Minimal-Time Synthesis for Parametric Timed Automata

Étienne André[1], Vincent Bloemen[2], Laure Petrucci[1], and Jaco van de Pol[2,3]

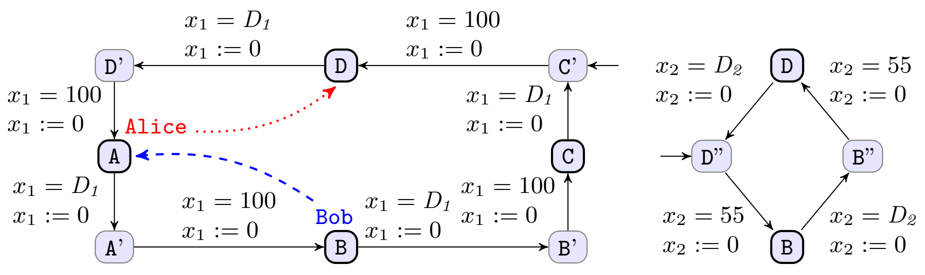
[1] LIPN, CNRS UMR 7030, Université Paris 13, Villetaneuse, France
[2] University of Twente, The Netherlands
[3] University of Aarhus, Denmark

Abstract. Parametric timed automata (PTA) extend timed automata by allowing parameters in clock constraints. Such a formalism is for instance useful when reasoning about unknown delays in a timed system. Using existing techniques, a user can synthesize the parameter constraints that allow the system to reach a specified goal location, regardless of how much time has passed for the internal clocks.
We focus on synthesizing parameters such that not only the goal location is reached, but we also address the following questions: *what is the minimal time to reach the goal location?* and *for which parameter values can we achieve this?* We analyse the problem and present an algorithm that solves it. We also discuss and provide solutions for minimizing a specific parameter value to still reach the goal.
We empirically study the performance of these algorithms on a benchmark set for PTAs and show that *minimal-time reachability synthesis* is more efficient to compute than the standard synthesis algorithm for reachability.

1 Introduction

Timed Automata (TA) [2] extend finite automata with *clocks*, for instance to model real-time systems. Timed automata allow for reasoning about temporal properties of the designed system. In addition to reachability problems, it is possible to compute for TAs the minimal or maximal time required to reach a specific goal location. Such a result is valuable in practice, as it can describe the response time of a system or it may indicate when a component failure occurs.

It may not always be possible to describe a real-time system with a TA. There are often uncertainties in the timing constraints, for instance how long it takes between sending and receiving a message. Optimising specific timing delays to improve the overall throughput of the system may also be considered, as shown in Example 1. Such uncertainties can however be modelled using a *parametric timed automaton (PTA)* [3]. A PTA adds parameters, or unknown constants, to the TA formalism. By examining the reachability of a goal location, the parameters get constrained and we can observe which parameter valuations preserve the reachability of the goal location.



(a) Train 1 (b) Train 2

Fig. 1: Train delay scheduling problem: `Alice` (depicted in dotted red), located at `A`, wants to go to location `D`. `Bob` (depicted in dashed blue), located at `B`, wants to go to `A`. By setting the train delays D_1 and D_2 for train 1 and 2, make sure that both `Alice` and `Bob` reach their target location in minimum total time.

This process, also called *parameter synthesis*, is definitely useful for analysing reachability properties of a system. However, this technique does disregard timing aspects to some extent. Given the parameter constraints, it is no longer possible to give clear boundaries on the time to reach the goal, as this may depend on the parameter valuations. We focus on the parameter synthesis problem while reaching the goal location in minimal time, as demonstrated in Example 1.

Example 1. Consider the example in Fig. 1, which depicts a train network consisting of two trains. Both trains share locations `B` and `D` (the stations platforms) while locations `A`$'$, `B`$'$, `C`$'$, `D`$'$, `B`$''$, and `D`$''$ represent a train travelling (tracks). The travel time for train 1 between any two stations is 100, and 55 for train 2. Train 1 stops at stations `A`, `B`, `C`, and `D`, for time D_1 (and train 2 stops for D_2 time units at `B` and `D`). Here, the train delays D_1 and D_2 are parameters and x_1 and x_2 are clocks. Both clocks start at 0 and reset after every transition. We assume that the trains use different tracks and changing trains at the platform of a station can be done in negligible time.

