Proceedings of the 8th IEEE Mediterranean Conference
on Control and Automation (MED 2000)
Rio, Patras, GREECE 17-19 July, 2000

# Minimum-Time Reachability for Timed Automata

Peter Niebert*, Stavros Tripakis† and Sergio Yovine*

June 9, 2000

## Abstract

The problem of minimum-time reachability for timed automata is: given an automaton, and initial state $q_0$ and a target state $q_f$, find whether a run from $q_0$ to $q_f$ exists, and if yes, a minimum time run. We show that this problem can be solved by examining acyclic paths in a forward reachability graph generated on-the-fly from the timed automaton. Based on this result, we then propose three algorithms with different complexities.

## 1 Introduction

Given a timed automaton $A$, an initial state $q_0$, and a target state $q_f$, we are interested in the *minimum-time reachability* problem:

> Is there a run of $A$ starting from some state in $q_0$ and ending to some state in $q_f$ ? If so, find such a run which consumes minimum-time.

This problem is useful for a number of applications, including the computation of optimal (minimum-time) schedulers for batch-plants [NY00].

In [AM99], a solution to this problem (as well as the more general minimum-time controller-synthesis problem) is presented. The solution uses a backward fixpoint computation, which might unnecessarily explore unreachable states. Backward reachability is also undesirable in case the system is not a "pure" timed automaton (only control states and clocks) but also contains discrete variables (e.g., booleans and bounded integers)[1], which is the typical case. This motivates a study for algorithms based on forward reachability. In this paper, we present three such algorithms, which are based on the so-called *simulation graph*.

The basic result of our paper is that cycles in the simulation graph need not be examined when finding minimum-time runs. More precisely, Theorem 1 (see section 3) states that if a run from $q_0$ to $q_f$ exists, then there exists a minimum-time run from $q_0$ to $q_f$, that does not traverse the same node in the simulation graph twice. We note that this property does not hold at the timed-automata level. For example, in the automaton of figure 1, there is a run along $a_1\ a_2\ c\ b_1\ b_2$ which is shorter than any run along $a_1\ a_2\ b_1\ b_2$.

Based on Theorem 1, we propose three algorithms to solve the minimum-time reachability problem for timed automata: (1) an algorithm using a backward Bellman-Ford iteration on the simulation graph (the advantage over the fixpoint computation is first, that the number of iterations here is bounded by the maximal length of a path reaching a target node in the simulation graph, and second, that only reachable states are considered); (2) an algorithm which uses an additional clock to keep track of the minimum time to the target and updates this time on-the-fly; (3) an algorithm using binary iteration on the upper bound to the target.

All three algorithms have worst-case complexity which is worse than polynomial in the size of the simulation graph. It remains to be seen how well these algorithms perform in practice. An open question also remains: is this the best we can do ? We are mostly interested in the complexity of the problem in the size of the simulation graph, rather than in the size of the automaton. For the latter, it is well-known that reachability is PSPACE-complete [ACD93], therefore, minimum-time reachability is PSPACE-hard. In practice, however, the simulation graph has been proven a reasonable-size structure to work with. Reachability is linear in the size of the simulation graph, but it remains to be seen if minimum-time reachability can also be solved in polynomial time in the size of the simulation graph.

The output of the algorithms is a path in the simulation graph. From this path, a concrete run of the automaton can be obtained using the techniques described in [Tri99].

The paper is organized as follows. Section 2 gives the background on timed automata (can be safely skipped). In section 3 we present the simulation graph and prove our main theorem. This section introduces some important properties of the simulation graph in terms of relation between sets of states (zones) and timing constraints, which may be relevant for proving statements in more general settings. In section 4 we give the algorithms. Section 5 concludes.

*Verimag.
†Verimag and UC Berkeley.

[1] For example, the successor of state $s$ after setting discrete variable $i := 5$ is a single state $s'$. On the other hand, there are as many predecessors of a state $s$ with respect to $i := 5$ as the domain of $i$.

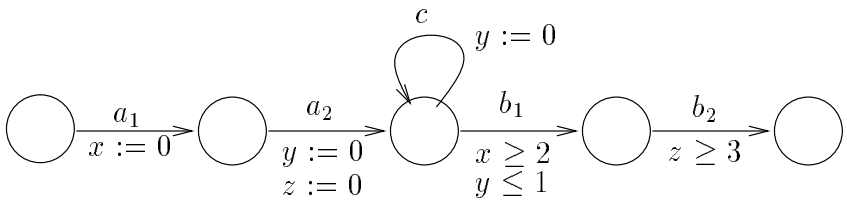


Figure 1: Loops can help in gaining time.

# 2 Background

## 2.1 Clocks and Polyhedra

**Clocks and valuations.** Let $\mathsf{R}$ be the set of non-negative reals and $\mathcal{X} = \{x_1, ..., x_n\}$ be a set of variables in $\mathsf{R}$, called *clocks*. An $\mathcal{X}$-*valuation* is a function $\mathbf{v} : \mathcal{X} \mapsto \mathsf{R}$. We write $\mathbf{0}$ for the valuation that assigns zero to all clocks. For some $X \subseteq \mathcal{X}$, $\mathbf{v}[X := 0]$ is the valuation $\mathbf{v}'$, such that $\forall x \in X \ . \ \mathbf{v}'(x) = 0$ and $\forall x \notin X \ . \ \mathbf{v}'(x) = \mathbf{v}(x)$. For every $\delta \in \mathsf{R}$, $\mathbf{v} + \delta$ (resp. $\mathbf{v} - \delta$) is a valuation such that for all $x \in \mathcal{X}$, $(\mathbf{v}+\delta)(x) = \mathbf{v}(x) + \delta$ (resp. $(\mathbf{v} - \delta)(x) = \mathbf{v}(x) - \delta$). Given $c \in \mathsf{N}$, two valuations $\mathbf{v}$ and $\mathbf{v}'$ are called $c$-*equivalent* if:

- for any clock $x$, either $\mathbf{v}(x) = \mathbf{v}'(x)$, or $\mathbf{v}(x) > c$ and $\mathbf{v}'(x) > c$;
- for any pair of clocks $x, y$, either $\mathbf{v}(x) - \mathbf{v}(y) = \mathbf{v}'(x) - \mathbf{v}'(y)$, or $|\mathbf{v}(x) - \mathbf{v}(y)| > c$ and $|\mathbf{v}'(x) - \mathbf{v}'(y)| > c$.

