Available online at www.sciencedirect.com
SCIENCE DIRECT®
Theoretical Computer Science 318 (2004) 297–322
Theoretical Computer Science
www.elsevier.com/locate/tcs

Optimal paths in weighted timed automata☆

Rajeev Alur[a], Salvatore La Torre[b,*], George J. Pappas[c]

[a] *Department of Computer and Information Science, University of Pennsylvania, 200 South 33rd Street, Philadelphia, PA 19104, USA*

[b] *Dipartimento di Informatica ed Applicazioni, Università degli Studi di Salerno, Via Urbano II, Baronissi 84081 (SA), Italy*

[c] *Department of Electrical and System Engineering, University of Pennsylvania, 200 South 33rd Street, Philadelphia, PA 19104, USA*

Received 17 July 2002; received in revised form 10 September 2003; accepted 22 October 2003
Communicated by T.A. Henzinger

Abstract

We consider the *optimal-reachability problem* for a timed automaton with respect to a linear cost function which results in a *weighted timed automaton*. Our solution to this optimization problem consists of reducing it to computing (parametric) shortest paths in a finite weighted directed graph. We call this graph a *parametric sub-region graph*. It refines the region graph, a standard tool for the analysis of timed automata, by adding the information which is relevant to solving the optimal-reachability problem. We present an algorithm to solve the optimal-reachability problem for weighted timed automata that takes time exponential in $O(n(|\delta(A)| + |w_{\max}|))$, where n is the number of clocks, $|\delta(A)|$ is the size of the clock constraints and $|w_{\max}|$ is the size of the largest weight. We show that this algorithm can be improved, if we restrict to weighted timed automata with a single clock. In case we consider a single starting state for the optimal-reachability problem, our approach yields an algorithm that takes exponential time only in the length of clock constraints.

Keywords: Hybrid systems; Model checking; Optimal reachability; Timed automata

☆ A preliminary version of this paper appears in the Proc. of the 4th Internat. Workshop on Hybrid Systems: computation and control, HSCC'01, Rome, Italy, March 28–30, 2001, Lecture Notes in Computer Science, Vol. 2034, pp. 49–62.

* Corresponding author.
E-mail addresses: alur@cis.upenn.edu (R. Alur), slatorre@unisa.it, latorre@dia.unisa.it (S. La Torre), pappasg@ee.upenn.edu (G.J. Pappas).

0304-3975/$ - see front matter © 2003 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2003.10.038

1. Introduction

Reachability is a core decision problem in system verification since it can be used to detect the violation of *safety* properties ("nothing bad will eventually happen"), or for the presence of deadlocks. For some applications, it is natural to assign costs to system transitions, so that it is meaningful to ask questions such as "what is the minimum cost to reach a state of the system?". Optimal reachability is of interest in many practical problems and has been extensively studied for discrete graphs (*shortest path problem*). In this paper, we are concerned with the optimal-reachability problem for timed automata.

Timed automata [2] is a formalism to model the behavior of real-time systems: a discrete transition graph is equipped with a finite set of *clock variables* that are used to express *timing constraints*. The semantics of timed automata is given by an infinite-state transition system where transitions correspond either to a change of location (discrete transition) or to a time consumption (time transition). In order to define the optimal-reachability problem we need to associate a cost with each transition. A natural way of doing this is to assign a weight to each transition of the timed automaton (this will give a cost to each discrete transition) and a real function to each location (this will give a cost to each time transition depending on the elapsed time). Here we restrict to functions of the form $f(t) = a \cdot t$ for a nonnegative constant a.

We define a *weighted timed automaton* as a timed automaton augmented with weights (different costs) on both locations and transitions. The cost of a run is given by the sum of two addend: the sum of the costs of the taken switches and the sum of the costs of the time spent in the visited locations, each such cost given by the product of the cost associated with the location multiplied by the time spent in it. Given a weighted timed automaton A and two zones (sets of states) of A, S and T, the *optimal-reachability problem* is the problem of determining for each state s in S the infimum cost over all the runs of A from s to a state in T. The set S is called the *source zone* and the set T is called the *target zone*. If the source zone contains only one state of the automaton, we refer to this problem as the *single-source* optimal-reachability problem.

Our solution to the optimal-reachability problem consists of two main steps: first we reduce the optimal-reachability problem to a shortest-path problem in directed graphs, then we solve the latter. In the first step, for each clock region of the source zone we construct a finite graph which is a refinement of the region graph [2]. We call such a graph a *parametric sub-region graph*. The parametric sub-region graph of a weighted timed automaton contains vertices corresponding to sub-regions of clock regions that are defined depending on the starting state and the sequences of resets that may occur in "potential" optimal runs. It is parameterized on the nonzero differences of two consecutive fractional parts in the clock valuation of the starting state, and its size is linear in the size of the timed automaton and exponential in the size of clock constraints.

When we consider a general source zone, we leave unspecified the parameters and the above construction reduces the optimal-reachability problem for weighted timed automata to a parametric shortest-path problem on weighted directed graphs. We give a fix-point computation algorithm to solve this problem that takes time linear in the

number of vertices and exponential in $O(k(|a_{\max}| + \log(l_{\max}))$, where k is the number of parameters, $|a_{\max}|$ is the largest coefficient used in the expressions labeling the edges of the graph, and $l_{\max}$ is the length of the longest simple path. We also provide a lower bound by giving an instance of the problem where the size of the solution is exponential in the number of parameters. Combining this solution with the parametric sub-region graph construction, we obtain an algorithm to solving the optimal-reachability problem that takes time exponential in $O(n(|\delta(A)| + |w_{\max}|))$, where n is the number of clocks, $|\delta(A)|$ is the size of the clock constraints and $|w_{\max}|$ is the size of the largest weight.

For the single-source optimal-reachability problem, we substitute the parameters in the parametric sub-region graph with the actual values from the starting state. This reduces our optimization problem to a standard shortest-path problem on finite graphs. Running Dijkstra's algorithm on the obtained graph, we obtain an algorithm that takes time exponential only in the length of clock constraints, thus improving the algorithm for the general case. We also provide a different algorithm for the optimal-reachability problem when the input automaton has only one clock variable. This algorithm improves the algorithm for the general case by a constant factor in the exponent, and is not depending on the size of the weights of the automaton. It runs in time linear in the length of the longest single path, cubic in the size of the automaton and exponential in the size of the largest constant used in the clock constraints. Finally, we recall that if we restrict to weighted timed automata whose control graph is acyclic, the decision version of the optimal-reachability problem becomes NP-complete [11], while for control graphs with arbitrary shapes it is known to be PSPACE-complete [12].

The rest of the paper is organized as follows. We conclude this section with a discussion on the related work. In Section 2, we define the optimal-reachability problem and give some examples. In Section 3, we introduce a graph construction to reduce the optimal-reachability problem to the shortest-path problem in directed graphs. In Section 4, we present our solutions to the single-source optimal-reachability problem. In Section 5, we give a general solution to the shortest-path problem on graphs with parametric costs and then we use it to solve the optimal-reachability problem. We also give the algorithm for the case of weighted timed automata with a single clock. In Section 6, we conclude with few final observations.

1.1. Related work

Time-optimal reachability was first considered in [7], where the problem of computing lower and upper bounds on time delays in timed automata was solved. In [1], a weight w is associated with each location q such that w gives the cost of a unit of time spent in q. Therefore, given a cost interval I and two states s and t, the decision problem "is t reachable from s along a run with a cost $c \in I$?" (*duration-bounded reachability*) is addressed. For timed automata with only closed guards (i.e., guards with atoms of the form $l \leqslant x \leqslant u$), the duration-bounded reachability can be rephrased as the satisfiability of a duration constraint in an integration graph [9]. The solution given in [9] consists of reducing an instance of this satisfiability problem to a generalized integer linear program of exponential size. In [3], the minimum-time reachability and the more general minimum-time control synthesis problems for timed automata are

solved by a backward fix-point algorithm. Minimal-time reachability is also solved by a forward fix-point algorithm in [14].

The results obtained in the above papers can be used to solve special cases of the general optimal-reachability problem we consider in this paper. We also recall that the single-source optimal-reachability problem was independently defined and solved by [5]. There, the authors give a branch-and-bound algorithm to compute for each "corner" of any clock region the minimum cost of a run to the target. Thus, the minimum cost for a state in a region is given by a convex linear combination of the costs associated with the corners of the region. The computational complexity of this algorithm is not analyzed. A zone-based solution related to this approach is presented in [10].

2. Preliminaries

In this section, we give the definitions of the single-source and of the parametric optimal-reachability problems. We first introduce some notation and the definition of timed automaton.

Given a set C of n variables, a k-zone is a subset of $\mathbb{R}^n$ that can be obtained as a boolean combination of inequalities of the form $x \leqslant y + c$, $x < y + c$, $x \leqslant c$, and $x < c$ where $x, y \in C$ and $c \in \{0, 1, \ldots, k\}$. We denote by TRUE the inequality which is true for any value of the variables. We denote by $Z(C)$ the set of all the k-zones, for all $k \in \mathbb{N}$. A function $\lambda : \mathbb{R}^n \to \mathbb{R}^n$ is called a reset function if it is the identity function on some of the coordinates and maps the others to zero. We denote by Λ_n the set of all reset functions over $\mathbb{R}^n$. A *timed automaton* [1] A is a tuple $(Q, C, \Delta, \mathrm{Inv})$ where:

- Q is a finite set of locations;
- C is a set of n clock variables;
- Δ is a finite subset of $Q \times Z(C) \times \Lambda_n \times Q$;
- $\mathrm{Inv} : Q \to Z(C)$ maps each location q to its invariant $\mathrm{Inv}(q)$.

A state is a tuple (q, v) where $q \in Q$ and $v \in \mathbb{R}^n$. We denote by $S = Q \times \mathbb{R}^n$ the set of states for A. For a tuple $v \in \mathbb{R}^n$ and a real number $t \in \mathbb{R}$, we denote by $v + t$ the tuple obtained by adding t to all v components. A *discrete step* is $(q, v) \xrightarrow{e} (q', v')$ where $e = (q, \delta, \lambda, q') \in \Delta$, v satisfies δ, $v' = \lambda(v)$, and v' satisfies $\mathrm{Inv}(q')$. A *time step* is $(q, v) \xrightarrow{t} (q, v')$ where $v' = v + t$, $t \geqslant 0$, and $v + t'$ satisfies $\mathrm{Inv}(q)$ for all t' such that $0 \leqslant t' \leqslant t$. A *step* is $(q, v) \xrightarrow{t} \xrightarrow{e} (q', v')$ where $(q, v) \xrightarrow{t} (q, v'')$ and $(q, v'') \xrightarrow{e} (q', v')$, for some $v'' \in \mathbb{R}^n$, that is a transition e taken after spending some time t in the current location q. A run r of a timed automaton A is a finite sequence $(q_0, v_0) \xrightarrow{t_1} \xrightarrow{e_1} (q_1, v_1) \xrightarrow{t_2} \xrightarrow{e_2} \cdots \xrightarrow{t_{k-1}} \xrightarrow{e_{k-1}} (q_{k-1}, v_{k-1}) \xrightarrow{t_k} (q_k, v_k)$. We say that r starts at (q_0, v_0) and ends at (q_k, v_k). The definition of r allows time to be spent after taking the last transition e_{k-1}.