`Alice` is starting her journey from `A` and would like to go to `D`. `Bob` is located at `B` and wants to go to `A`. Train 1 and/or 2 can be used to travel, if both the train and the person are at the same location. Initially, both `Alice` and `Bob` wait for a train, since the initial positions of train 1 and 2 are respectively `C`' and `D`".

We would like to set the train delays D_1 and D_2 in such a way that the total time for `Alice` and `Bob` to reach their target location is minimal. The optimal solution is $D_1 = 25 \wedge D_2 = 15$, which leads to a total time of 405 units[4]. Note that this is neither optimal for `Alice` (the fastest would be $D_1 = 0 \wedge D_2 = 5$), nor optimal for `Bob` ($D_1 = 10 \wedge D_2 = 0$).

[4] `Alice` waits for train 1 to reach `A` at time 225, then she hops on and exits the train on time 350 at `B`. There she can immediately take train 2 and reach `D` at time 405. `Bob` waits for train 2 to reach `B` at time 55 and takes this train. At time 125 he reaches `D` and can immediately hop on train 1. Bob reaches `A` at time 225.

Note that in other instances, the time to reach a goal location may be an interval, describing the lower- and upper-bound on the time. This can be achieved in the example by changing the travel time from train 1 to be between 95 and 105, by guarding the outgoing transitions from locations $\mathtt{A}'$, $\mathtt{B}'$, $\mathtt{C}'$ and $\mathtt{D}'$ with $95 \leq x_1 \leq 105$ (instead of $x_1 = 100$). We focus on the lower-bound *global time*, meaning that we look at the minimal *total* time passed in the system, which may differ from the clock values as the clocks can be reset.

In this paper we address the following problems:

- *minimal-time reachability*: synthesizing *a single* parameter valuation for which the goal location can be reached in minimal (lower-bound) time,
- *minimal-time reachability synthesis*: synthesizing all parameter valuations such that the time to reach the goal location is minimized, and
- *parameter minimization synthesis*: synthesizing all parameter valuations such that a particular parameter is minimized and the goal location can still be reached (this problem can also address the *minimal-time reachability synthesis problem* by adding a parameter to equal with the final clock value).

For all stated problems we provide algorithms to solve them and empirically compare them with a set of benchmark experiments for PTAs, obtained from [5]. Interestingly, compared to standard reachability and synthesis, minimal-time reachability and synthesis is in general computed faster as fewer states have to be considered in the exploration. We also look at the computability and intractability of the problems for PTAs and L/U-PTAs (PTAs for which each parameter only appears as a lower- or upper-bound).

Related work. The earliest work on minimal-time reachability was by Courcoubetis and Yannakis [17], who first addressed the problem of computing lower and upper bounds in timed automata. Several algorithms have been developed since to improve performance [22,24,25], by e. g. using parallelism. Related problems have been studied, such as minimal-time reachability for weighted timed automata [4], minimal-cost reachability in priced timed automata [12], and job scheduling for timed automata [1].

Concerning parametric timed automata, to the best of our knowledge, the minimal-time reachability problem was not tackled in the past. The reachability-emptiness problem ("the emptiness of the parameter valuation set for which a given set of locations is reachable") is undecidable [3], with various settings considered, notably a single clock compared to parameters [21] or a single rational-valued or integer-valued parameter [21,14] (see [6] for a survey). Only severely limiting the number of clocks (e. g. [3,16,14,11]), and often restricting to integer-valued parameters, can bring some decidability. Emptiness for the subclass of L/U-PTAs is also decidable [13]. Minimizing a parameter can however be considered done in the setting of upper-bound PTAs (PTAs in which the clocks are only restricted from above): the exact synthesis of integer valuations for which a location is reachable can be done [15], and therefore the minimum valuation of a parameter can be obtained.