**Polyhedra.** An *atomic constraint* on $\mathcal{X}$ is an expression of the form $x \sim c$ or $x - y \sim c$, where $x, y \in \mathcal{X}$, $\sim \in \{\leq, \geq\}$ and $c \in \mathsf{N}$. An $\mathcal{X}$-valuation $\mathbf{v}$ *satisfies* the constraint $x \sim c$ if $\mathbf{v}(x) \sim c$; $\mathbf{v}$ satisfies $x - y \sim c$ if $\mathbf{v}(x) - \mathbf{v}(y) \sim c$.

An $\mathcal{X}$-*hyperplane* is a set of valuations satisfying an atomic clock constraint. The class $\mathcal{H}_{\mathcal{X}}$ of $\mathcal{X}$-*polyhedra* is defined as the smallest subset of $2^{\mathsf{R}^{\mathcal{X}}}$ which contains all $\mathcal{X}$-hyperplanes and is closed under set union, intersection and complementation.

We often use the following notation for polyhedra: we write $x \leq 5$ for the hyperplane defined by the constraint $x \leq 5$, $x \leq 5 \wedge y = 2$ for the polyhedron defined as the intersection of $x \leq 5$ and $y = 2$, and so on. We also write $\mathsf{true}$ for $\mathsf{R}^{\mathcal{X}}$, $\mathsf{false}$ for $\emptyset$ and $\mathsf{zero}$ for $\{\mathbf{0}\}$.

A polyhedron $Z$ is called *convex* if for all $\mathbf{v}_1, \mathbf{v}_2 \in Z$, for any $0 < \delta < 1$, $\delta\mathbf{v}_1 + (1 - \delta)\mathbf{v}_2 \in Z$. A polyhedron is convex iff it can be defined as the intersection of a finite number of hyperplanes.

A (convex) $\mathcal{X}$-polyhedron is an $\mathcal{X}$-*box* if it is defined as the intersection of hyperplanes of the form $x \sim c$, $x \in \mathcal{X}$, $\sim \in \{\leq, \geq\}$ and $c \in \mathsf{N}$.

**Operations on polyhedra.** By definition, intersection, union and complementation are well-defined operations on polyhedra. Polyhedra difference is defined via complementation as: $Z_1 \setminus Z_2 = Z_1 \cap \overline{Z_2}$. The test for inclusion $Z_1 \subseteq Z_2$ is equivalent to $Z_1 \setminus Z_2 = \emptyset$. We now define some more operations which will be used in the sequel. Examples of operations are shown in figure 2.

**$c$-closure.** This operation is necessary for guaranteeing termination of the reachability algorithms. Given a convex $\mathcal{X}$-polyhedron $Z$ and a natural constant $c$, the $c$-*closure* of $Z$, denoted $\mathsf{close}(Z, c)$, is the greatest convex $\mathcal{X}$-polyhedron $Z' \supseteq Z$, such that for all $\mathbf{v}' \in Z'$ there exists $\mathbf{v} \in Z$ such that $\mathbf{v}$ and $\mathbf{v}'$ are $c$-equivalent. Intuitively, $Z'$ is obtained by $Z$ by "ignoring" all bounds which involve constants greater than $c$. $Z$ is said to be $c$-*closed* if $\mathsf{close}(Z, c) = Z$.

**Lemma 1** *1. If $Z$ is $c$-closed then it is $c'$-closed, for any $c' > c$.*

*2. If $Z_1$ and $Z_2$ are $c$-closed then $Z_1 \cap Z_2$ is also $c$-closed.*

*3. For any $Z$, there exists a constant $c$ such that $Z$ is $c$-closed.*

From now on, $c_{max}(Z)$ will denote the smallest constant $c$ such that $Z$ is $c$-closed.

**Lemma 2** *For any constant $c$, there is a finite number of $c$-closed $\mathcal{X}$-polyhedra.*

**Clock resets.** We define the operations $Z[Y := 0]$ and $[Y := 0]Z$ of *forward* and *backward clock reset*, respectively, as follows:

$$Z[Y := 0] \stackrel{\text{def}}{=} \{\mathbf{v}[Y := 0] \mid \mathbf{v} \in Z\}$$
$$[Y := 0]Z \stackrel{\text{def}}{=} \{\mathbf{v} \mid \mathbf{v}[Y := 0] \in Z\}$$

Intuitively, $Z[Y := 0]$ contains all valuations which can be obtained from some valuation in $Z$ by resetting clocks in $Y$. It contains all valuations which, after resetting clocks in $Y$, yield a valuation in $Z$.

**Time elapse.** We define the operations of *backward* and *forward time elapse* of an $\mathcal{X}$-polyhedron $Z$ to be

the $\mathcal{X}$-polyhedra $\swarrow Z$ and $\nearrow Z$, respectively, such that:

$$\mathbf{v}' \in \swarrow Z \quad \text{iff} \quad \exists \delta \in \mathsf{R} \,.\, \mathbf{v}' + \delta \in Z$$
$$\mathbf{v}' \in \nearrow Z \quad \text{iff} \quad \exists \delta \in \mathsf{R} \,.\, \mathbf{v}' - \delta \in Z$$

**Projection.** Given an $\mathcal{X}$-polyhedron $Z$ and $Y \subseteq \mathcal{X}$, $Z/_{\mathcal{Y}}$ is defined to be the $\mathcal{Y}$-polyhedron $Z' = \{\mathbf{v}' \mid \exists \mathbf{v} \in Z \,.\, \forall y \in \mathcal{Y} \,.\, \mathbf{v}'(y) = \mathbf{v}(y)\}$.