A *weighted timed automaton* is a timed automaton A with the following cost functions:

[1] The standard definition of timed automata requires also an acceptance condition and a symbol alphabet. Since we are not interested in studying languages accepted by timed automata we omit these features here.

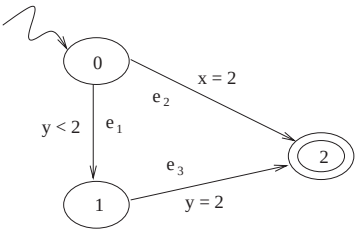

Fig. 1. A timed automaton with more than one optimal run from the same location.

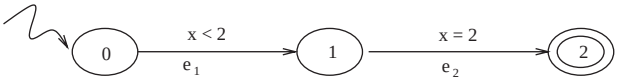

Fig. 2. A timed automaton with no optimal runs from a location.

- $J_s : \Delta \to \mathbb{N}$ (*switch cost*), and
- $J_d : Q \to \mathbb{N}$ (*duration cost*).

Given a run r of A and cost functions J_s, and J_d, we associate costs to r as follows:

- $J_s(r) = \sum_{i=1}^{k-1} J_s(e_i)$, and
- $J_d(r) = \sum_{i=0}^{k-1} t_{i+1} \cdot J_d(q_i)$.

The total cost associated to a run r is $J(r) = J_s(r) + J_d(r)$. We are interested in determining optimal-cost runs for a timed automaton. In the following examples, we informally introduce some notions that we will formalize in the rest of the section.

Example 1. Consider the timed automaton shown in Fig. 1 such that $J_d(0) = 3$, $J_d(1) = 1$, the switch costs are all 1, and the invariants are all TRUE. Suppose that we start from state $s = (0, \langle x, y \rangle)$, where $0 \leqslant x \leqslant 2$ and $0 \leqslant y < 2$, and that we want to reach a state in location 2. Possible minimal-cost runs from s to a state $s' = (2, \langle x', y' \rangle)$ are of two forms: $r_1 = (0, \langle x, y \rangle) \xrightarrow{t_1} (1, \langle x + t_1, y + t_1 \rangle) \xrightarrow{t_2} (2, \langle x + 2 - y, 2 \rangle)$, where $y + t_1 < 2$ and $y + t_1 + t_2 = 2$, and $r_2 = (0, \langle x, y \rangle) \xrightarrow{t_3} (2, \langle 2, y + 2 - x \rangle)$, where $t_3 = (2 - x)$ (obviously, staying at location 2 longer might only increase the overall cost). According to the cost function J, the cost of r_1 is $J_s(r_1) + J_d(r_1) = 2 + 3t_1 + (2 - y - t_1) = 4 - y + 2t_1$ and the cost of r_2 is $J_s(r_2) + J_d(r_2) = 1 + 3(2 - x) = 7 - 3x$. Clearly, $J(r_1)$ is minimized when $t_1 = 0$, that is the transition from 0 to 1 is taken immediately. Moreover, assuming $t_1 = 0$, $J(r_1) \leqslant J(r_2)$ if $y \geqslant 3(x - 1)$, and $J(r_1) > J(r_2)$, otherwise. Thus, a minimal-cost run from s to a state in location 2 depends on the clock valuation of the starting state s.

Example 2. Consider the timed automaton shown in Fig. 2 such that $J_d(0) = 1$, $J_d(1) = 2$, and the switch costs are all 1. Suppose that we start from state $s = (0, \langle x \rangle)$ with $0 \leqslant x < 2$ and we want to reach a state in location 2. We observe that due to the constraint on transition e_2, location 2 is visited for the first time with $x = 2$. Any run reaching $(2, \langle 2 \rangle)$ from s can be parameterized over the time spent in location 0,

that is, if t is the time spent in location 0, the only run from s to $(2,\langle 2\rangle)$ is $r_t = (0,\langle x\rangle) \xrightarrow{t}_{e_1} (1,\langle x+t\rangle) \xrightarrow{t'}_{e_2} (2,\langle 2\rangle)$ with $t' = 2-x-t$. Thus $J(r_t) = J_s(r_t)+J_d(r_t) = 2+t+2(2-t-x) = 6-t-2x$. Hence the cost of r_t is minimized if t is maximized. Since $t<(2-x)$ must hold, the optimal cost for a run starting at s is $(4-x)$, but none of the runs starting at s has such a cost. In fact, for any actual run r_t there exists a $\xi>0$ such that $t=(2-x-\xi)$, and $J(r_t)=(4-x+\xi)$. Vice versa, for any $\xi>0$ there exists a run r such that $J(r)=(4-x+\xi)$. Clearly, there is no minimal-cost run but we can determine a run whose cost is arbitrarily close to the optimal one.

Now, we formalize the notion of optimal cost, optimal run, and approximation of an optimal run. Given a timed automaton A, a state s, and a target zone T, the *optimal cost* for a run from s to $t\in T$ is defined as $J^* = \inf\{J(r) \mid r \text{ is a run from } s \text{ to } T\}$. If there exists a run r^* such that $J(r^*)=J^*$, then r^* is said to be an *optimal* run. As shown in Example 2, sometimes an optimal run from a state s to a target zone T does not exist. When this is the case, we are interested in determining a set R of runs such that all the runs coincide on the sequence of switches and for any $\xi>0$ there exists a run $r\in R$ such that $J(r) < J^*+\xi$, where J^* is the optimal cost over all runs from s to T. That is, we can determine a sequence of runs in R whose costs are arbitrarily close to J^*. We call such a set of runs R an *approximation* of an optimal run. Given a timed automaton A, a source state s, and a target zone T, we consider the problem of determining an optimal run from s to a state in T, if one exists, or an approximation of an optimal run, otherwise. We call this problem the *single-source optimal-reachability problem*.

We are also interested in solving the above problem starting from any state of a source zone S. We call this problem the *optimal-reachability problem*. A solution to the optimal-reachability problem is a symbolic representation of the solutions to the single-source optimal-reachability problem for all states in S. In Example 1, if we consider as target region all the states in location 2 and as source state only $(0,\langle 0,0\rangle)$, then a solution to the corresponding instance of the single-source optimal-reachability problem is r_1 with $t_1=0$. As observed in Example 1, if we consider as a source zone the set of states $(0,\langle x,y\rangle)$ such that $0\leqslant x\leqslant 2$ and $0\leqslant y<2$, then the solution of the corresponding instance of the optimal-reachability problem is r_1 with $t_1=0$ if $y\geqslant 3(x-1)$, and r_2, otherwise.

Optimal-reachability in weighted timed automata is suitable for modeling several optimization problems, such as scheduling problems and air-traffic control problems. We end this section by modeling in our framework an air-traffic control problem that we also use in the next section to illustrate our graph construction.

In the current practice, when an aircraft enters a sky region around an airport, it follows a trajectory chosen from a predetermined set of trajectories, to approach a runway (see Fig. 3). Air-traffic control is responsible for the coordination of approaching aircrafts in order to guarantee safety and high performance of the global system (i.e., the system composed of the aircrafts and the airport). It is not rare, that two aircrafts, that are approaching the same runway, reach the joining point of their trajectories almost at the same time, or that they share the same trajectory and the aircraft that is following is faster. Both these cases can lead the two aircrafts to an unsafe distance from each

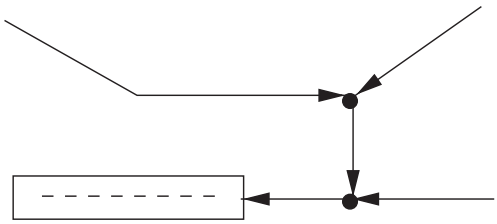

Fig. 3. Possible trajectories for aircrafts approaching a runway.

other, that is, to a collision danger. As soon as such a dangerous situation is detected, it is important to coordinate the aircrafts to avoid their collision. Possible decisions are to slow down an aircraft, or to force one of them to change its current trajectory. Each of the possible actions usually adds to the cost. Examples of such costs are fuel consumption, customer comfort, travel times. As shown in the following example, we can model such scenarios by weighted timed automata and the related control problem as the optimal-reachability problem on them.

Example 3 (Air-traffic control problem). Consider the scenario in which two aircrafts are landing at an airport and a collision danger has been detected. Our goal is to allow both the aircrafts to land safely and at the minimum cost. The safety requirement forces that only one aircraft at a time must be acknowledged for landing on the runway. Thus there are two possible choices: aircraft 1 has the priority over aircraft 2, or vice versa. We model the related control problem as the optimal-reachability problem from the source location to the target location on the weighted timed automaton in Fig. 4. We penalize with discrete costs c_1 and c_2 the choice of forcing, respectively, aircraft 1 and aircraft 2 to wait. Moreover, we also consider a cost, given by w_i, which penalizes the time spent on waiting for each aircraft. For the aircraft that has to wait for clearance, we model two possible maneuvers. A first one is to reduce the speed and in this case the aircraft stays in location W_i. The other possibility is to take a detour from the original trajectory, which is modeled by the loop through location W_i'. Taking this detour requires a fixed cost c_i', and the cost for the time spent during this maneuver is w_i' instead of w_i per each time unit. To make our example more realistic, we also assume that a detour takes at least time 1 and we penalize an unused runway by a cost c_0 per time unit. Finally, we assume that the landing of each aircraft takes at least time 1 since the related acknowledgment was issued by the control tower. Clearly, in the modeled collision scenario, the least-expensive strategy corresponds to an optimal-cost run in the weighted timed automaton from Fig. 4.

3. The graph construction

In this section, we give the graph construction that underlies the reduction of the single-source optimal-reachability problem to the shortest-path problem and the reduction of the optimal-reachability problem to a parametric shortest-path problem. The

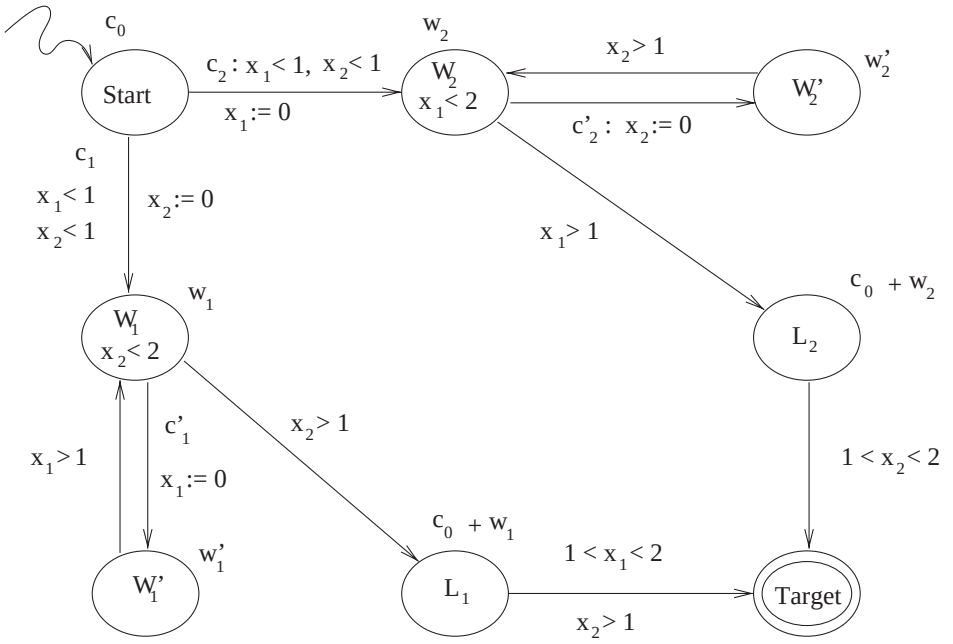

Fig. 4. An air-traffic control problem.

resulting graph is a refinement of the region graph of a timed automaton [2], in the sense that for each region we consider some sub-regions and the graph preserves the transitions of the region graph. For each of these sub-regions, only the states that might be visited in some of the runs of the timed automaton are symbolically represented by the vertices of the graph. The set of states corresponding to a vertex are parameterized over the starting (source) states. We start by recalling the concepts of labeled directed graph and region graph, then we describe our graph construction.