2 Preliminaries

We assume a set $\mathbb{X} = \{x_1, \dots, x_{|\mathbb{X}|}\}$ of *clocks*, i. e. real-valued variables that evolve at the same rate. A clock valuation is $\nu_\mathbb{X} : \mathbb{X} \to \mathbb{R}_{\geq 0}$. We write $\mathbf{0}$ for the clock valuation assigning 0 to all clocks. Given $d \in \mathbb{R}_{\geq 0}$, $\nu_\mathbb{X} + d$ is the valuation s.t. $(\nu_\mathbb{X} + d)(x) = \nu_\mathbb{X}(x) + d$, for all $x \in \mathbb{X}$. Given $R \subseteq \mathbb{X}$, we define the *reset* of a valuation $\nu_\mathbb{X}$, denoted by $[\nu_\mathbb{X}]_R$, as follows: $[\nu_\mathbb{X}]_R(x) = 0$ if $x \in R$, and $[\nu_\mathbb{X}]_R(x) = \nu_\mathbb{X}(x)$ otherwise.

We assume a set $\mathbb{P} = \{p_1, \dots, p_{|\mathbb{P}|}\}$ of *parameters*. A parameter *valuation* $\nu_\mathbb{P}$ is $\nu_\mathbb{P} : \mathbb{P} \to \mathbb{Q}_+$. We denote $\bowtie \in \{<, \leq, =, \geq, >\}$, $\triangleleft \in \{<, \leq\}$, and $\triangleright \in \{>, \geq\}$. A guard g is a constraint over $\mathbb{X} \cup \mathbb{P}$ defined by a conjunction of inequalities of the form $x \bowtie d$ or $x \bowtie p$, with $x \in \mathbb{X}$, $d \in \mathbb{N}$ and $p \in \mathbb{P}$. Given a guard g, we write $\nu_\mathbb{X} \models \nu_\mathbb{P}(g)$ if the expression obtained by replacing each $x \in \mathbb{X} \cap g$ by $\nu_\mathbb{X}(x)$ and each $p \in \mathbb{P} \cap g$ by $\nu_\mathbb{P}(p)$ evaluates to true.

2.1 Parametric timed automata

Definition 1 (PTA). *A PTA $\mathcal{A}$ is a tuple $\mathcal{A} = (\Sigma, L, \ell_0, \mathbb{X}, \mathbb{P}, \mathcal{I}, E)$, where: i) Σ is a finite set of actions, ii) L is a finite set of locations, iii) $\ell_0 \in L$ is the initial location, iv) $\mathbb{X}$ is a finite set of clocks, v) $\mathbb{P}$ is a finite set of parameters, vi) $\mathcal{I}$ is the invariant, assigning to every $\ell \in L$ a guard $\mathcal{I}(\ell)$, vii) E is a finite set of edges $e = (\ell, g, a, R, \ell')$ where $\ell, \ell' \in L$ are the source and target locations, $a \in \Sigma$, $R \subseteq \mathbb{X}$ is a set of clocks to be reset, and g is a guard.*

Given a parameter valuation $\nu_\mathbb{P}$ and PTA $\mathcal{A}$, we denote by $\nu_\mathbb{P}(\mathcal{A})$ the non-parametric structure where all occurrences of a parameter $p \in \mathbb{P}$ have been replaced by $\nu_\mathbb{P}(p)$. Any structure $\nu_\mathbb{P}(\mathcal{A})$ is also a *timed automaton*. By assuming a rescaling of the constants (multiplying all constants in $\nu_\mathbb{P}(\mathcal{A})$ by their least common denominator), we obtain an equivalent (integer-valued) TA.

Definition 2 (L/U-PTA). *An* L/U-PTA *is a PTA where the set of parameters is partitioned into lower-bound parameters and upper-bound parameters, i. e. parameters that appear in guards and invariants in inequalities of the form $p \triangleleft x$, and of the form $p \triangleright x$ respectively.*

Concrete semantics of TAs. Let us now recall the concrete semantics of TA.