**Preservation of convexity.** The following result is easy to derive from the definitions.

**Lemma 3** *If $Z$ is convex and $Y \subseteq \mathcal{X}$ then $Z[Y := 0]$, $[Y := 0]Z$, $\swarrow Z$, $\nearrow Z$ and $Z/_{\mathcal{Y}}$ are also convex.*

**Effective representation.** Convex $\mathcal{X}$-polyhedra can be effectively represented as a *difference bound matrix* (DBM) [Dil89]. Semantic operations on polyhedra can be implemented as syntactic DBM transformations. The cost of these operations is at most $O(k^3)$, where $k$ is the number of clocks. A non-convex polyhedron $Z$ is the union of $k$ convex polyhedra $Z = Z_1 \cup \cdots \cup Z_k$. Therefore, $Z$ can be represented (in a non-canonical way) as a list of $k$ DBMs, one for each polyhedron $Z_1, ..., Z_k$.

## 2.2 Timed Automata

A *timed automaton* (TA) [ACD93, HNSY94] is a tuple $A = (\mathcal{X}, Q, E)$, where:

- $\mathcal{X}$ is a finite set of clocks.
- $Q$ is a finite set of *discrete states*.
- $E$ is a finite set of *jumps* of the form $a = (q, Z, X, q')$. $q, q' \in Q$ are the *source* and *target* discrete states. $Z$ is a conjunction of atomic constraints on $\mathcal{X}$ defining an $\mathcal{X}$-box, called the *guard* of $e$. $X \subseteq \mathcal{X}$ is a set of clocks to be reset on the jump.

Given a discrete state $q$, we write $\mathsf{in}(q)$ (resp. $\mathsf{out}(q)$) for the set of jumps of the form $(\_, \_, \_, q)$ (resp. $(q, \_, \_, \_)$).

$c_{max}(A)$ is defined as the maximum of $c_{max}(Z)$, where $Z$ is a guard of $A$.

A *state* of $A$ is a pair $(q, \mathbf{v})$, where $q \in Q$ is a location, and $\mathbf{v}$ is a valuation. Two states $(q, \mathbf{v}_1)$ and $(q, \mathbf{v}_2)$ are $c$-equivalent if $\mathbf{v}_1$ and $\mathbf{v}_2$ are $c$-equivalent.

Consider a state $s = (q, \mathbf{v})$. We write $s + \delta$ instead of $(q, \mathbf{v} + \delta)$. A *timed transition* from $s$ has the form $s \xrightarrow{\delta} s + \delta$, where $\delta \in \mathsf{R}$. $s + \delta$ is called the *$\delta$-successor* of $s$. Given a jump $a = (q, Z, X, q')$ such that $\mathbf{v} \in Z$, a *discrete transition* with respect to $a$ has the form $s \xrightarrow{a} s'$, where $s' = (q', \mathbf{v}[X := 0])$. $s'$ is called the *$a$-successor* of $s$.

We write $s \xrightarrow{a} \xrightarrow{\delta} s'$ if, either $\delta = 0$ and $s \xrightarrow{a} s'$ is a discrete transition, or $\delta > 0$, and $s \xrightarrow{a} s''$ is a discrete transition and $s'' \xrightarrow{\delta} s'$ is a timed transition.

**Runs.** A *run* of $A$ starting from a discrete state $q_0$ and reaching a discrete state $q_f$ is a finite sequence $\rho = s_1 \xrightarrow{\delta_1} s_1 + \delta_1 \xrightarrow{a_1} s_2 \xrightarrow{\delta_2} s_2 + \delta_2 \xrightarrow{a_2} \cdots \xrightarrow{a_k} s_{k+1}$, such that $s_1 = (q_0, \_)$, $s_{k+1} = (q_f, \_)$, and for all $i = 1, 2, ..., k$, $s_i + \delta_i$ is the $\delta$-successor of $s_i$ and $s_{i+1}$ is the $a_i$-successor of $s_i + \delta_i$. The *time spent in* $\rho$, denoted $\mathsf{time}(\rho)$, is defined to be the sum $\delta_1 + \cdots + \delta_k$.

**Minimum-time reachability.** We say that $q_f$ is reachable from $q_0$ in time $T$ if there is a run $\rho$ starting from $q_0$ and reaching $q_f$, such that $\mathsf{time}(\rho) \leq T$. We define $MinTime(q_0, q_f)$ to be the minimum such time $T$, or $\infty$ if $q_f$ is not reachable from $q_0$.

# 3 Simulation Graph and Main Theorem

**The simulation graph.** Consider a TA $A$ with discrete states $Q$ and clocks $\mathcal{X}$. A *symbolic state* of $A$ is a set of states $S = \{(q, \mathbf{v}) \mid \mathbf{v} \in Z\}$, where $q \in Q$ and $Z$ is an $\mathcal{X}$-polyhedron. For simplicity, we denote $S$ as $(q, Z)$. If $S$ is convex, it is called a *zone*.