3.1. Labeled directed graphs

Let $\Theta = \{\vartheta_1, \ldots, \vartheta_k\}$ be a set of nonnegative real-valued parameters, we denote by D the set of linear expressions over Θ, that is, the set of first-degree polynomials over $\vartheta_1, \ldots, \vartheta_k$ with nonnegative integer coefficients. A *D-labeled directed graph* G is a pair (V,E), where V is a set of vertices, and $E \subseteq V \times D \times V$ is a set of D-labeled edges. A path π from v_0 to v_n in G is a sequence $v_0 \xrightarrow{f_1} v_1 \xrightarrow{f_2} \cdots \xrightarrow{f_{n-1}} v_{n-1} \xrightarrow{f_n} v_n$ such that $v_i \in V$ for $i = 0, \ldots, n$, and $f_i \in D$ and $v_{i-1} \xrightarrow{f_i} v_i \in E$ for $i = 1, \ldots, n$. For a path π, the cost of π is given by the expression $\sum_{i=1}^{n} f_i$. Given two expressions $f_1 = a_0 + a_1\vartheta_1 + \cdots + a_k\vartheta_k$ and $f_2 = b_0 + b_1\vartheta_1 + \ldots + b_k\vartheta_k$, we say that f_1 is less than f_2, denoted $f_1 \prec f_2$, if and only if $a_i < b_i$ for all $i = 0, \ldots, k$. Clearly, if $f_1 \prec f_2$, for all values of the parameters $\vartheta_1, \ldots, \vartheta_k$ we have that the value of f_1 is smaller than the

value of f_2. According to the relation $\prec$, we define a shortest path for a D-labeled graph as follows. Given a path π from v to v' of cost f, π is a *shortest path* if, for any path π' from v to v' of cost f', $f' \prec f$ does not hold (i.e., the cost associated with π' is not smaller than the cost associated with π). Notice that, by this definition, for any pair of vertices v and v', we may have many shortest paths connecting them. An important fact is that, when the parameters are assigned, the shortest path from v to v' can be found among such paths. Clearly, varying the values of parameters, the shortest paths of a graph may change, that is, to different valuations of parameters may correspond different sets of shortest paths in the graph.

3.2. Region graph

Let A be a timed automaton. By definition, its set of states is infinite. However, the set can be partitioned into a finite number of equivalence classes, called *regions*, which are defined by a location and a *clock region*. Let c_x be the largest constant in the guards and the invariants involving the clock variable x. Then, a clock region is described by

- a constraint of the form $c-1<x<c$, $x>c_x$, or $x=c$ for each clock variable x and natural number $c \leqslant c_x$;
- the ordering of the fractional parts of the clock variables x such that $x<c_x$.

Thus a clock region denotes a set of clock valuations. Given a clock valuation v, $[v]$ denotes the clock region containing v. A state (q, v) belongs to a region $\langle q', \alpha\rangle$ if $q=q'$ and $v \in \alpha$. A clock region α is said to be *open* if for any clock variable x and $c \leqslant c_x$, $x=c$ does not hold in α. Otherwise α is said to be a *boundary* clock region. These definitions apply to regions in an obvious way. Directly from the definition, we have that all the valuations belonging to a region satisfy the same set of clock constraints from a given timed automaton. Consequently, we say that a clock region α satisfies a constraint δ if v satisfies δ for any $v \in \alpha$. A clock region α' is a *time-successor* of a clock region α, denoted $\alpha \sqsubset \alpha'$, if and only if for any $v \in \alpha$ there exists a $d>0$ such that $v+d \in \alpha'$. Consistently with the notation used for clock valuations, given a reset function λ and a clock region α, with $\lambda(\alpha)$ we denote the clock region of the clock valuations $\lambda(v)$ for $v \in \alpha$.

The *region graph* of A is a transition system defined by

- a set of vertices $R(S)=\{\langle q, \alpha\rangle \mid q \in Q$ and α is a clock region for $A\}$;
- a set of edges $R(\Delta)$ such that: $(\langle q, \alpha\rangle, \langle q', \alpha'\rangle) \in R(\Delta)$ if and only if there exist a transition (q, δ, λ, q') of A and a time-successor α'' of α such that α satisfies $\mathrm{Inv}(q)$, α'' satisfies δ and $\mathrm{Inv}(q)$, $\alpha'=\lambda(\alpha'')$ satisfies $\mathrm{Inv}(q')$, and β satisfies $\mathrm{Inv}(q)$ for each β such that $\alpha \sqsubset \beta \sqsubset \alpha''$. In the following, we refer to (q, δ, λ, q') as the A transition corresponding to $(\langle q, \alpha\rangle, \langle q', \alpha'\rangle)$.

We denote by $R(A)$ the region graph corresponding to A. For the sake of simplicity, in the following when no confusion can arise we refer to the value of a clock variable x by x itself and with $\bar{x}$ we denote the fractional part of a clock value x. We recall that the key property of the regions is that they define a bisimulation over the states of A and thus reachability in A can be reduced to reachability in $R(A)$.

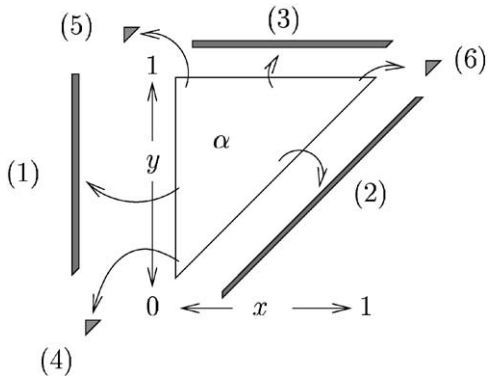

Fig. 5. A graphical representation of the limit regions contained in α.

3.3. Parametric sub-region graph

In this paper, we consider cost functions that are linear in the time spent in each location, thus they reach their infimum on the boundaries of regions. As a consequence, in an optimal run the transitions from open regions are taken from the states which are visited either upon entering the regions or just before leaving them. Thus states arbitrarily close to the boundaries, or characterized by clock values with arbitrarily close fractional parts, may be visited in optimal runs. This motivates in our construction the choice of focusing on some particular subsets of regions that we call limit regions.

For defining a limit region we use the notation $a \lesssim b$ to denote that $a < b$ and $(b-a)$ is "very small" (i.e., it is very close to 0). For convenience, we do not give a quantitative definition of "very small", we only assume that in any context where we need to add up many $(b_i - a_i)$'s such that $a_i \lesssim b_i$, the differences $(b_i - a_i)$'s are sufficiently small to make their sum also very close to 0. Let α be a clock region and $(0 \approx_1 \bar{x}_1 \approx_2 \cdots \approx_h \bar{x}_h \approx_{h+1} 1)$ be the ordering of the fractional parts in α. A *limit region* α' of α is a convex subset of α defined by the clock constraints of α and the ordering of fractional parts $(0 \approx'_1 \bar{x}_1 \approx'_2 \cdots \approx'_h \bar{x}_h \approx'_{h+1} 1)$, where $\approx'_i$ is $=$ if $\approx_i$ is $=$ and is either $<$ or $\lesssim$, otherwise. In other words, we specify in the ordering of the fractional parts of α which $<$'s corresponds to "small differences" and which to "large" ones. For example, consider two clocks x and y, and let α be the open clock region defined by $0 < x < y < 1$. All the possible limit regions contained in α are reported in Fig. 5, where (1) corresponds to $0 \lesssim x < y < 1$, (2) corresponds to $0 < x \lesssim y < 1$, (3) corresponds to $0 < x < y \lesssim 1$, (4) corresponds to $0 \lesssim x \lesssim y < 1$, (5) corresponds to $0 \lesssim x < y \lesssim 1$, (6) corresponds to $0 < x \lesssim y \lesssim 1$. Clearly, there is no limit region corresponding to $0 \lesssim x \lesssim y \lesssim 1$ since this would imply that the difference between 0 and 1 is very small, and this is not the case.

For our assumptions, the actual value of the small differences is negligible. Thus, while computing the optimal cost of a run we can assume that it is actually 0. For this reason, we do not distinguish among states of a limit region which differ only in the values of the small differences, and thus we want to keep only the large differences of

the states visited in a run. We now introduce some notation before discussing how we store the large differences.

Let $s=(q,v)$ be a state of A and $(0 \approx_1 \bar{x}_1 \approx_2 \cdots \approx_N \bar{x}_N \approx_{N+1} 1)$ be the ordering of the fractional parts for the region containing the clock valuation v (notice that for $i=1,\ldots,N$ the operator $\approx_i$ is either $=$ or $<$, and $\approx_{N+1}$ is $<$). With $\vartheta(s)=(\vartheta_1,\ldots,\vartheta_{N+1})$ we denote the differences between consecutive values in the above ordering, that is $\vartheta_1=\bar{x}_1$, $\vartheta_{N+1}=1-\bar{x}_N$, and $\vartheta_i=\bar{x}_i-\bar{x}_{i-1}$ for $i=2,\ldots,N$. Notice that our graph construction is parameterized over the differences between consecutive values in the ordering of the fractional parts of a starting state s, thus in the following, tuple $(\vartheta_1,\ldots,\vartheta_{N+1})$ is used to denote the differences corresponding to a starting state. Moreover, for $i,j \leqslant N+1$, we denote by $I(i,j)$ the set of integers $\{i,\ldots,j-1\}$, if $i<j$, and $\{i,\ldots,N+1\} \cup \{1,\ldots,j-1\}$, otherwise.

The value of the large differences is given symbolically via a tuple of indices. These tuples reflect the starting state and the reset history of the computation. Recall that the values of parameters $\vartheta_1,\ldots,\vartheta_{N+1}$ give the differences of the fractional parts of the clocks in the starting state. Also, notice that for the runs we consider, if a state $(q',\langle y_1,\ldots,y_n\rangle)$ is reached, we have that $\bar{y}_{i+1}-\bar{y}_i$ is either "small" or given by the sum of consecutive such differences in the starting state. Thus, we augment each limit region and each region with tuple of indices $(i_1,\ldots,i_k)$ from $\{1,\ldots,N+1\}$ such that:

- k is the number of large differences in the ordering of the fractional parts;
- i_l corresponds to the lth large difference in this ordering, in the sense that for $l=1,\ldots,k-1$, the value of the l-th large difference is $\sum_{j \in I(i_l,i_{l+1})} \vartheta_j$, and the value of the kth large difference is $\sum_{j \in I(i_k,i_1)} \vartheta_j$;
- there exists a $d \in \{1,\ldots,k\}$ such that $i_{d+h} < i_{d+h+1}$ for $h=0,\ldots,k-1$, where the sums $(d+h+1)$ and $(d+h)$ are modulo k (i.e., shifting circularly all the indices for $d-1$ positions to the left we obtain an increasing sequence of indices).