Definition 3 (Semantics of a TA). *Given a PTA $\mathcal{A} = (\Sigma, L, \ell_0, \mathbb{X}, \mathbb{P}, \mathcal{I}, E)$, and a parameter valuation $\nu_\mathbb{P}$, the semantics of $\nu_\mathbb{P}(\mathcal{A})$ is given by the timed transition system (TTS) $(S, s_0, \to)$, with:*

- $S = \{(\ell, \nu_\mathbb{X}) \in L \times \mathbb{R}_{\geq 0}^{|\mathbb{X}|} \mid \nu_\mathbb{X} \models \nu_\mathbb{P}(\mathcal{I}(\ell))\}$, $s_0 = (\ell_0, \mathbf{0})$,
- *$\to$ consists of the discrete and (continuous) delay transition relations: i) discrete transitions: $(\ell, \nu_\mathbb{X}) \overset{e}{\mapsto} (\ell', \nu'_\mathbb{X})$, if $(\ell, \nu_\mathbb{X}), (\ell', \nu'_\mathbb{X}) \in S$, and there exists $e = (\ell, g, a, R, \ell') \in E$, such that $\nu'_\mathbb{X} = [\nu_\mathbb{X}]_R$, and $\nu_\mathbb{X} \models \nu_\mathbb{P}(g)$, ii) delay transitions: $(\ell, \nu_\mathbb{X}) \overset{d}{\mapsto} (\ell, \nu_\mathbb{X} + d)$, with $d \in \mathbb{R}_{\geq 0}$, if $\forall d' \in [0, d], (\ell, \nu_\mathbb{X} + d') \in S$.*

Moreover we write $(\ell, \nu_{\mathbb{X}}) \overset{(d,e)}{\longrightarrow} (\ell', \nu'_{\mathbb{X}})$ for a combination of a delay and discrete transition if $\exists \nu''_{\mathbb{X}} : (\ell, \nu_{\mathbb{X}}) \overset{d}{\mapsto} (\ell, \nu''_{\mathbb{X}}) \overset{e}{\mapsto} (\ell', \nu'_{\mathbb{X}})$.

Given a TA $\nu_{\mathbb{P}}(\mathcal{A})$ with concrete semantics $(S, s_0, \rightarrow)$, we refer to the states of S as the *concrete states* of $\nu_{\mathbb{P}}(\mathcal{A})$. A *run* ρ of $\nu_{\mathbb{P}}(\mathcal{A})$ is a possibly infinite alternating sequence of concrete states of $\nu_{\mathbb{P}}(\mathcal{A})$, and pairs of edges and delays, starting from the initial state s_0 of the form $s_0, (d_0, e_0), s_1, \cdots$, with $i = 0, 1, \dots$, and $d_i \in \mathbb{R}_{\geq 0}$, $e_i \in E$, and $(s_i, e_i, s_{i+1}) \in \rightarrow$. The set of all finite runs over $\nu_{\mathbb{P}}(\mathcal{A})$ is denoted by $\mathit{Runs}(\nu_{\mathbb{P}}(\mathcal{A}))$. The *duration* of a finite run $\rho = s_0, (d_0, e_0), s_1, \cdots, s_i$, is given by $\mathit{duration}(\rho) = \sum_{0 \leq j \leq i-1} d_j$.

Given a state $s = (\ell, \nu_{\mathbb{X}})$, we say that s is reachable in $\nu_{\mathbb{P}}(\mathcal{A})$ if s is the last state of a run of $\nu_{\mathbb{P}}(\mathcal{A})$. By extension, we say that ℓ is reachable; and by extension again, given a set T of locations, we say that T is reachable if there exists $\ell \in T$ such that ℓ is reachable in $\nu_{\mathbb{P}}(\mathcal{A})$. The set of all finite runs of $\nu_{\mathbb{P}}(\mathcal{A})$ that reach T is denoted by $\mathit{Reach}(\nu_{\mathbb{P}}(\mathcal{A}), T)$.

Minimal reachability. As the minimal time may not be an integer, but also the smallest value larger than an integer[5], we define a minimum as either a pair in $\mathbb{Q}_+ \times \{=, >\}$ or ∞. The comparison operators function as follows: $(c, =) < \infty$, $(c, >) < \infty$, and $(c_1, \succ_1) < (c_2, \succ_2)$ iff either $c_1 < c_2$ or $c_1 = c_2$, $\succ_1$ is $=$ and $\succ_2$ is $>$.

Given a set of locations T, the minimal time reachability of T in $\nu_{\mathbb{P}}(\mathcal{A})$, denoted by $\mathit{MinTimeReach}(\nu_{\mathbb{P}}(\mathcal{A}), T) = \min\{\mathit{duration}(\rho) \mid \forall \rho \in \mathit{Reach}(\nu_{\mathbb{P}}(\mathcal{A}), T)\}$, is the minimal duration over all runs of $\nu_{\mathbb{P}}(\mathcal{A})$ reaching T.