Given a zone $S = (q, Z)$, two edges $e_1 = (q, Z_1, X_1, q_1)$ and $e_2 = (q_2, Z_2, X_2, q)$ of $A$, and a natural constant $c \geq c_{max}(A)$, we define the following successor and predecessor operations:

$$\mathsf{post}(S, e_1) \stackrel{\text{def}}{=} \Big(q_1, \mathsf{close}(\nearrow((Z \cap Z_1)[X_1 := 0]), c)\Big)$$
$$\mathsf{pre}(S, e_2) \stackrel{\text{def}}{=} \Big(q_2, ([X_2 := 0](\swarrow Z)) \cap Z_2\Big)$$

Intuitively, $\mathsf{post}()$ contains all states (and their $c$-equivalents) that can be reached from some state in $(q, Z)$, by taking an $e$-transition, then letting some time pass; $\mathsf{pre}()$ contains all states that can reach some state in $(q, Z)$ by taking an $e$-transition, then letting some time pass. Since all operations preserve convexity of polyhedra, the result of $\mathsf{post}()$ and $\mathsf{pre}()$ is a zone.

Given an initial discrete state $q_0$, we generate the simulation graph using an on-the-fly depth-first search starting from $S_0 = (q_0, \mathsf{zero})$ and generating, for each node $S$ in the stack, the successors $S_i = \mathsf{post}(S, e_i)$ of $S$, for each each $e_i \in \mathsf{out}(S)$. We stop exploring the branch leading to $S_i$ if: either $S_i = \emptyset$; or there is a previously generated node $S' \supseteq S_i$. Otherwise, we add $S_i$ to the set of nodes and $S \xrightarrow{e_i} S_i$ to the set of edges of the simulation graph. By lemma 2, the simulation graph is finite.

**Relation of runs to paths in the simulation graph.** It has been shown [Tri98] that a run from

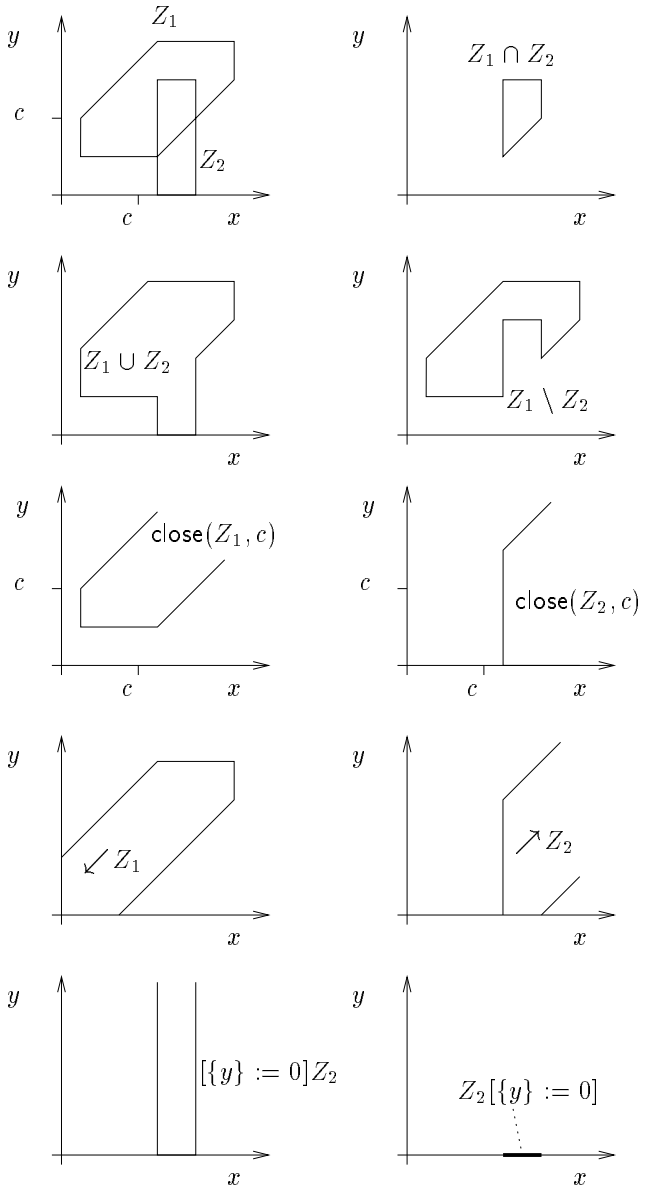


Figure 2: Polyhedra on $\{x, y\}$ and some of their operations.

$q_0$ to $q_f$ exists iff in the simulation graph starting from $S_0 = (q_0, \mathsf{zero})$, there is a node $S = (q_f, \_)$. Moreover, for each path $S_0 \xrightarrow{a_1} S_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} S_n$ in the simulation graph, there is a run $\rho = s_0 \xrightarrow{\delta_1}\xrightarrow{a_1} s_1 \cdots \xrightarrow{d_n} a_n s_n$, such that $s_i \in S_i$, for $i = 1, ..., n$, and vice versa. We say that $\rho$ *traverses* nodes $S_i$ of the simulation graph, and *passes through* $a_1 \cdots a_n$ for $i = 1, ..., n$. We also say that $\rho$ is *inscribed* in the path of the simulation graph.

**Jump sequences and graphs of timing constraints.** We first need some definitions and two lemmas which will be used below.

A *jump sequence* is a set of jumps $a_1 \cdots a_n$ such that the destination discrete state of $a_i$ is the source discrete state of $a_{i+1}$. For example, in the automaton of figure 1, $a_1 a_2 b_1 b_2$ is a jump sequence, but $a_1 b_1$ is not.

Given a jump sequence, we define the *graph of timing constraints over* $a_1 \cdots a_n$, to be the graph $G$ such that:

- $G$ has $n$ nodes, named $a_1, ..., a_n$.
- For all $1 \leq i < j \leq n$, $G$ has an edge $a_i \xrightarrow{0} a_j$.
- For all $1 \leq i < j \leq n$, if there is a clock $x$ such that the guard of $a_j$ is included in $x \leq \alpha$, and either $i = 1$ or $x$ is reset at $a_i$, then $G$ has an edge $a_j \xrightarrow{\alpha} a_i$.
- For all $1 \leq i < j \leq n$, if there is a clock $x$ such that the guard of $a_j$ is included in $x \geq \beta$, and either $i = 1$ or $x$ is reset at $a_i$, then $G$ has an edge $a_i \xrightarrow{-\beta} a_j$.