We call such tuples *distance tuples*. We observe that the sum $\sum_{l \in I(i_k,i_1)} \vartheta_l$ gives also the time that is left for reaching the boundary of the current region from a state corresponding to a distance tuple $(i_1,\ldots,i_k)$.

Given a tuple of parameters $\vartheta=(\vartheta_1,\ldots,\vartheta_{N+1})$, we define the *parametric sub-region graph* of a timed automaton A relative to ϑ, denoted by $G_A(\vartheta)$, as the D-labeled directed graph (V,E) where V and E are defined as follows. The set of vertices V is the set of tuples $\langle q,\alpha,(i_1,\ldots,i_k)\rangle$ where q is a location, α is either a limit region or a clock region, and $(i_1,\ldots,i_k)$ is a distance tuple from $\{1,\ldots,N+1\}$. The set of edges E contains three types of edges: *time edges*, *immediate switches*, and *delayed switches*. Informally, time edges correspond to letting time elapse until the next boundary region is reached, immediate switches correspond to transitions taken in the current state, and delayed switches correspond to transitions taken at the "beginning" or at the "end" of the closest open region (this region, if it is an open region, and the next, otherwise). In the following, we formally define all of these three kinds of edges.

Time edges. Consider a vertex $v=\langle q,\alpha,(i_1,\ldots,i_h)\rangle$ and let $(0 \approx_1 \bar{y}_1 \approx_2 \cdots \approx_k \bar{y}_k \approx_{k+1} 1)$ be the ordering of the fractional parts in α. If for each $v \in \alpha$, $v(y_k)+1$ is not larger than the largest constant appearing in the timing constraints involving the clock variable y_k (i.e., when time elapses the first integer value reached by y_k is strictly smaller than this constant), then we add to E a time edge $v \xrightarrow{c} v'$ for

Table 1
$(0 \approx'_1 \bar{y}'_1 \approx'_2 \cdots \approx'_k \bar{y}'_k \approx'_{k+1} 1)$ denotes the ordering of the fractional parts in β, and $l = 2, \ldots, k$

	$\approx_1$	$\approx_{k+1}$	$\approx'_1$	$\approx'_2$	$\approx'_{l+1}$	$(j_1, \ldots, j_{h'})$	c
1.	$<$	$<$	$=$	$<$	$\approx_l$	$(i_h, i_2, \ldots, i_{h-1})$	$J_d(q) \cdot \sum_{l \in I(i_h, i_1)} \vartheta_l$
2.	$\lesssim$ or $=$	$<$	$=$	$<$	$\approx_l$	$(i_h, i_1, \ldots, i_{h-1})$	$J_d(q) \cdot \sum_{l \in I(i_h, i_1)} \vartheta_l$
3.	$<$	$\lesssim$	$=$	$<$	$\approx_l$	$(i_1, \ldots, i_h)$	0
4.	$\lesssim$ or $=$	$\lesssim$	$=$	$\lesssim$	$\approx_l$	$(i_1, \ldots, i_h)$	0

$v' = \langle q, \beta, (j_1, \ldots, j_{h'}) \rangle$ where β is the closest time-successor of α such that the conditions expressed by one of the rows of Table 1 are satisfied.

In the other case, time edges are defined in the same way except for the fact that clock y_k does not appear in the ordering of the fractional parts of v' since it has reached its highest constant.

In the Table 1, rows 1 and 2 captures the cases when the time to reach the following boundary region is not negligible (i.e., the difference between 1 and the largest fractional part is "large"). If the current first difference is "large" (row 1), then the first difference of the next vertex is the sum of the first and the last differences of the current vertex. Otherwise, it is just the last difference of the current vertex (row 2). Rows 3 and 4 capture the cases when the time to the following boundary region is "small".

To see an example of a time edge, consider a starting vertex $v = \langle q,\ 0 < x < y < z < 1,\ (1,2,3,4) \rangle$. We have that the first large difference is x_1, the second large difference is $(y_1 - x_1)$, the third large difference is $(z_1 - y_1)$ and the fourth large difference is $(1 - z_1)$. By row 1 of Table 1, we add to E a time edge from v to $v' = \langle q,\ 0 < x < y < 1 \wedge z = 1,\ (4,2,3) \rangle$ with cost $J_d(q) \cdot (1 - z_1)$. Since $I(4,2) = \{1,4\}$, v' corresponds to a state whose first large difference is $\sum_{i \in \{1,4\}} \vartheta_i = x_1 + 1 - z_1$. From $I(2,3) = \{2\}$ and $I(3,4) = \{3\}$, the second and the third large differences are $(y_1 - x_1)$ and $(z_1 - y_1)$ as for v. Thus the distance tuple $(4,2,3)$ correctly captures the fact that time $(1 - z_1)$ has elapsed and the distance in time from 0 to x is now $(1 - z_1 + x_1)$, the fractional part of z is 0, and all the other distances stay unchanged.

Immediate switches. Given two vertices $v = \langle q, \alpha, (i_1, \ldots, i_h) \rangle$ and $v' = \langle q', \beta, (j_1, \ldots, j_k) \rangle$, there is an immediate switch $v \xrightarrow{J_s(e)} v'$ if there exists an edge of $R(A)$ corresponding to e from $\langle q, \alpha' \rangle$ to $\langle q', \beta' \rangle$ such that α' and β' are, respectively, the regions of $R(A)$ containing α and β, and the sequence $(j_1, \ldots, j_k)$ is obtained from $(i_1, \ldots, i_h)$ by deleting, for $l = 1, \ldots, h-1$, all the indices i_{l+1}, such that all the clocks between the lth and the $(l+1)$th large difference (in the ordering of the fractional parts of α') are reset in e.

Continuing with the above example, consider vertex $v' = \langle q,\ 0 < x < y < 1 \wedge z = 1,\ (4,2,3) \rangle$ and suppose that $R(A)$ has an edge from $\langle q,\ 0 < x < y < 1 \wedge z = 1 \rangle$ to $\langle q',\ 0 = y < x < 1 \wedge z = 1 \rangle$. We add to E an immediate switch from v' to $v'' = \langle q',\ 0 = y < x < 1 \wedge z = 1,\ (4,2) \rangle$. Notice that the distance tuple $(4,2,3)$ gets updated to $(4,2)$ since clock y is reset in the transition and this is the only clock between the second

Table 2
$(0 \approx'_1 \bar{y}'_1 \approx'_2 \cdots \approx'_k \bar{y}'_k \approx'_{k+1} 1)$ denotes the ordering of the fractional parts in β, and $l = 2, \ldots, k$.

	$\approx_1$	$\approx_{k+1}$	$\approx'_1$	$\approx'_l$	$\approx'_{k+1}$	$(j_1, \ldots, j_{h'})$	c'
1.	$<$	$<$	$<$	$\approx_l$	$\lesssim$	$(i_h, i_2, \ldots, i_{h-1})$	$J_d(q) \cdot \sum_{l \in I(i_h, i_1)} \vartheta_l$
2.	$\lesssim$	$<$	$<$	$\approx_l$	$\lesssim$	$(i_h, i_1, \ldots, i_{h-1})$	$J_d(q) \cdot \sum_{l \in I(i_h, i_1)} \vartheta_l$
3.	$=$	$<$	$\lesssim$	$\approx_l$	$<$	$(i_1, \ldots, i_h)$	0
4.	$=$	$<$	$<$	$\approx_l$	$\lesssim$	$(i_h, i_1, \ldots, i_{h-1})$	$J_d(q) \cdot \sum_{l \in I(i_h, i_1)} \vartheta_l$
5.	$=$	$\lesssim$	$\lesssim$	$\approx_l$	$\lesssim$	$(i_1, \ldots, i_h)$	0

and the third large difference of v'. Moreover, the cost associated with this immediate switch is the cost of taking the A transition corresponding to the edge from $\langle q,\ 0 < x < y < 1 \wedge z = 1 \rangle$ to $\langle q',\ 0 = y < x < 1 \wedge z = 1 \rangle$.

Delayed switches. Given a vertex $v \in V$ as above, we add to E a delayed switch $v \xrightarrow{c} v''$ for any vertex $v'' \in V$ such that there exists an immediate switch $v' \xrightarrow{J_s(e)} v''$ and $c = c' + J_s(e)$, where $v' = \langle q, \beta, (j_1, \ldots, j_{h'}) \rangle$ and β is the closest time-successor of α such that the conditions expressed by one of the rows of Table 2 are satisfied.

In the above table, rows 1 and 2 capture the situation that we wait up to the end of the current open region and then take an immediate switch. If the current region is a boundary region, an immediate switch can be taken right after entering (row 3) or at the end of the following open region (rows 4 and 5). We observe also that each immediate switch, and thus each delayed switch, corresponds to a unique transition of the starting timed automaton. In the following, to refer to this transition simply as the transition corresponding to the immediate (resp. delayed) switch that we are considering.

Back to our running example, consider a vertex $v'' = \langle q',\ 0 = y < x < 1 \wedge z = 1,\ (4,2) \rangle$ and suppose that $R(A)$ has an edge from $\langle q',\ 0 = y < x < 1 \wedge z = 1 \rangle$ to $\langle q'',\ 0 < y < x < 1 \wedge z = y + 1 \rangle$, and let e be the corresponding transition of A. We add to E two delayed switches from v'': a delayed switch to $\langle q'',\ 0 \lesssim y < x < 1 \wedge z = y + 1,\ (4,2) \rangle$ (by row 3 of Table 2) and a delayed switch to $\langle q'',\ 0 < y < x \lesssim 1 \wedge z = y + 1,\ (2,4) \rangle$ (by row 4 of Table 2). Notice that no clock gets reset in the transition. Since the first switch corresponds to taking the transition as soon as the following open region is entered, the distance tuple $(4,2)$ stays unchanged, and the associated cost is $J_s(e)$. The second switch instead corresponds to taking the transition before leaving the following open region, the distance tuple $(4,2)$ thus gets updated to $(2,4)$ and the associated cost is $J_d(q')(z_1 - x_1) + J_s(e)$.

As an example of the given construction, we discuss a fragment of the graph $G_A(\vartheta)$ for the weighted timed automaton modeling the air-traffic control problem from Example 3 (see Fig. 6). For the sake of simplicity, we have marked with $1, \ldots, 5$ the vertices of $G_A(\vartheta)$ in Fig. 6, and we refer to them by these numbers. Consider vertex 1. Since in the timed automaton from Fig. 4 there is a transition from W_1 to W'_1 which resets clock x_1, we have in $G_A(\vartheta)$ an immediate switch from 1 to 2. Edges from 1 to 3 and from 1 to 4 are delayed switches obtained by the same transition above and,

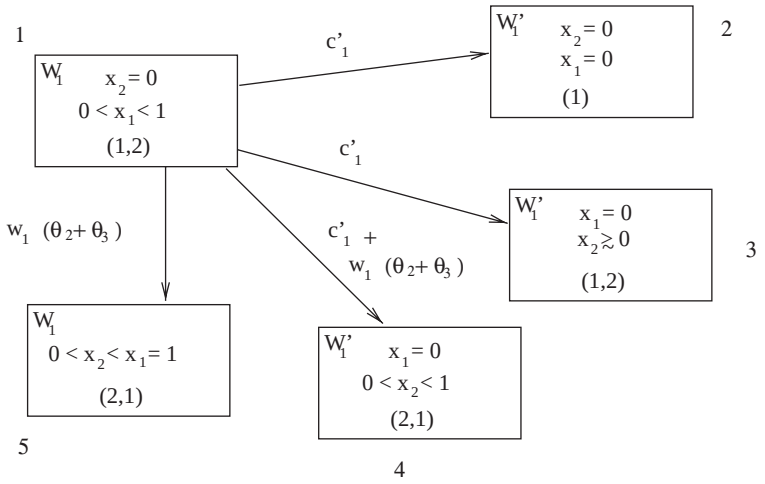

Fig. 6. A fragment of $G_A(\vartheta)$ for the weighted timed automaton from Example 3.

respectively, rows 3 and 4 of Table 2. The edge from 1 to 5 is a time edge and is defined by row 2 of Table 1.