By extension, given a PTA, we denote by $\mathit{MinTimePTA}(\mathcal{A}, T)$ the minimal time reachability of T over all valuations, i.e. $\mathit{MinTimePTA}(\mathcal{A}, T) = \min_{\nu_{\mathbb{P}}} \mathit{MinTimeReach}(\nu_{\mathbb{P}}(\mathcal{A}), T)$. As we will be interested in synthesizing the valuations leading to the minimal time, let us define $\mathit{MinTimeSynth}(\mathcal{A}, T) = \{\nu_{\mathbb{P}} \mid \mathit{MinTimeReach}(\nu_{\mathbb{P}}(\mathcal{A}), T) = \mathit{MinTimePTA}(\mathcal{A}, T)\}$.

We will also be interested in minimizing the valuation of a given parameter p_i (without any notion of time) reaching a given location, and we therefore define $\mathit{MinParamReach}(\mathcal{A}, p_i, T) = \min_{\nu_{\mathbb{P}}}\{\nu_{\mathbb{P}}(p_i) \mid \mathit{Reach}(\nu_{\mathbb{P}}(\mathcal{A}), T) \neq \emptyset\}$. Similarly, we will be interested in synthesizing *all* valuations leading to the minimal valuation of $\nu_{\mathbb{P}}$ reaching T, so let us define $\mathit{MinParamSynth}(\mathcal{A}, p_i, T) = \{\nu_{\mathbb{P}} \mid \mathit{Reach}(\nu_{\mathbb{P}}(\mathcal{A}), T) \neq \emptyset \wedge \nu_{\mathbb{P}}(p_i) = \mathit{MinParamReach}(\mathcal{A}, p_i, T)\}$.

2.2 Computation problems

Minimal-time reachability problem:
INPUT: A PTA $\mathcal{A}$, a subset $T \subseteq L$ of its locations.
PROBLEM: Compute $\mathit{MinTimePTA}(\mathcal{A}, T)$.

Minimal-time reachability synthesis problem:
INPUT: A PTA $\mathcal{A}$, a subset $T \subseteq L$ of its locations.
PROBLEM: Compute $\mathit{MinTimeSynth}(\mathcal{A}, T)$.

[5] Consider a TA with a transition guarded by $x > 1$ from ℓ_0 to ℓ_1, then the minimal duration of runs reaching ℓ_1 is not 1 but slightly more.

Before addressing the problems defined in Section 2.2, we will address the slightly different problem of minimal-parameter reachability, i. e. the minimization of a parameter reaching a given location (independently of time). We will see in Lemma 1 that this problem can also give an answer to the minimal-time reachability (synthesis) problem.

Minimal-parameter reachability problem:
INPUT: A PTA $\mathcal{A}$, a parameter p, a subset $T \subseteq L$ of the locations of $\mathcal{A}$.
PROBLEM: Compute $\mathit{MinParamReach}(\mathcal{A}, T, p)$.

Minimal-parameter reachability synthesis problem:
INPUT: A PTA $\mathcal{A}$, a parameter p, a subset $T \subseteq L$ of the locations of $\mathcal{A}$.
PROBLEM: Synthesize $\mathit{MinParamSynth}(\mathcal{A}, T, p)$.

2.3 Symbolic semantics

Let us now recall the symbolic semantics of PTAs (see e. g. [19,8]), that we will use to solve these problems.

Constraints We first define operations on constraints. A linear term over $\mathbb{X} \cup \mathbb{P}$ is of the form $\sum_{1 \leq i \leq |\mathbb{X}|} \alpha_i x_i + \sum_{1 \leq j \leq |\mathbb{P}|} \beta_j p_j + d$, with $x_i \in \mathbb{X}$, $p_j \in \mathbb{P}$, and $\alpha_i, \beta_j, d \in \mathbb{Z}$. A *constraint* C (i. e. a convex polyhedron) over $\mathbb{X} \cup \mathbb{P}$ is a conjunction of inequalities of the form $lt \bowtie 0$, where lt is a linear term. $\bot$ denotes the false parameter constraint, i. e. the constraint over $\mathbb{P}$ containing no valuation.