Intuitively, $G$ captures the set of all timing constraints on the occurrence of events $a_1, ..., a_n$, induced by the order of the sequence ($a_1$ must happen before $a_2$, therefore $a_1 \xrightarrow{0} a_2$) and by the clock resets and guards. For example, in the automaton of figure 1, since $x$ is reset at $a_1$ and $x \geq 2$ is the guard of $b_1$, $b_1$ must happen at least 2 time units after $a_1$, which is represented by a constraint $a_1 \xrightarrow{-2} b_1$.

The following lemma relates the existence of runs and time spent in them with the graphs of timing constraints over jump sequences.

**Lemma 4** *Consider a jump sequence $a_1 \cdots a_n$ and let $G$ be the graph of timing constraints over $a_1 \cdots a_n$. Let $\alpha$ be the cost of the shortest path from $a_1$ to $a_n$ and $\beta$ be the cost of the shortest path from $a_n$ to $a_1$ in $G$. There exists a run $\rho$ passing through $a_1 \cdots a_n$ such that $\mathsf{time}(\rho) = \delta$ iff $-\alpha \leq \delta \leq \beta$.*

We see, therefore, that there is a duality between runs and paths in the graph of timing constraints. Such a relation exists also between the constraints of polyhedra in the simulation graph and timing constraints in the above graphs.

**Lemma 5** *Consider a path in the simulation graph $S_0 \xrightarrow{a_1} \cdots \xrightarrow{a_n} S_n$, where $S_i = (q_i, Z_i)$, for $i = 0, ..., n$. Let $G$ be the graph of timing constraints over $a_1 \cdots a_n$. If there exist indices $0 \leq i, j, k \leq n$, $k \geq i, k \geq j$, and clocks $x \neq y$, $x, y \in \mathcal{X} \cup \{0\}$, such that:*

- *$x$ is reset in $a_i$, $y$ is reset in $a_j$ (that is, for the last time before $S_k$); and*
- *$Z_k \subseteq x - y \leq \alpha$, for some integer constant $\alpha$,*

*then there is a path in $G$ from $a_i$ to $a_j$ with total cost at most $\alpha$.*

In the above lemma, we assume that $x$ or $y$ can be either real clocks of a "fictitious clock" which always has the value 0 and by convention is taken to be reset on every discrete transition. For example, in the automaton of figure 1, $S_0 \xrightarrow{a_1} S_1 \xrightarrow{a_2} S_2 \xrightarrow{b_1} S_3$ is a path in the simulation graph, the polyhedron of $S_3$ is $x \geq 2 \wedge y \leq 1 \wedge y \leq x \wedge y = z$, and there are paths $a_1 \xrightarrow{-2} a_3$ (of total cost $-2$, corresponding to the constraint $x \geq 2 \equiv 0 - x \leq -2$), $a_1 \xrightarrow{0} a_2 \xrightarrow{1} a_3$ (of total cost 1, corresponding to the constraint $y \leq 1 \equiv y - 0 \leq 1$), and so on.

The proofs of the two lemmas are omitted.

**The main result.** We are now ready to state the main theorem.

**Theorem 1** *If a run from $q_0$ to $q_f$ exists, then there exists a run $\sigma$ from $q_0$ to $q_f$, such that $\sigma$ does not traverse the same node in the simulation graph twice, and $\mathsf{time}(\sigma) = MinTime(q_0, q_f)$, that is, $\sigma$ is a minimum-time run.*

**Proof** Let $\rho = s_0 \xrightarrow{\delta_1}\xrightarrow{a_1} \cdots \xrightarrow{\delta_n}\xrightarrow{a_n} s_n \xrightarrow{\epsilon_1}\xrightarrow{c_1} \cdots \xrightarrow{\epsilon_k}\xrightarrow{c_k} s'_k \xrightarrow{\zeta_1}\xrightarrow{b_1} \cdots \xrightarrow{\zeta_m}\xrightarrow{b_m} s_f$ be a minimum-time run from $q_0$ to $q_f$. Let $\rho$ be inscribed in the path $S_0 \xrightarrow{a_1} \cdots \xrightarrow{a_n} S_n \xrightarrow{c_1} \cdots \xrightarrow{c_k} S'_k \xrightarrow{b_1} \cdots \xrightarrow{b_m} S_f$ of the simulation graph. We will assume that $S_n = (q, Z)$, $S'_k = (q, Z')$ and $Z' \subseteq Z$, that is, $\rho$ traverses this node twice. We will show that there exists another run $\sigma$ which does not traverse a node twice, such that $\mathsf{time}(\rho) \leq \mathsf{time}(\sigma)$, therefore, $\sigma$ is also a minimum-time run.

Let $G'$ be the graph of timing constraints over $a_1 \cdots a_n c_1 \cdots c_k b_1 \cdots b_m$ and $\delta'$ be the cost of the shortest path from $a_1$ to $b_m$ in $G'$. Notice that $\delta' \leq 0$, since $a_1 \xrightarrow{0} b_m$ is an edge of $G'$. According to lemma 4, $-\delta' = \mathsf{time}(\rho)$.