We have the following lemma.

Lemma 3.1. *Given a timed automaton A with k_l locations and k_t transitions, the size of the parametric sub-region graph $G_A(\vartheta)$ is $O(|A|\, n\, 2^{2n}\, 2^{|\delta(A)|})$, where $|\delta(A)|$ denotes the length of the clock constraints of A. Moreover, the number of vertices of $G_A(\vartheta)$ is $O(k_l\, n\, 2^{2n}\, 2^{|\delta(A)|})$ and the number of edges of $G_A(\vartheta)$ is $O((k_l + k_t)\, n\, 2^{2n}\, 2^{|\delta(A)|})$.*

Proof. In [2] the authors proved for the region automaton that the size is $O(|A|\, 2^{|\delta(A)|})$, the number of vertices is $O(k_l\, 2^{|\delta(A)|})$, and the number of edges is $O(k_t\, 2^{|\delta(A)|})$. A simple counting argument gives that the number of ways to substitute $<$ with $\lesssim$ in the ordering of the fractional parts of a clock region is at most 2^{n+1} and the number of tuples of indices we use to represent the relative differences between the fractional parts is at most $(n+1)2^n$. Thus the number of vertices of $G_A(\vartheta)$ is $O(k_l\, n\, 2^{2n}\, 2^{|\delta(A)|})$. Moreover, from each vertex of $G_A(\vartheta)$ there is exactly an outgoing time edge and for each transition of the timed automaton leaving the corresponding location there are at most three edges (among delayed and immediate switches). Thus, the total number of time edges is exactly the number of vertices, while the total number of delayed and immediate switches is $O(k_t\, n\, 2^{2n}\, 2^{|\delta(A)|})$. Hence, we have that the number of edges of $G_A(\vartheta)$ is $O((k_l + k_t)\, n\, 2^{2n}\, 2^{|\delta(A)|})$. From the above observation we can also conclude that the total size of $G_A(\vartheta)$ is $O(|A|\, n\, 2^{2n}\, 2^{|\delta(A)|})$. □

We end this section with few observations on the parametric sub-region graph. The construction of $G_A(\vartheta)$ is general in the sense that it does not depend on the particular source zone S and target zone T of the problem, but only on the timed automaton.

Different choices of S and T will only affect the choice of the starting and the target vertices, while the graph $G_A(\vartheta)$ will stay unchanged. This allows us to use it for solving both the single-source optimal-reachability problem (for a fixed ϑ) and the optimal-reachability problem (ϑ belongs to a convex set). Moreover, for a given state $s=(q,v)$, we have corresponding vertices of $G_A(\vartheta(s))$ of the form $\langle q,\alpha,(i_1,\ldots,i_k)\rangle$, where $v\in\alpha$. Each edge is labeled by the actual cost of the corresponding "activity" in A, that is, for immediate switches we have just the cost of the A transition, for time edges the cost of spending time up to the end of the current region in the current A location, and for delayed switches the cost corresponding to the A transition plus the cost of spending time in the current location before the transition is taken.

4. Single-source optimal-reachability

In this section, we prove that the single-source optimal-reachability problem for timed automata can be reduced to the shortest-path problem on a weighted directed graph. We start introducing some notation and then prove the claimed result.

Let $s=(q,v)$ be a state of a weighted timed automaton A and $\vartheta(s)=(\vartheta_1,\ldots,\vartheta_{N+1})$, we denote by $g(s)$ a vertex $\langle q,\alpha,(1,\ldots,N+1)\rangle$ of $G_A(\vartheta(s))$ such that $v\in\alpha$. We want to define a set of runs of A that corresponds to a path π of $G_A(\vartheta(s))$, in the sense that, if for each of such runs we map the states to the regions and sub-regions they belong to, then we must obtain the path π. For this purpose, we need a measure of a "small" difference, and thus, of the distance from the border of the region when a delayed switch is taken. We will define a set of runs parameterized on an upper bound on such a distance.

Let ξ be a small positive real number arbitrarily close to 0 and s a state of A. Let $\pi=\langle q_0,\alpha_0,(i_{0,1},\ldots,i_{0,N_0})\rangle \xrightarrow{c_1} \langle q_1,\alpha_1,(i_{1,1},\cdots,i_{1,N_1})\rangle \xrightarrow{c_2} \cdots \xrightarrow{c_h} \langle q_h,\alpha_h,(i_{h,1},\ldots,i_{h,N_h})\rangle$ be a path of $G_A(\vartheta(s))$ starting at $g(s)$ (i.e., $g(s)=\langle q_0,\alpha_0,(i_{0,1},\ldots,i_{0,N_0})\rangle$. We denote by $R_\pi(\xi)$ the set of runs of A starting at s, that are obtained by replacing a step $(q_j,v_j)\xrightarrow[e_j]{t_j}(q_k,v_k)$ for the fragment of π from $\langle q_j,\alpha_j,(i_{j,1},\ldots,i_{j,N_j})\rangle$ through $\langle q_k,\alpha_k,(i_{k,1},\ldots,i_{k,N_k})\rangle$ such that

- $\langle q_{j-1},\alpha_{j-1},(i_{j-1,1},\ldots,i_{j-1,N_{j-1}})\rangle \xrightarrow{c_j} \langle q_j,\alpha_j,(i_{j,1},\ldots,i_{j,N_j})\rangle$ is either an immediate or a delayed switch;
- for $l=j\ldots,k-2$, $\langle q_l,\alpha_l,(i_{l,1},\ldots,i_{l,N_l})\rangle \xrightarrow{c_{l+1}} \langle q_{l+1},\alpha_{l+1},(i_{l+1,1},\ldots,i_{l+1,N_{l+1}})\rangle$ is a time edge;
- denoting $\langle q_{k-1},\alpha_{k-1},(i_{k-1,1},\ldots,i_{k-1,N_{k-1}})\rangle \xrightarrow{c_k} \langle q_k,\alpha_k,(i_{k,1},\ldots,i_{k,N_k})\rangle$ by e, e is either an immediate or a delayed switch corresponding to e_j. Moreover, if e is an immediate switch then $v_j+t_j\in\alpha_{k-1}$. If e is instead a delayed switch, then we have the following cases:
  - e is obtained from rows 1 and 2 of Table 2: t_j is such that $v_j+t_j\in\alpha_{k-1}$ and the largest fractional part of v_j+t_j according to the ordering of the fractional parts associated with α_{k-1} is greater than $(1-\xi)$.
  - e is obtained from rows 3–5 of Table 2: denoting by α' the time-successor of α_{k-1} which is first entered by letting time elapse, then $v_j+t_j\in\alpha'$. Moreover,

the largest fractional part in the ordering of the fractional parts associated with α' is greater than $(1-\xi)$, if e is obtained by rows 4 and 5 of Table 2, and the smallest fractional part is less than ξ, if e is obtained by rows 3 and 5 of Table 2.

Let s be a state of a weighted timed automaton A and $g(s)$ be $\langle q\ 0<x<y<1 \wedge z=1, (4,2,3)\rangle$. As an example of the above definition, consider the following path of $G_A(\vartheta)$ from $g(s)$:

$$\begin{aligned}\pi &= \langle q,\ 0<x<y<z<1,\ (1,2,3,4)\rangle \\ &\xrightarrow{c_1} \langle q,\ 0<x<y<1 \wedge z=1,\ (4,2,3)\rangle \\ &\xrightarrow{c_2} \langle q',\ 0=y<x<1 \wedge z=1,\ (4,2)\rangle \\ &\xrightarrow{c_3} \langle q'',\ 0=z \lesssim y<x<1,\ (4,2)\rangle.\end{aligned}$$

The first edge is a time edge, the second is an immediate switch corresponding to a transition e of A where clock y gets reset, and the third is a delayed switch corresponding to a transition e' of A where clock z gets reset. According to the above definition, the set $R_\pi(\xi)$ contains all the runs $(q,v) \xrightarrow{t} \xrightarrow{e} (q',v') \xrightarrow{t'} \xrightarrow{e'} (q'',v'')$ such that:

- $t=\vartheta_4$ (recall that $t=\sum_{j\in I(4,1)}\vartheta_j$ and $I(4,1)=\{4\}$),
- $0<t'<\xi$,
- $v'=\lambda(v)+t$ where λ is the reset function that resets clock y,
- $v''=\lambda'(v')+t'$ where λ' is the reset function that resets clock z.

In the following, we assume that ξ is a small positive real number arbitrarily close to 0. Directly from the definition of $G_A(\vartheta)$ and $R_\pi(\xi)$, we have the following property of $R_\pi(\xi)$.

Proposition 4.1. *Given a timed automaton A and a state $s=(q,v)$ of A, if π is a path of $G_A(\vartheta(s))$ from $g(s)$ of cost c_π then $R_\pi(\xi)$ is a set of runs of A such that for any $\varepsilon>0$ there exists an $r\in R_\pi(\xi)$ such that $|c_\pi - J(r)|<\varepsilon$.*

To complete our reduction we need the following lemma.

Lemma 4.2. *Given a run r of A from a state s to a target zone T, there exists a path π of $G_A(\vartheta(s))$ from $g(s)$ to a vertex corresponding to a state in T such that the cost of π is not larger than $J(r)$.*

Proof. Let r be a run $(q_0,v_0) \xrightarrow{t_1} \xrightarrow{e_1} (q_1,v_1) \xrightarrow{t_2} \xrightarrow{e_2} \cdots \xrightarrow{t_k} \xrightarrow{e_k} (q_k,v_k) \xrightarrow{t_{k+1}} (q_k,v_{k+1})$. Define R as the set of all A runs $r(d_1,\ldots,d_{k+1})=(q_0,v_0) \xrightarrow{d_1} \xrightarrow{e_1} (q_1,\gamma_1) \xrightarrow{d_2} \xrightarrow{e_2} \cdots\cdots \xrightarrow{d_k} \xrightarrow{e_k} (q_k,\gamma_k) \xrightarrow{d_{k+1}} (q_k,\gamma_{k+1})$, where $v_0+d_1\in[v_0+t_1]$, and $\gamma_i+d_{i+1}\in[v_i+t_{i+1}]$ for $i=1,\ldots,k$. That is, any $r'\in R$ differs from r only for the clock valuations at which transitions happen, but each of these valuations is in the same clock region as the corresponding valuation in r. Clearly, $r\in R$.