Given a parameter valuation $\nu_{\mathbb{P}}$, $\nu_{\mathbb{P}}(C)$ denotes the constraint over $\mathbb{X}$ obtained by replacing each parameter p in C with $\nu_{\mathbb{P}}(p)$. Likewise, given a clock valuation $\nu_{\mathbb{X}}$, $\nu_{\mathbb{X}}(\nu_{\mathbb{P}}(C))$ denotes the expression obtained by replacing each clock x in $\nu_{\mathbb{P}}(C)$ with $\nu_{\mathbb{X}}(x)$. We say that $\nu_{\mathbb{P}}$ *satisfies* C, denoted by $\nu_{\mathbb{P}} \models C$, if the set of clock valuations satisfying $\nu_{\mathbb{P}}(C)$ is non-empty. Given a parameter valuation $\nu_{\mathbb{P}}$ and a clock valuation $\nu_{\mathbb{X}}$, we denote by $\nu_{\mathbb{X}}|\nu_{\mathbb{P}}$ the valuation over $\mathbb{X} \cup \mathbb{P}$ such that for all clocks x, $\nu_{\mathbb{X}}|\nu_{\mathbb{P}}(x) = \nu_{\mathbb{X}}(x)$ and for all parameters p, $\nu_{\mathbb{X}}|\nu_{\mathbb{P}}(p) = \nu_{\mathbb{P}}(p)$. We use the notation $\nu_{\mathbb{X}}|\nu_{\mathbb{P}} \models C$ to indicate that $\nu_{\mathbb{X}}(\nu_{\mathbb{P}}(C))$ evaluates to true. We say that C is *satisfiable* if $\exists \nu_{\mathbb{X}}, \nu_{\mathbb{P}}$ s.t. $\nu_{\mathbb{X}}|\nu_{\mathbb{P}} \models C$.

We define the *time elapsing* of C, denoted by $C^{\nearrow}$, as the constraint over $\mathbb{X}$ and $\mathbb{P}$ obtained from C by delaying all clocks by an arbitrary amount of time. That is, $\nu'_{\mathbb{X}}|\nu_{\mathbb{P}} \models C^{\nearrow}$ iff $\exists \nu_{\mathbb{X}} : \mathbb{X} \to \mathbb{R}_+, \exists d \in \mathbb{R}_+$ s.t. $\nu'_{\mathbb{X}}|\nu_{\mathbb{P}} \models C \wedge \nu'_{\mathbb{X}} = \nu_{\mathbb{X}} + d$. Given $R \subseteq \mathbb{X}$, we define the *reset* of C, denoted by $[C]_R$, as the constraint obtained from C by resetting the clocks in R, and keeping the other clocks unchanged. Given a subset $\mathbb{P}' \subseteq \mathbb{P}$ of parameters, we denote by $C{\downarrow_{\mathbb{P}'}}$ the projection of C onto $\mathbb{P}'$, i. e. obtained by eliminating the clock variables and the parameters in $\mathbb{P} \setminus \mathbb{P}'$ (e. g. using Fourier-Motzkin). Therefore, $C{\downarrow_{\mathbb{P}}}$ denotes the elimination of the clock variables only, i. e. the projection onto $\mathbb{P}$. Given p, we denote by $\mathsf{GetMin}(C, p)$ the minimum of p in a form $(c, \succ)$. Technically, GetMin can be implemented using polyhedral operations as follows: $C{\downarrow_{\{p\}}}$ is computed, and then the infimum is extracted; then the operator in $\{=, >\}$ is inferred depending whether $C{\downarrow_{\{p\}}}$ is bounded from below using a closed or an open constraint. We extend GetMin to accommodate clocks, thus $\mathsf{GetMin}(C, x)$ returns the minimal clock value that x can take, while conforming to C.

A symbolic state is a pair (ℓ, C) where $\ell \in L$ is a location, and C its associated parametric zone.