Now, let $G$ be the graph of timing constraints over $a_1 \cdots a_n b_1 \cdots b_m$ and let $\delta$ be the cost of the shortest path from $a_1$ to $b_m$ in $G$. Again, $\delta \leq 0$. We will prove that $\delta' \leq \delta$. Therefore, according to lemma 4, there exists a run $\sigma$ passing through $a_1 \cdots a_n b_1 \cdots b_m$, such that $\mathsf{time}(\sigma) = -\delta \leq -\delta' = \mathsf{time}(\rho)$. Now, notice that $\sigma$ passes through $a_1 \cdots a_n b_1 \cdots b_m$, therefore, we have eliminated the loop $c_1 \cdots c_k$. If $\sigma$ traverses

the same node twice, it means that the jump sequence $a_1 \cdots a_n b_1 \cdots b_m$ contains another loop as well. But we can use the same technique as before to eliminate all loops. At the end, we get a run which does not traverse the same node twice.

It remains to show that $\delta' \leq \delta$. Let $\phi$ be the shortest path from $a_1$ to $b_m$ in $G$. $\phi$ has the following general form: $a_1 \overset{\gamma_1}{\rightsquigarrow}_a a^1 \overset{\alpha_1}{\rightarrow} b^1 \overset{\gamma_2}{\rightsquigarrow}_b b^2 \overset{\beta_2}{\rightarrow} a^2 \cdots a^l \overset{\alpha_l}{\rightarrow} b^l \overset{\gamma_p}{\rightsquigarrow}_b b_m$, where $a' \overset{\gamma}{\rightsquigarrow}_a a''$ denotes a sub-path that starts from $a$, ends in $a''$, passes only from nodes $a_i$ and has total cost $\gamma$ (if $a' = a''$ then $\gamma = 0$). Similarly for the notation $\overset{\gamma}{\rightsquigarrow}_b$.

Notice that we can safely assume that $\phi$ is acyclic. Indeed, there is no negative-cost cycle in $G$: otherwise the set of constraints induced by $G$ would be unsatisfiable, which would imply (by lemma 4) that there is no run passing through $a_1 \cdots a_n b_1 \cdots b_m$. Since there is no negative-cost cycle in $G$, there exists always an acyclic shortest path from $a_1$ to $b_m$: take any shortest path and remove any cycles that may exist (these cycles can only have cost 0).

We will construct a path $\phi'$ in $G'$ such that the cost of $\phi'$ is $\leq \delta$. Since the shortest path in $G'$ has cost $\delta'$, it must be that $\delta' \leq \delta$ (otherwise $\phi'$ would be shorter).

$\phi'$ will start with the part $a_1 \overset{\gamma_1}{\rightsquigarrow}_a a^1$ (since these are also edges in $G'$). We will continue extending $\phi'$, depending on the corresponding part of $\phi$.

If the edge $a^1 \overset{\alpha_1}{\rightarrow} b^1$ exists in $G'$, then we add it to $\phi'$ and continue extending it, otherwise, we have to find a replacement. If $a^1 \overset{\alpha_1}{\rightarrow} b^1$ does not exist in $G'$, the following things hold: first, $\alpha_1 < 0$ (the edge $a^1 \overset{0}{\rightarrow} b^1$ certainly exists in $G'$, since $a^1$ comes before $b^1$); second, there is a clock $x$ reset in $a^1$ and tested as $x \geq -\alpha_1$ in $b^1$ (this resulted in the edge $a^1 \overset{\alpha_1}{\rightarrow} b^1$ being added in $G$); third, $x$ is reset again in some $c_i$ (otherwise, $a^1 \overset{\alpha_1}{\rightarrow} b^1$ would also be in $G'$). Let $c^1$ be the last $c_i$ where $x$ is reset before $b^1$. Clearly, there is an edge $c^1 \overset{\alpha_1}{\rightarrow} b^1$ in $G'$. Moreover, since $c^1$ comes after $a^1$, there is an edge $a^1 \overset{0}{\rightarrow} c^1$ in $G'$. Combining the two edges, we extend $\phi'$ into: $a_1 \overset{\gamma_1}{\rightsquigarrow}_a a^1 \overset{0}{\rightarrow} c^1 \overset{\alpha_1}{\rightarrow} b^1$. Notice that the cost of $\phi'$ up to $b^1$ is the same as the cost of $\phi$ up to $b^1$.

We can continue extending $\phi'$ by adding the part $b^1 \overset{\gamma_2}{\rightsquigarrow}_b b^2$, since all edges in this part are also edges of $G'$.

If the edge $b^2 \overset{\beta_2}{\rightarrow} a^2$ exists in $G'$, we add it to $\phi'$ and continue extending it, otherwise, we have to find a replacement. If $b^2 \overset{\beta_2}{\rightarrow} a^2$ does not exist in $G'$, the following things hold: first, $\beta_2 \geq 0$ (otherwise, from the fact that $a^2$ comes before $b^2$, the set of constraints would be inconsistent); second, there is a clock $y$ reset in $a^2$ and tested as $y \leq \beta_2$ in $b^2$ (this generated the edge in $G$); third, $y$ is reset again in some $c_i$ (otherwise, $b^2 \overset{\beta_2}{\rightarrow} a^2$ would also be in $G'$). Let $c^2$ be the last $c_i$ where $y$ is reset before $b^2$. Clearly, there is an edge $b^2 \overset{\beta_2}{\rightarrow} c^2$ in $G'$. There is also an edge $a^2 \overset{0}{\rightarrow} c^2$, but this does not help us in filling the gap. We need something more.

