min}$ and $z \notin B_{\min}$ (Lines 15–22). □

Proof (of Theorem 2). At Lines 1 and 15, $\texttt{SA}_\Delta$ does not assign a slack time to arcs (v, v_i). It follows that for any pair of consecutive arcs $a = (x, y), b = (y, z) \in A$ either $\Pi(z) = \Pi(x)+L(x, y)+L(y, z)$ or $\Pi(z) = \Pi(x) + L(x, y) + L(y, z) + \alpha$. 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 an optimal solution Π' of TT, $\Pi'(v) = d(r, v)$. Therefore, the statement holds. □

Proof (of Theorem 3). 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 4). For each $v \in V$, $\texttt{SA}_\Delta(v)$ is performed only once as the result $(\Pi, f(\Pi))$ can be stored.

For each $v \in V$, and for each $v_i \in N_o(v)$, Lines 4–14 of $\texttt{SA}_\Delta$ require the following computation times: Lines 5–8 require $O(1)$ time as the nodes in a ball B are at most Δ; Lines 9–11 require $O(|T_{v_i}|)$ time as there are $O(|T_{v_i}|)$ arcs fulfilling condition at Line 9; and Lines 12–14 require $O(1)$ time. The number of possible maximal balls rooted in v_i is bounded by:

$$\sum_{i=1}^{\Delta-1} \binom{|T_{v_i}|}{\Delta - i} \cdot |T_{v_i}|^{i-1} \leq \sum_{i=1}^{\Delta-1} |T_{v_i}|^{\Delta-i} \cdot |T_{v_i}|^{i-1} = (\Delta - 1)|T_{v_i}|^{\Delta-1}.$$

As there are at most $(\Delta - 1)|T_{v_i}|^{\Delta-1}$ possible maximal balls, Lines 4–14 requires $O(|T_i|^\Delta)$ time. Lines 15–17 require $O(1)$ time and Lines 18–20 requires $O(|T_{v_i}|)$ time.

It follows that, for each $v \in V$, Lines 2–21 require $O(\sum_{v_i \in N_o(v)} |T_i|^\Delta)$ time. Note that,

$$\sum_{v_i \in N_o(v)} |T_i|^\Delta \leq \left(\sum_{v_i \in N_o(v)} |T_i| \right)^\Delta \leq n^\Delta.$$

Hence, the computation time of calling $\texttt{SA}_\Delta(v)$ is $O(n^\Delta)$. Then, the overall computation time is $O(n \cdot n^\Delta) = O(n^{\Delta+1})$ since $\texttt{SA}_\Delta$ is called at most n times.

The overall space occupancy is $O(n^2)$ as, for each $v \in V$, the result $(\Pi, f(\Pi))$ of $\texttt{SA}_\Delta(v)$ has to be stored and Π requires $O(n)$ space. □

B Further experiments

Corridors `BzVr` and `PzBo` (see Figure 7). 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 7.232 milliseconds for `BzVr` and 1.354 milliseconds for `PzBo`).

For the corridor `BzVr`, the price of robustness for small values of Δ is high, whereas, it is small for the 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.

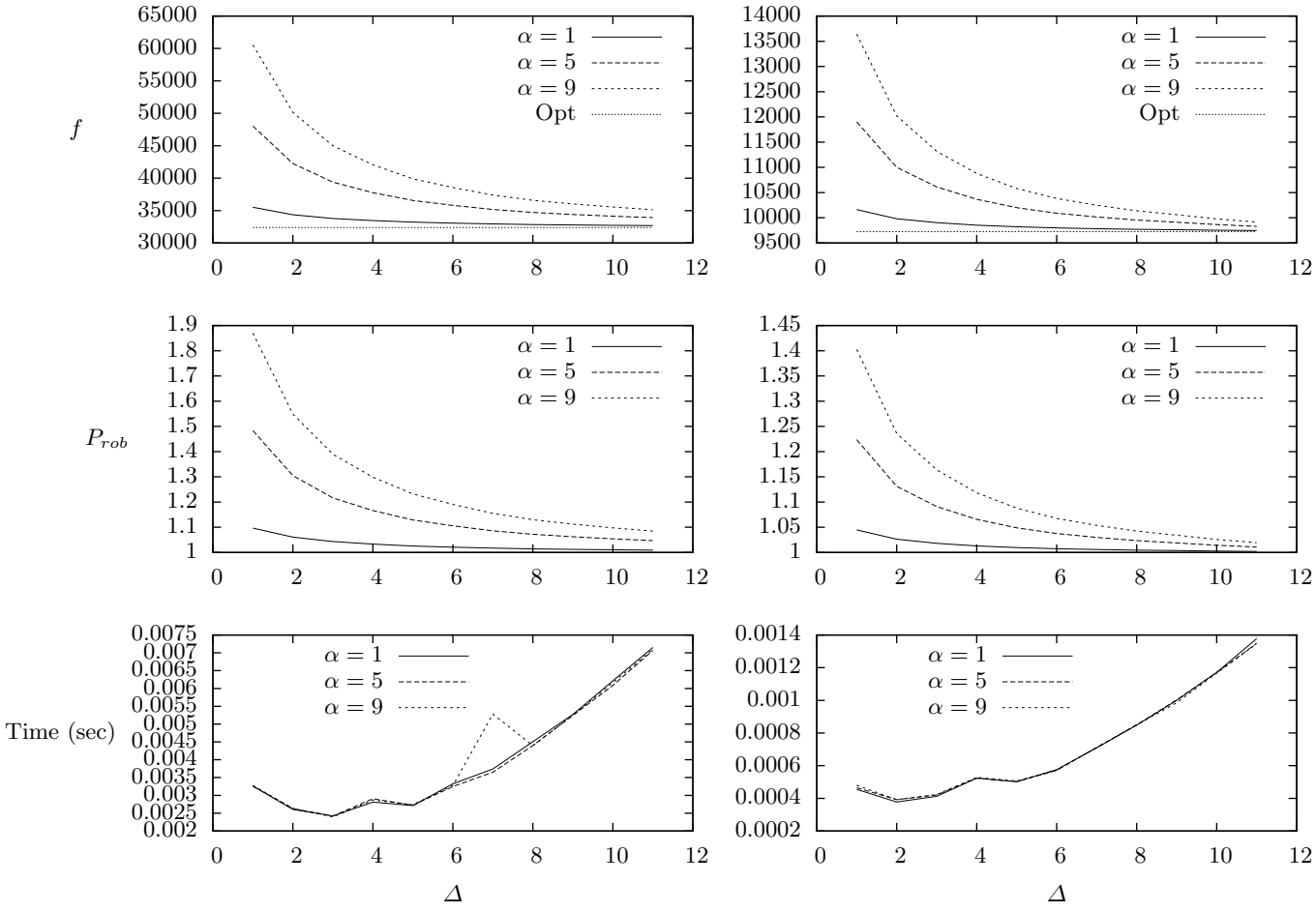

Fig. 7. Corridors `BzVr` and `PzBo`