We recall that for a run r', $J(r') = J_s(r') + J_d(r')$. Thus, for $r' \in R$, $J_s(r') = J_s(r)$, while $J_d(r')$ may vary according to the values of $d_1, \ldots, d_{k+1}$. From the definition of R, we have that $d_1, \ldots, d_{k+1}$ vary in a convex polyhedron P (from clock constraints, regions, and resets of run $r(d_1, \ldots, d_{k+1})$ we obtain a system of linear inequalities over $d_1, \ldots, d_{k+1}$ which is satisfied if and only if $r(d_1, \ldots, d_{k+1}) \in R$). Since $J_d(r(d_1, \ldots, d_{k+1}))$ is a linear combination over $d_1, \ldots, d_{k+1}$, its minimum value over a convex polyhedron is reached at one of its corner points.

Let s be a state (q_0, v_0). To complete the proof, it is sufficient to show that for each tuple $(d_1, \ldots, d_{k+1})$ corresponding to a corner point of P, there exists a path π of $G_A(\vartheta(s))$ from $g(s)$ such that the cost of π is exactly the limit of $J(r(d'_1, \ldots, d'_{k+1}))$ for $(d'_1, \ldots, d'_{k+1})$ converging to $(d_1, \ldots, d_{k+1})$.

Let $(d_1, \ldots, d_{k+1})$ be a corner point of P, and for a small positive number ξ, let P_ξ be the set of points $(d'_1, \ldots, d'_{k+1}) \in P$ such that $d'_i \in]d_i - \xi, d_i + \xi[$ for $i = 1, \ldots, k+1$. Consider a run $r(d'_1, \ldots, d'_{k+1})$ for $(d'_1, \ldots, d'_{k+1}) \in P_\xi$. Observe that for $d_i > 0$, if $d'_i = d_i$ then the edge e_i is taken from a boundary region, otherwise e_i is taken from an open region within time ξ from an adjacent boundary region. In this second case, we abstract away the actual difference $|d_i - d'_i|$ assuming that e_i is taken from a limit region. Using this abstraction and from the definition of $G_A(\vartheta(s))$, it is possible to construct a path π such that $r' \in R_\pi(\xi)$ if and only if $r' = r(d'_1, \ldots, d'_{k+1})$ for $(d'_1, \ldots, d'_{k+1}) \in P_\xi$. From Proposition 4.1, we have that for any $\varepsilon > 0$ there exists an $r \in R_\pi(\xi)$ such that $|c_\pi - J(r)| < \varepsilon$, where c_π is the cost of π. Since this holds for any small positive ξ, letting ξ converge to 0 we obtain that the cost of π is the limit of $J(r(d'_1, \ldots, d'_{k+1}))$ for $(d'_1, \ldots, d'_{k+1})$ converging to $(d_1, \ldots, d_{k+1})$. □

We use Proposition 4.1 and Lemma 4.2 to prove the following two theorems.

Theorem 4.3. *Given a timed automaton A, a state s of A, a target zone T, π is a shortest path of $G_A(\vartheta(s))$ from $g(s)$ to a vertex corresponding to a state in T if and only if $R_\pi(\xi)$ is an approximation of an optimal run of A from s to T.*

Proof. First we consider the forward direction. Let π be a shortest path of $G_A(\vartheta(s))$ from $g(s)$ to a vertex corresponding to a state in T, and denote with c_π the cost of π, by Lemma 4.2 we have that $c_\pi \leqslant J(r)$ for any run r of A. By Proposition 4.1, we have that for all $\varepsilon > 0$, there exists an $r \in R_\pi(\xi)$ such that $c_\pi \leqslant J(r) < c_\pi + \varepsilon$ holds, and thus $R_\pi(\xi)$ is an approximation of an optimal run of A from s to T. Vice versa, suppose $R_\pi(\xi)$ is an approximation of an optimal run of A from s to T. Denoting by π' a shortest path of $G_A(\vartheta(s))$ from $g(s)$ to a vertex corresponding to a state in T, then by Lemma 4.2, the cost of π', say $c_{\pi'}$, is not larger than $J(r)$ for any $r \in R_\pi(\xi)$. By Proposition 4.1, we have that for all $\varepsilon > 0$, there exists an $r' \in R_{\pi'}(\xi)$ such that $|c_{\pi'} - J(r')| < \varepsilon$, and thus $c_{\pi'} \leqslant J(r') < c_{\pi'} + \varepsilon$. Since for any $r' \in R_{\pi'}(\xi)$ there exists an $r \in R_\pi(\xi)$ such that $J(r) \leqslant J(r')$ (costs of runs in $R_\pi(\xi)$ converge to the optimal cost), we have also that for all $\varepsilon > 0$, there exists an $r \in R_\pi(\xi)$ such that $c_{\pi'} \leqslant J(r) < c_{\pi'} + \varepsilon$, and thus $|c_{\pi'} - J(r)| < \varepsilon$. Since from Proposition 4.1 for all $\varepsilon > 0$ there exists an $r \in R_\pi(\xi)$ such that $|c_\pi - J(r)| < \varepsilon$, we have that $c_\pi = c_{\pi'}$ and π is a shortest path from $g(s)$ to a vertex corresponding to a state in T. □

As observed in Section 2, while an approximation of an optimal run always exists, an optimal run may not exist. In the following theorem, we give a necessary and sufficient condition for the existence of an optimal run, and a way to compute the optimal run when it exists.

Theorem 4.4. *Given a timed automaton A, a state s of A, a target zone T, there exists an optimal run of A from s to T if and only if for a shortest path π of $G_A(\vartheta(s))$ from $g(s)$ to a vertex corresponding to a state in T, there exists a run $r \in R_\pi(\xi)$ such that the cost of π is equal to $J(r)$. Moreover, r is an optimal run of A from s to T.*

Proof. Let r be an optimal run of A from s to T. We observe that by Lemma 4.2 we have that there exists a path π from $g(s)$ to a vertex corresponding to a state in T, such that, denoted by c_π the cost associated with π, $c_\pi \leqslant J(r)$. Moreover by Proposition 4.1, we have that for any $\varepsilon > 0$ there exists a run r' from s to T such that $c_\pi \leqslant J(r') < c_\pi + \varepsilon$. Since r is an optimal run from s to T, we have that $J(r) \leqslant J(r'')$ for any r'' from s to T, and thus by the above observations, that $c_\pi = J(r)$. Consider now the vice versa. Let π be a shortest path of $G_A(\vartheta(s))$ from $g(s)$ to a vertex corresponding to a state in T and let $r \in R_\pi(\xi)$ be such that $c_\pi = J(r)$. By Lemma 4.2, since π is optimal we get that $J(r) \leqslant J(r')$ for any r' from s to T, and thus, r is optimal. □

We recall that the parametric sub-region graph is parameterized over the differences of the fractional parts of the starting state. Clearly, if we fix a starting state, we obtain a weighted directed graph. By the results shown in this section, we can thus solve the single-source optimal-reachability problem for weighted timed automata.

Theorem 4.5. *Given a weighted timed automaton A with n clock variables, a source state s and a target zone T, the single-source optimal-reachability problem can be solved in $O(|\delta(A)|\,|A|\,n\,2^{2n}\,2^{|\delta(A)|})$ time, where $|\delta(A)|$ denotes the length of the clock constraints of A.*

Proof. An algorithm to solve the single-source optimal-reachability problem can be obtained by solving the shortest-path problem on $G_A(\vartheta(s))$. By Theorems 4.3 and 4.4, such an algorithm is correct. We recall that given a graph $G = (V, E)$ with non-negative weights, it is possible to solve the single-source shortest-path problem in $O(|E| + |V| \log |V|)$ by using the Dijkstra's algorithm with Fibonacci heaps [8]. From Lemma 3.1 we have that the number of vertices and the number of edges of $G_A(\vartheta(s))$ are bounded, respectively, by $O(k_l\, n\, 2^{2n}\, 2^{|\delta(A)|})$ and $O((k_l + k_t)\, n\, 2^{2n}\, 2^{|\delta(A)|})$, where k_l and k_t are, respectively, the number of locations and the number of locations is A. Thus, from $\log(k_l\, n\, 2^{2n+1}\, 2^{|\delta(A)|}) = O(|\delta(A)| + \log k_l)$, we can thus solve the single-source optimal-reachability problem in $O((k_l \log k_l + k_l|\delta(A)| + k_t + k_l) \cdot n\, 2^{2n}\, 2^{|\delta(A)|})$ time. Since $k_l + k_t + k_l \log k_l = O(|A|)$ we have that the single-source optimal-reachability problem can be solved in $O(|\delta(A)|\,|A|\,n\,2^{2n}\,2^{|\delta(A)|})$ time. □

5. Parametric shortest-path and optimal-reachability problems

In this section, we first define and solve a parametric shortest-path problem on weighted graphs, then we apply the resulting solution to the optimal-reachability problem in weighted timed automata. The algorithm for the first problem takes exponential time, and we provide a lower bound on the size of a solution to such a problem, that is exponential in the number of parameters. Combining this algorithm with the parametric sub-region graph construction given in Section 3, we obtain also an exponential-time algorithm for the optimal-reachability problem in weighted timed automata. As we will see, the time required by this algorithm is significantly larger than the time required by the algorithm given in Section 4 for the case of a single source state. In fact, the algorithm we propose here takes time exponential in the product of the number of clocks multiplied for the sum of the sizes of the clock constraints and of the largest weight. Finally, we show that if we restrict to weighted timed automata with only a single clock variable, we can improve the algorithm given for the general case. The running time of this algorithm does not depend on the size of the weights of the automaton.

5.1. Parametric shortest-path problem

Let $\{\vartheta_1,\ldots,\vartheta_k\}$ be a set of nonnegative real-valued parameters, and D be the set of linear expressions over $\{\vartheta_1,\ldots,\vartheta_k\}$ with nonnegative integer coefficients. Given a D-labeled directed graph $G=(V,E)$, the parametric shortest path problem from a vertex $u\in V$ to a target set $V_T\subseteq V$ is defined as the problem of determining a minimal set P of paths starting at u such that for each valuation of parameters $\vartheta_1,\ldots,\vartheta_k$, there exists a path in P which is a shortest path to a vertex in V_T from u. We recall that, in a D-labeled directed graph once the parameters are assigned with actual values, we obtain a directed graph with nonnegative real weights on the edges.

To solve the parametric shortest path problem, we give an algorithm that labels each vertex v of G (except for the vertices from V_T) with a set of pairs each consisting of a linear expression f over $\vartheta_1,\ldots,\vartheta_k$ and a vertex v'. The expression f gives the cost of a path to a target vertex and the coupled vertex v' is the next vertex on this path. In the labeling computed by our algorithm, given a valuation of the parameters and a vertex v, the minimum over the values corresponding to the expressions labeling v gives the cost of an optimal path from v to a vertex in V_T. We denote an expression $f(\vartheta_1,\ldots,\vartheta_k)=a_0+a_1\vartheta_1+\cdots+a_k\vartheta_k$, by the tuple of coefficients $(a_0,\ldots,a_k)$. We extend to pairs having as first component an expression and as second component a vertex, the ordering over expressions we introduced in Section 3. That is, let f,f' be two expressions, and v,v' be two vertices of G, we write $(f,v)\prec(f',v')$ if and only if $f\prec f'$. A set X of tuples of the form (f,v), where f is an expression and v is a vertex, is *minimized* (with respect to $\prec$) if for any $(f,v),(f',v')\in X$, (f,v) and (f',v') are not comparable with respect to $\prec$. In the following, we will lose the inner parenthesis in the tuples $((a_0,\ldots,a_k),v)$, and thus we will simply write $(a_0,\ldots,a_k,v)$ to denote the pair given by the expression $(a_0,\ldots,a_k)$ and the vertex v.