Consider the part $a^2 \overset{\alpha_2}{\rightsquigarrow}_a a^3 \overset{\alpha_3}{\rightarrow} b^3$ of $\phi$ (this part certainly exists, since we have to reach $b_m$ eventually; $a^3$ might be the same as $a^2$ of course, or $b^3$ the same as $b_m$). Notice that $\alpha_3 \leq 0$, since $a^3 \overset{0}{\rightarrow} b^3$ is an edge of $G$, and $\phi$ is a shortest path. The sub-path $a^2 \overset{\alpha_2}{\rightsquigarrow}_a a^3$ induces the constraint $a^2 - a^3 \leq \alpha_2$. We distinguish two cases:

- $\alpha_3 = 0$ This implies that $\alpha_2 \leq 0$ (otherwise, $a^2 \overset{0}{\rightarrow} b^3$ would be shorter than $a^2 \overset{\alpha_2}{\rightsquigarrow}_a a^3 \overset{0}{\rightarrow} b^3$). Moreover, since $y$ is not reset until after $a_n$, the zone $Z$ (of node $S_n$) is included in the hyperplane $y \geq -\alpha_2$. Since $Z' \subseteq Z$, we also have $Z' \subseteq y \geq -\alpha_2$ (recall that $Z'$ is the zone of node $S'_k$).

  Now, $y$ is reset in $c^2$ for the last time in the loop $c_1 \cdots c_k$. This, together with the fact that $Z' \subseteq y \geq -\alpha_2$, and lemma 5, implies that there is a path $c^2 \overset{\gamma}{\rightsquigarrow}_c c_k$ with $\gamma \leq \alpha_2$. Therefore, we can extend $\phi'$ by adding the sub-path $b^2 \overset{\beta_2}{\rightarrow} c^2 \overset{\gamma}{\rightsquigarrow}_c c_k \overset{0}{\rightarrow} b^3$, which has total cost $\beta_2 + \gamma$, that is, less than or equal to the cost of the sub-path $b^2 \overset{\beta_2}{\rightarrow} a^2 \overset{\alpha_2}{\rightsquigarrow}_a a^3 \overset{0}{\rightarrow} b^2$ of $\phi$. Then we can continue extending $\phi'$ as usual.

- $\alpha_3 < 0$ In this case, let $z$ be the clock reset in $a^3$ and tested as $z \geq -\alpha_3$ in $b^3$ (which generated the edge $a^3 \overset{\alpha_3}{\rightarrow} b^3$). Since $y$ is not reset between $a^2$ and $b_1$ and $z$ is not reset between $a^3$ and $b_1$, this implies that in all zones after the latest of $a^2$ and $a^3$ until $b_1$, the clock difference $y - z$ satisfies the constraint $y - z \leq \alpha_2$. In particular, this is true for $Z$, that is, $Z \subseteq y - z \leq \alpha_2$. Since $Z' \subseteq Z$, we have $Z' \subseteq y - z \leq \alpha_2$.

  Now, let $d$ be the point where $z$ is reset for the last time before $b^3$, in the sequence $a_1 \cdots a_n c_1 \cdots c_k b_1 \cdots b_m$ ($d$ can be $a^3$ itself, or some point $c_i$). Recall that $y$ is reset in $c^2$. Since $Z' \subseteq y - z \leq \alpha_2$, by lemma 5, there must be a path $c^2 \overset{\gamma}{\rightsquigarrow} d$ in $G'$, such that $\gamma \leq \alpha_2$. Moreover, since $z$ is reset in $d$ for the last time before $b^3$, there must be an edge $d \overset{\alpha_3}{\rightarrow} b^3$ in $G'$ as well. Therefore, we can extend $\phi'$ by adding $b^2 \overset{\beta_2}{\rightarrow} c^2 \overset{\gamma}{\rightsquigarrow} d \overset{\alpha_3}{\rightarrow} b^3$. Then we can continue extending $\phi'$ as usual.

The construction of $\phi'$ continues in a similar manner (we have presented all possible cases already) until we arrive at $b_m$. The constructed path $\phi'$ has a total cost at most equal to the cost of $\phi$ (by construction). ■

# 4 Algorithms

In this section, we present the three algorithms for minimum-time reachability. The input is a timed automaton, an initial discrete state $q_0$ and a target state $q_f$. All three start by generating the simulation graph on the fly, say, using a depth-first search (DFS). This DFS stops exploring a node $S$ further if there exists an already visited node $S' \supseteq S$. If the target state is not hit, then it is unreachable, otherwise, it is reachable and an upper bound $T$ on the time to reach it is obtained. The details of the algorithms follow.

## 4.1 A Bellman-Ford Algorithm

First, the entire simulation graph is generated. During this process, the length $L$ of the longest acyclic path from the initial node to a target node is recorded.

Then, to each node $S$ in the graph, we associate a (generally non-convex) polyhedron $Z^+(S)$, on an extended space of clocks $\mathcal{X} \cup \{t\}$. Initially, $Z^+(S) = Z \wedge t = 0$ if $S$ is a target node $S = (q_f, Z)$, and $Z^+(S) = \emptyset$ otherwise. Then the backward iteration is performed as follows:

repeat at most $L$ times
  for each edge $S_1 \xrightarrow{a} S_2$ do
    $Z^+(S_1) := Z^+(S_1) \cup \mathsf{pre}(Z^+(S_2), a)$
until no polyhedron $Z^+(S)$ is updated.

In the above pre() operation, the extra clock $t$ counts backwards. After at most $L$ iterations the algorithm stops: at that point, for each node $S$, $Z^+(S)$ contains the set of points in $\mathcal{X} \cup \{t\}$ from which the target can be reached in at most $L$ steps. In particular, if $S_0$ is the initial node, $Z^+(S_0)/_{\{t\}}$ is an interval $[a, b]$ $(a, b \leq 0)$ such that the target can be reached within any $t \in [-b, -a]$. Therefore, $MinTime(q_0, q_f) = -b$. The fact that it suffices to look at paths of at most $L$ steps comes from Theorem 1.

**Complexity.** The algorithm performs at most $L$ iterations, where $L$ is bounded by the number of nodes in the simulation graph. At each iteration, there are at most $m$ updates, where $m$ is the number of edges of the simulation graph. Each update involves the computation of pre() and a union operation. In the case of convex polyhedra, pre() is computed in $O(k^3)$ time, where $k$ is the number of clocks. Inclusion can be checked in $O(k)$ time. In general, however, union will result in non-convex polyhedra, for which these operations become expensive. Compact methods for representing unions of polyhedra can be used [BLP+99].