Definition 4 (Symbolic semantics). *Given a PTA $\mathcal{A} = (\Sigma, L, \ell_0, \mathbb{X}, \mathbb{P}, \mathcal{I}, E)$, the symbolic semantics of $\mathcal{A}$ is defined by the labelled transition system called the* parametric zone graph $\mathcal{PZG} = (E, \mathbf{S}, \mathbf{s}_0, \Rightarrow)$*, with*

- $\mathbf{S} = \{(\ell, C) \mid C \subseteq \mathcal{I}(\ell)\}$*,* $\mathbf{s}_0 = \big(\ell_0, (\bigwedge_{1 \leq i \leq |\mathbb{X}|} x_i = 0)^{\nearrow} \wedge \mathcal{I}(\ell_0)\big)$*, and*
- $\big((\ell, C), e, (\ell', C')\big) \in \Rightarrow$ *if* $e = (\ell, g, a, R, \ell')$ *and* $C' = \big([(C \wedge g)]_R \wedge \mathcal{I}(\ell')\big)^{\nearrow} \wedge \mathcal{I}(\ell')$ *with* C' *satisfiable.*

That is, in the parametric zone graph, nodes are symbolic states, and arcs are labeled by *edges* of the original PTA. If $\big((\ell, C), e, (\ell', C')\big) \in \Rightarrow$, we write $\mathsf{Succ}(\mathbf{s}, e) = (\ell', C')$. By extension, we write $\mathsf{Succ}(\mathbf{s})$ for $\cup_{e \in E} \mathsf{Succ}(\mathbf{s}, e)$.

3 Computability and intractability

3.1 Minimal-time reachability

The following result is a consequence of a monotonicity property of L/U-PTAs [19]. We can safely replace parameters with some constants in order to compute the solution to the minimal-time reachability problem, which reduces to the minimal-time reachability in a TA, which is PSPACE-complete [17].

Proposition 1 (minimal-time reachability for L/U-PTAs). *The minimal-time reachability problem for L/U-PTAs is PSPACE-complete.*

Computing the minimal time for which a location is reached (Proposition 1) does not mean that we are able to compute exactly all valuations for which this location is reachable in minimal time. In fact, we show that it is not possible in a formalism for which the emptiness of the intersection is decidable—which notably rules out its representation as a finite union of polyhedra.

Proposition 2 (intractability of minimal-time reachability synthesis for L/U-PTAs). *The solution to the minimal-time reachability synthesis problem for L/U-PTAs cannot be represented in a formalism for which the emptiness of the intersection is decidable.*

3.2 Minimal-parameter reachability

For the full class of PTAs, we will see that these problems are clearly out of reach: if it was possible to compute the solution to the minimal-parameter reachability or minimal-parameter reachability synthesis, then it would be possible to answer the reachability emptiness problem—which is undecidable in most settings [6].

We first show that an algorithm for the minimal-parameter synthesis problem can be used to solve the minimal-time synthesis problem, i. e. the minimal-parameter synthesis problem is harder than the minimal-time synthesis problem.

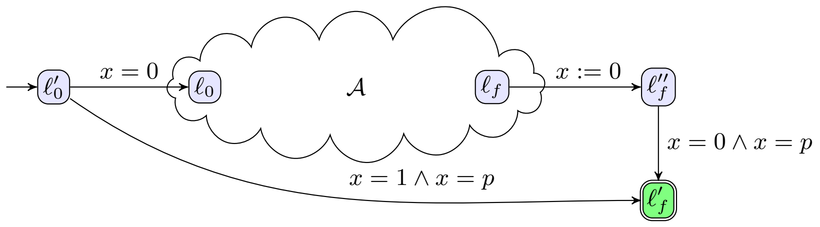


Fig. 2: Intractability of minimal-parameter reachability for PTAs

Lemma 1 (minimal-time from minimal-parameter synthesis). *An algorithm that solves the minimal-parameter synthesis problem can be used to solve the minimal-time synthesis problem by extending the PTA.*

Proof. Assume we are given an arbitrary PTA $\mathcal{A}$, a set of target locations T, and a global clock x_{global} that never resets. We construct the PTA $\mathcal{A}'$ from $\mathcal{A}$ by adding a new parameter p_{global}, and for every edge (ℓ, g, a, R, ℓ') in $\mathcal{A}'$ such that $\ell' \in T$, we replace g by $g \wedge x_{global} = p_{global}$. Note that when a target location from T is reached, we have that $x_{global} = p_{global}$, hence by minimizing p_{global} we also minimize x_{global}. Thus, by solving $\mathit{MinParamSynth}(\mathcal{A}', T, p_{global})$, we effectively solve $\mathit{MinTimeSynth}(\mathcal{A}, T)$.