Our algorithm to solve the parametric shortest-path problem is shown in Fig. 7. In the initialization step (definition of φ_0), each vertex from the target set V_T is labeled

$$1.\ \varphi_0(v) = \begin{cases} \{(0,\ldots,0,\perp)\} & \text{if } v \in V_T \\ \emptyset & \text{otherwise} \end{cases}$$
$$2.\ \varphi_{j+1} = \textsc{Simplify}(\textsc{Update}(\varphi_j))$$

Fig. 7. Algorithm for solving the parametric shortest path problem.

with the null expression (all coefficients are 0) and a special symbol $\perp \notin V$, while the other vertices are labeled with the empty set. Notice that the labeling of a $v \in V_T$ is consistent with the fact that v is a target vertex thus the cost of a minimal path reaching the target is clearly 0 and we do not need to visit other vertices. In each iteration, we use procedures UPDATE and SIMPLIFY. For a labeling φ_j, we denote by $\textsc{Update}(\varphi_j)$ the labeling computed adding to $\varphi_j(v)$, for any vertex $v' \in V$, the tuples $(a'_0,\ldots,a'_k,v')$ such that

- $v \xrightarrow{f} v' \in E$;
- for $i=1,\ldots,k$, $a'_i = a_i + b_i$ where $f(\vartheta_1,\cdots,\vartheta_k) = b_0 + b_1\vartheta_1 + \cdots + b_k\vartheta_k$ and $(a_0,\ldots,a_k,v'') \in \varphi_j(v')$.

The function SIMPLIFY deletes from $\textsc{Update}(\varphi_j)$ all the tuples (f,v) such that $f' \prec f$ holds for at least a tuple $(f',v') \in \textsc{Update}(\varphi_j)$. Our algorithm halts when no more tuples are added to the labeling of G vertices.

It is easy to verify by induction the following property.

Lemma 5.1. *Given a D-labeled directed graph G and a vertex v, for any natural number j the labeling function φ_j is such that if $(f,v') \in \varphi_j(v)$ then there exists a path π from v to a vertex in V_T such that*
- *the first edge of π connects v to v', and*
- *the cost of π is given by f.*

The computational complexity of the algorithm in Fig. 7 is addressed in the following lemma.

Lemma 5.2. *Given a D-labeled graph $G=(V,E)$ and a target set $V_T \subseteq V$, Algorithm 1 from Fig. 7 runs in $O(l_{\max} \cdot |V| \cdot \ell^2)$ time, where $l_{\max}$ is the length of the longest simple path[2] of G, and ℓ is the size of the maximum number of incomparable tuples computed by Algorithm 1 for a vertex. Moreover, $\ell = O(l_{\max}^{k+1} \cdot \prod_{i=0}^{k} a_i^{\max})$, where $a_i^{\max}$ is the largest ith coefficient used in the expressions labeling the edges of G.*

Proof. Since the coefficients of the expressions we consider are nonnegative, any path π in G which is not simple has a cost that is not less than the cost of a path π' obtained deleting the cycles from π. Thus it is easy to show that for each vertex $v \in V$, $\varphi_j(v) = \varphi_h$ holds for any $j \geqslant h$, where h is the length of the longest simple path from v to a vertex in V_T. As a consequence, our algorithm will reach a fix-point after $l_{\max}$

[2] For length of a path we mean the number of edges of the path.

iterations. Since UPDATE and SIMPLIFY can be implemented in quadratic time in the size of the vertex labels and in each iteration we need to call them on each vertex, we have that the total time taken by our algorithm is $O(l_{\max} \cdot |V| \cdot \ell^2)$. To complete the proof, we only need to show the upper bound on ℓ. As a consequence of the fact that we can disregard cycles in the computation of the shortest paths, by Lemma 5.1 and the definition of SIMPLIFY, if $(a_0, \ldots, a_k, v') \in \varphi_j(v)$ for some $j \geqslant 0$, then $0 \leqslant a_i \leqslant l_{\max} \cdot a_i^{\max}$ holds for $i = 0, \ldots, k$. Thus we get an $O(l_{\max}^{k+1} \cdot \prod_{i=0}^{k} a_i^{\max})$ upper bound on the number of tuples that can appear in a labeling computed by our algorithm. □

The following theorem holds.

Theorem 5.3. *Given a D-labeled graph $G = (V, E)$ and a target set $V_T \subseteq V$, the parametric shortest-path problem from any vertex $u \in V$ to V_T can be solved in $O(l_{\max}^{2k+3} \cdot |V| \cdot (\prod_{i=0}^{k} a_i^{\max})^2)$ time, where $a_i^{\max}$ is the largest ith coefficient used in the expressions labeling the edges of G, and $l_{\max}$ is the length of the longest simple path of G.*

Proof. By Lemma 5.2, we only need to prove correctness for the algorithm in Fig. 7.

Let φ be the labeling computed by the algorithm in Fig. 7. By Lemma 5.1, to show the correctness of our algorithm it is sufficient to prove the following claim: given a vertex v, for each path π of G from v to a vertex in V_T, there exists a tuple $(f, v') \in \varphi(v)$ such that f is smaller than or equal to the cost of π. We observe that in our algorithm, tuples can be added only by UPDATE and deleted only by SIMPLIFY. Moreover, coefficients in the expressions labeling edges of G are nonnegative. Therefore, since $(0, \ldots, 0, \perp) \in \varphi_0(v)$, we have that $(0, \ldots, 0, \perp) \in \varphi(v)$, and the above claim trivially holds for any $v \in V_T$. Now let $v \notin V_T$. Suppose, by contradiction, that the above claim is violated. Thus, there exists a vertex v_1 such that there exist:

- a path $\pi = v_1 \xrightarrow{f_1} v_2 \cdots v_{m-1} \xrightarrow{f_{m-1}} v_m$, with $v_m \in V_T$ and $m > 1$, with the property that for all $(f, v') \in \varphi(v_1)$, the cost of π is either smaller than or not comparable to f;
- a tuple $(f', v'') \in \varphi(v_2)$ with the property that f' is either smaller than or equal to the cost of the path $v_2 \xrightarrow{f_2} v_3 \cdots v_{m-1} \xrightarrow{f_{m-1}} v_m$.

Since $(f', v'') \in \varphi(v_2)$, we have that $(f', v'') \in \varphi_j(v_2)$ for some index j and let h be the smallest such index. By definition, the set UPDATE$(\varphi_h(v_1))$ contains (f'', v_2), where $f'' = f' + f_1$, and thus f'' is either smaller than or equal to the cost of π. By our hypothesis, $(f'', v_2) \notin \varphi(v_1)$ must hold. Thus, (f'', v_2) must have been deleted at the jth iteration, for $j > h$. By definition of SIMPLIFY, there exists a tuple $(f''', v''') \in$ UPDATE $(\varphi_{j-1}(v_1))$ such that $f''' \prec f''$. As a consequence, f''' is smaller than or equal to the cost of π, and this proves that there must exist a tuple in $\varphi(v_1)$ such that the corresponding expression is smaller than or equal to the cost of π. Therefore, we have a contradiction and the above claim is proved. □

We observe that the path corresponding to a tuple (f, v), computed by the algorithm in Fig. 7, can be constructed by using the information contained in the last component of the tuples. We end this section giving an example of an instance of the parametric

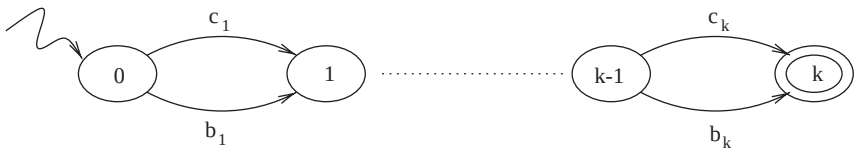

Fig. 8. A D-labeled directed graph.

shortest-path problem where the size of the solution is exponential in the number of parameters.

Example 4. Consider the graph in Fig. 8, where $c_i = a_i \cdot \vartheta_i$, and assume that $a_i > b_i$ for $i = 1,\ldots,k$. We want to compute the shortest paths from vertex 0 to vertex k. Let $I \subseteq \{1,\ldots,k\}$, we define $f_I(\vartheta_1,\ldots,\vartheta_k)$ as $\sum_{i\in I} a_i\vartheta_i + \sum_{i\notin I} b_i$. Clearly, for any path π from 0 to k, there exists a set I such that f_I gives the cost of π, and vice versa. We observe that for the parameter assignment ϑ defined by $\vartheta_i = 0$, for $i \in I$, and $\vartheta_i = 1$, otherwise, the unique shortest path from 0 to k is the one corresponding to f_I. To see this, consider a set $I' \neq I$. We have that, according to ϑ, $f_I - f_{I'}$ is $\sum_{i\in I\setminus I'}(-b_i) + \sum_{i\in I'\setminus I}(b_i - a_i)$. Since $a_i > b_i$, $f_I - f_{I'} < 0$ and thus $f_I < f_{I'}$. Thus we have that each path from 0 to k is a shortest path for some parameter valuation, and thus the solution to the considered instance of the parametric shortest-path problem has size exponential in the number of parameters.

5.2. Optimal-reachability problem

In this section, we deal with the optimal-reachability problem. We first give the result for the general case, then we discuss how to obtain a faster algorithm for the case of weighted timed automata with only one clock.

Combining the construction of the parametric sub-region graph and the solution to the parametric shortest path problem we have the following result.

Theorem 5.4. *Given a weighted timed automaton A, a source zone S and a target zone T, the optimal-reachability problem can be solved in $O(K^{2n+6} \cdot w_{\max}^{2n+4})$ time, where $K = O(|A|\, n\, 2^{2n}\, 2^{|\delta(A)|})$, $|\delta(A)|$ denotes the length of the clock constraints of A, n is the number of clocks, and $w_{\max}$ is the largest weight in A.*

Proof. By constructing the parametric sub-region graph, we can reduce the optimal-reachability problem to the parametric shortest-path problem. Since S may contain many regions, an instance of the optimal-reachability problem reduces to m instances of the parametric shortest-path problem. However, the iterative algorithm in Fig. 1 needs to be executed only once since it will compute the shortest paths from each vertex to the target. By Theorems 4.3, 4.4, and 5.3, we have that the algorithm obtained as described above is correct, in the sense that solves the optimal-reachability problem. Since the length of the longest simple path is bounded above by the size of the graph,

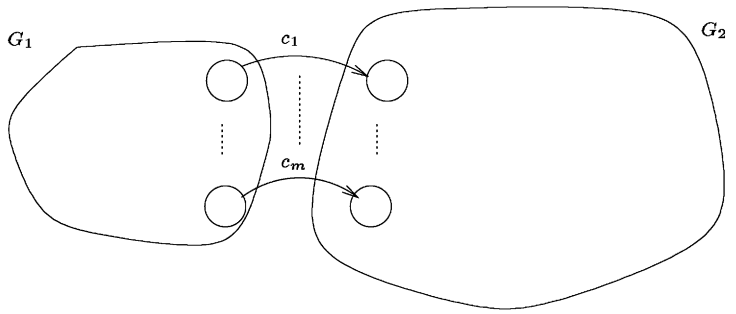

Fig. 9. Simplification of the parametric sub-region graph for a 1-clock weighted timed automaton.

the result on the upper bound on the time required by this algorithm follows directly from Lemma 3.1 and Theorem 5.3. □

For timed automata with just one clock variable, the above result gives an $O(K^8 \cdot w_{\max}^6)$ time upper bound to the optimal-reachability problem, for K denoting the size of the corresponding parametric sub-region graph. As shown in the next theorem, for such automata we can design a more efficient solution.