## 4.2 Depth-First Search with Extra Clock

As previously, this algorithm starts by generating the simulation graph. As soon as a path $p$ reaching the target is found, a DFS with an extra clock $t$ (counting forward) starts. This DFS starts from the initial node (plus the constraint $t = 0$) along the path $p$ (the parts of the simulation graph explored previously need not be re-explored since they do not lead to the target state). After recomputing $p$ with the extra clock, a first upper bound $T$ on reaching the target can be obtained, by projecting the last polyhedron of the path to $\{t\}$, which gives an interval $[T, T']$. The search then goes on, with some modifications described next.

First, since an extra clock $t$ is added, the simulation graph might now be much greater than the original graph (computed on $\mathcal{X}$). Moreover, the search might not terminate since the clock $t$ is not upper bounded (the close() operation does not apply to $t$) and is never reset. To ensure termination and at the same time reduce the size of the graph to be explored, two additional criteria to stop exploring a newly created node $S$ are added.

1. If there exists a node $S'$ in the stack such that $S'/_{\mathcal{X}} \supseteq S/_{\mathcal{X}}$, then a cycle in the original graph has been found, and by Theorem 1, $S$ needs not be explored further.

2. Since an upper bound $T$ on the time to reach the target is already known, if $S/_{\{t\}} = [a, b]$ and $a \geq T$ then $S$ needs not be explored further.

Second, each time a new path $p'$ reaching the target is found, $p$ is updated to $p'$ and $T$ is updated to the minimum time to reach the target along $p'$ (this value is guaranteed to be less than the previous value of $T$, by the second stop criterion above). The value of $T$ at the end of the search is the minimum time to reach the target, $MinTime(q_0, q_f)$.

**Complexity.** Because of the extra clock, two nodes $S_1$ and $S_2$ which are "fused" in the original simulation graph (because $S_1 \subseteq S_2$) are distinguished in the extended search. Therefore, in the worst case, the algorithm will explore all paths of the original simulation graph leading to a target node (this case occurs when the times of these paths are $t_1 \leq t_2 \leq t_3 \leq \cdots$). The number of such paths can be exponential in the number of nodes in the simulation graph.

## 4.3 Binary Search on Time Horizon

This algorithm is essentially a modification of the previous one. After having obtained the first path $p$ and the upper bound $T$ as previously, this algorithm uses a binary search on $T$. The algorithm is as follows:

Set $[a, b] = [0, T]$.
repeat
  $T_{min} = \lfloor \frac{a+b}{2} \rfloor$.
  Start an extended DFS along $p$, with bound $t \geq T_{min}$.
  If a target node is hit along path $p'$,
    set $p = p'$, and
    set $[a, b] = [a, T_{min}]$.
  Otherwise (i.e., the extended DFS finishes
      and no target is hit),
    set $[a, b] = [T_{min}, b]$.
until $a = b$.

At the end, $MinTime(q_0, q_f) = a$.

**Complexity.** The algorithm will converge after approximately $\log(T)$ iterations. At each iteration, an extended DFS is performed. The advantage over the previous algorithm might be that a path is quickly found, therefore quickly decreasing $T$, in case the first estimate is too large. On the other hand, the previous algorithm will explore each path at most once, whereas this algorithm might explore a path multiple times before it converges.

# 5 Conclusion

We have proposed three algorithms for computing a minimal-time path leading from an initial to a final state in a timed automaton. Our basic result is that this can be done by checking acyclic paths in the simulation graph. All three algorithms have worst-case complexity which is worse than polynomial in the size of the simulation graph. It remains to be seen how well these algorithms perform in practice. An open question is whether minimum-time reachability can be solved in polynomial time in the size of the simulation graph.

In this paper we have focused on obtaining optimal solutions. In practice, a compromise might be made, using heuristics which are not guaranteed to yield optimal solutions, but work well most of the time. A proposal of a branch-and-bound algorithm for computing job-shop schedules modeled as timed automata is made in [Feh00].

# References


[ACD93] R. Alur, C. Courcoubetis, and D.L. Dill. Model checking in dense real time. *Information and Computation*, 104(1):2–34, 1993.

[AM99] E. Asarin and O. Maler. As soon as possible: Time optimal control for timed automata. In F. Vaandrager and J. van Schuppen, editors, *Hybrid Systems: Computation and Control*, volume 1569 of *LNCS*, pages 19–30. Springer, Mars 1999.

[BLP+99] G. Behrmann, K. Larsen, J. Pearson, C. Weise, and W. Yi. Efficient timed reachability analysis using clock difference diagrams. In *CAV'99*, 1999.

[Dil89] D.L. Dill. Timing assumptions and verification of finite-state concurrent systems. In J. Sifakis, editor, *Automatic Verification Methods for Finite State Systems*, Lecture Notes in Computer Science 407, pages 197–212. Springer–Verlag, 1989.

[Feh00] A. Fehnker. Bounding and heuristics in forward reachability algorithms. Technical report, CSI report CSI-R0002, 2000.

[HNSY94] T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2):193–244, 1994.

[NY00] P. Niebert and S. Yovine. Computing optimal operation schemes for chemical plants in multi-batch mode. In *Proc. Hybrid Systems, Computation and Control, HSCC'00*, LNCS, Pittsburgh, PA, USA, March 2000. Springer-Verlag.

[Tri98] S. Tripakis. *The formal analysis of timed systems in practice*. PhD thesis, Université Joseph Fourrier de Grenoble, 1998.

[Tri99] S. Tripakis. Timed diagnostics for reachability properties. In *TACAS'99*, 1999.