Theorem 5.5. *Given a weighted timed automaton A with a single clock variable x, a source zone S, and a target zone T, the optimal-reachability problem can be solved in $O(K^3 + m \cdot l_{\max} \cdot |A|^3)$ time, where $K = O(|A|\, 2^{|\delta(A)|})$, $|\delta(A)|$ denotes the length of the clock constraints of A, m is the number of regions in S, and $l_{\max}$ is the length of the longest simple path in A.*

Proof. Since we want to solve the problem of determining the optimal runs from any state of the source zone S to a state of a target zone T, in $G_A(\vartheta)$ we consider as tuple of parameters ϑ only tuples of values given by $\vartheta(s)$ for a state $s \in S$. Thus it holds that $\vartheta_1 + \cdots + \vartheta_{N+1} = 1$ and we can eliminate a parameter by the substitution $\vartheta_{N+1} = 1 - \sum_{i=1}^{N} \vartheta_i$. For a weighted timed automaton A with only one clock, we can thus consider only the parameter ϑ_1 in the parametric sub-region graph corresponding to A. Furthermore, we can simplify this graph to have the structure shown in Fig. 9, with two disjoint subgraphs G_1 and G_2, and the following properties. Graph G_1 contains all the vertices whose distance tuple is $(1,2)$ and that are reachable from $g(s)$ for some $s \in S$. All the other vertices with distance tuple either $(1,2)$ or $(2,1)$, can be deleted along with the related edges. This is possible since they are not reachable from any source state. The remaining vertices are in G_2. Notice that there might be edges connecting a vertex in G_1 to a vertex in G_2, but not vice versa. Denote these edges by $e_1, \ldots, e_h$. In Fig. 9, we label them, respectively, by $c_1, \ldots, c_h$ where for $i = 1, \ldots, h$ we have the following cases:

(1) $c_i = a(1 - \vartheta_1)$ and the related edge is a time edge,
(2) $c_i = a(1 - \vartheta_1) + b$ and the related edge is a delayed switch, and

(3) c_i is a weight of an A transition e on which x is reset and the related edge is an immediate switch corresponding to e.

Denote by S' the set of vertices of G_2 that are linked to a vertex of G_1 simply by an edge. We can solve the optimal-reachability problem in two steps:

- we first solve the shortest-path problem from any vertex in S';
- denote by s_i the vertex in S' which is an endpoint of e_i for $i=1,\ldots,h$; we apply the algorithm in Fig. 7 to a graph G' obtained adding to G_1 the edges e_i with costs $c'_i = c_i + c''_i$ for $i=1,\ldots,h$, where c''_i is the cost of the shortest path from s_i computed in the first part of the algorithm. While solving the problem on G' the vertices from S' are also considered targets.

It is easy to verify that parameter ϑ_1 does not appear in any of the edges of the graphs G_1 and G_2. Thus, the first step of the above algorithm can compute a shortest path from any of the vertices in S' in $O(|G_2|^3)$ time (all-pairs shortest paths). Since the size of G_2 is $O(|A|\,2^{|\delta(A)|})$, this phase takes $O(K^3)$.

Now, let k_l and k_t be, respectively, the number of locations and the number of transitions of A. The cost of any of the shortest paths computed above is a precise value and not an expression over ϑ_1. As a consequence, the second step needs only to consider linear expressions over $(1-\vartheta_1)$ where the coefficient of $(1-\vartheta_1)$ is either 0 or the value of one of the weights associated to an A location. This reduces the number of incomparable pairs, that can be in a set labeling a vertex of G', to the number of different weights of A locations augmented by 1, and thus it is $O(k_l)$. Moreover, the number of vertices in G' is h plus the number of vertices in G_1. By definition, in G_1 we have only immediate switches and delayed switches which are not time consuming. These switches do not correspond to transitions of A that reset x. Thus, by a simple counting argument, we have that the number of vertices of G_1 is $O(m \cdot k_l)$ (recall that m is the number of regions in S). Since h is $O(m \cdot k_t)$, the number of vertices in G' is $O(m \cdot (k_l + k_t))$. Furthermore, the length of the longest simple path in G' is $O(l_{\max})$ (recall that in G' there are no time edges and switches do not reset x). Thus, applying to G' the algorithm in Fig. 7, by Lemma 5.2 we get that the total time for this phase is $O(m \cdot l_{\max} \cdot (k_l + k_t) \cdot k_l^2)$. Hence, the total algorithm takes $O(K^3 + m \cdot l_{\max} \cdot (k_l + k_t) \cdot k_l^2)$ time. From $k_l + k_t = O(|A|)$, we obtain the desired bound. □

6. Conclusions

In this paper, we have dealt with the optimal-reachability problem for weighted timed automata. We have presented an approach consisting of reducing this problem to the parametric shortest-path problem on directed graphs. This translation takes exponential time in the size of the clock constraints. To solve the parametric shortest-path problem, we have given an algorithm that takes time exponential in the product of the size of the largest coefficient used in the expressions and the number of parameters. We have also shown that a solution to this second problem may have size exponential in the number of parameters. Our reduction combined with this algorithm gives an algorithm for solving the optimal-reachability problem on weighted timed automata that takes time exponential in $O(n(|\delta(A)| + |w_{\max}|))$, where n is the number

of clocks, $|\delta(A)|$ is the size of the clock constraints and $|w_{\max}|$ is the size of the largest weight.

When we restrict to a single source state, the same reduction translates an instance of the optimal-reachability problem to an instance of the standard shortest-path problem. Using an efficient algorithm to solve the back-end problem, we thus obtain an algorithm for the single-source optimal reachability problem on weighted timed automata that takes time exponential only in the size of the clock constraints.

We have also given a different algorithm (still based on the parametric sub-region graph) to solve the general problem for weighted timed automata with only one clock. The running time of this algorithm improves that of the algorithm given for the general case by a constant factor in the exponent, and is independent of the size of the weights of the automaton.

The best-known lower bound for the considered problems is PSPACE-hard, which is directly obtained from the complexity of reachability in timed automata [2]. It would be interesting to close this complexity gap.

A generalized version of the optimal-reachability problem is the problem of synthesizing an optimal controller. The *optimal-control synthesis problem* can be informally stated as the problem of designing a control which is able to drive, at a minimum cost, a system (usually called *plant*) into a given target zone. In the literature, control synthesis problems have been considered in the context of discrete automata [6,16], timed automata [3,4,13], linear hybrid automata [17], and general hybrid systems [12,15]. The design of an optimal control for hybrid systems is, in general, undecidable. The approach presented in this paper, does not generalize to solving the optimal-control synthesis problem for weighted timed automata. This problem is solved for acyclic weighted timed automata in [11]. To the best of our knowledge, a solution for the general case is still an open problem.

Acknowledgements

This work was partially supported by the DARPA/ITO MoBIES grant F33615-00-C-1707, the NSF Career award CCR97-34115, the NSF award CCR99-70925, the SRC award 99-TJ-688, the MIUR grant MEFISTO.

References

[1] R. Alur, C. Courcoubetis, T.A. Henzinger, Computing accumulated delays in real-time system, in: Proc. 5th Internat. Conf. on Computer-Aided Verification, CAV'93, Lecture Notes in Computer Science, Vol. 697, Springer, Berlin, 1993, pp. 181–193.

[2] R. Alur, D.L. Dill, A theory of timed automata, Theoret. Comput. Sci. 126 (1994) 183–235.

[3] E. Asarin, O. Maler, As soon as possible: time optimal control for timed automata, in: Proc. 2nd Internat. Workshop on Hybrid Systems: Computation and Control, Lecture Notes in Computer Science, Vol. 1569, Springer, Berlin, 1999, pp. 19–30.

[4] E. Asarin, O. Maler, A. Pnueli, Symbolic controller synthesis for discrete and timed systems, in: Proc. 2nd Internat. Workshop on Hybrid Systems, Lecture Notes in Computer Science, Vol. 999, Springer, Berlin, 1995, pp. 1–20.

[5] G. Behrman, T. Hune, A. Fehnker, K. Larsen, P. Pettersson, R. Romijn, F. Vaandrager, Minimum-cost reachability for priced timed automata, in: Proc. 4th Internat. Workshop on Hybrid Systems: Computation and Control, HSCC'01, Lecture Notes in Computer Science, Vol. 2034, Springer, Berlin, 2001, pp. 147–161.
[6] A. Church, Logic, arithmetic, and automata, in: Proc. Internat. Cong. of Mathematics, Stockholm, Sweden, 1962, pp. 23–35.
[7] C. Courcoubetis, M. Yannakakis, Minimum and maximum delay problems in real-time systems, in: Proc. 3rd International Conference on Computer Aided Verification, Lecture Notes in Computer Science, Vol. 575, Springer, Berlin, 1991, pp. 399–409.
[8] M. Fredman, R. Tarjan, Fibonacci heaps and their uses in improved network optimization algorithms, J. Assoc. Comput. Mach. 34 (3) (1987) 596–615.
[9] Y. Kesten, A. Pnueli, J. Sifakis, S. Yovine, Decidable integration graphs, Inform. and Comput. 150 (2) (1999) 209–243.
[10] K.G. Larsen, G. Behrman, E. Brinksma, A. Fehnker, T. Hune, P. Petersson, J. Romijn, As cheap as possible: efficient cost-optimal reachability for priced timed automata, in: Proc. 13th Internat. Conf. on Computer aided Verification, CAV'01, Lecture Notes in Computer Science, Vol. 2102, Springer, Berlin, 2001, pp. 493–505.
[11] S. La Torre, S. Mukhopadhyay, A. Murano, Optimal-reachability and control for acyclic weighted timed automata, in: Proc. 2nd IFIP Internat. Conf. Theoretical Computer Science, IFIP TCS'02, Kluwer Academic Publishers, Dordrecht, 2002, pp. 498–510.
[12] J. Lygeros, C. Tomlin, S.S. Sastry, Controllers for reachability specifications for hybrid systems, Automatica 35 (3) (1999) 349–370.
[13] O. Maler, A. Pnueli, J. Sifakis, On the synthesis of discrete controllers for timed systems, in: Proc. 12th Annu. Symp. on Theoretical Aspects of Computer Science, STACS'95, Lecture Notes in Computer Science, Vol. 900, Springer, Berlin, 1995, pp. 229–242.
[14] P. Niebert, S. Tripakis, S. Yovine, Minimum-time reachability for timed automata, in: Proc. 8th IEEE Mediterranean Conf. on Control and Automation, Rio, Greece, 2000.
[15] O. Shakernia, G.J. Pappas, S. Sastry, Decidable controller synthesis for classes of linear systems, in: Proc. 3rd Internat. Workshop on Hybrid Systems: Computation and Control, HSCC'00, Lecture Notes in Computer Science, Vol. 1790, Springer, Berlin, 2000, pp. 407–420.
[16] W. Thomas, On the synthesis of strategies in infinite games, in: 12th Annu. Symp. on Theoretical Aspects of Computer Science, STACS'95, Lecture Notes in Computer Science, Vol. 900, Springer, Berlin, 1995, pp. 1–13.
[17] H. Wong-Toi, The synthesis of controllers for linear hybrid automata, in: Proc. 36th IEEE CDC, San Diego, CA, December, 1997.