The following result states that synthesis of the minimal-value of the parameter is intractable for PTAs.

Proposition 3 (intractability of minimal-parameter reachability for PTAs). *The solution to the minimal-parameter reachability for PTAs cannot be computed in general.*

Proof (sketch). By showing that testing equality of "$p = 0$" against the solution of the minimal-parameter reachability problem for the PTA in Fig. 2 and ℓ'_f is equivalent to solving reachability emptiness of ℓ_f in $\mathcal{A}$—which is undecidable [3]. Details and other proofs are given in [7].

The intractability of minimal-parameter reachability synthesis for PTAs will be implied by the upcoming Proposition 4 in a more restricted setting.

Intractability of the synthesis for L/U-PTAs. The following result states that synthesis is intractable for L/U-PTAs. In particular, this rules out the possibility to represent the result using a finite union of polyhedra.

Proposition 4 (intractability of minimal-parameter reachability synthesis for L/U-PTAs). *The solution to the minimal-parameter reachability synthesis for L/U-PTAs cannot always be represented in a formalism for which the emptiness of the intersection is decidable and for which the minimization of a variable is computable.*

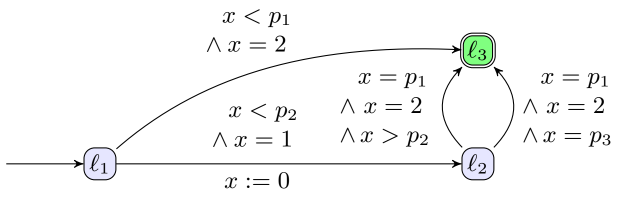


Fig. 3: PTA exemplifying Algorithm 1.

Proof. From Lemma 1 and Proposition 2. □

The minimal-parameter reachability problem remains open for L/U-PTAs (see Section 7). Despite these negative results, we will define procedures that address not only the class of L/U-PTAs, but in fact the class of full PTAs. Of course, these procedures are not guaranteed to terminate.

4 Minimal parameter reachability synthesis

We give $\mathsf{MinParamSynth}(\mathcal{A}, T, p)$ in Algorithm 1. It maintains a set $\mathbf{W}$ of waiting symbolic states, a set $\mathbf{P}$ of passed states, a current optimum *Opt* and the associated optimal valuations K. While $\mathbf{W}$ is not empty, a state is picked in line 6. If it is a target state (i. e. $\ell \in T$) then the projection of its constraint onto p is computed, and the minimum is inferred (line 10). If that projection improves the known optimum, then the associated parameter valuations K are completely replaced by the one obtained from the current state (i. e. the projection of C onto $\mathbb{P}$). Otherwise, if $C\downarrow_{\{p\}}$ is equal to the known optimum (line 14), then we add (using disjunction) the associated valuations. Finally, if the current state is not a target state and has not been visited before, then we compute its successors and add them to $\mathbf{W}$ in lines 17 and 18.

Note that if $\mathbf{W}$ is implemented as a FIFO list with "pick" the first element, then this algorithm is a classical BFS procedure.

Also note that if we replace lines 10-15 with the statement $K \leftarrow K \lor C\downarrow_{\mathbb{P}}$ (i. e. adding the parameter valuations to K every time the algorithm reaches a target location), we obtain the standard synthesis algorithm $\mathsf{EFSynth}$ from e. g. [20].

Example 2. Consider the PTA $\mathcal{A}$ in Fig. 3, and run $\mathsf{MinParamSynth}(\mathcal{A}, \{\ell_3\}, p_1)$. The initial state is $\mathbf{s}_1 = (\ell_1, x \geq 0)$ (we omit the trivial constraints $p_i \geq 0$). Its successors $\mathbf{s}_2 = (\ell_3, x \geq 2 \land p_1 > 2)$ and $\mathbf{s}_3 = (\ell_2, x \geq 0 \land p_2 > 1)$ are added to $\mathbf{W}$. Pick $\mathbf{s}_2$ from $\mathbf{W}$: it is a target, and therefore $\mathsf{GetMin}(C_2, p_1)$ is computed, which gives $(2, >)$. Since $(2, >) < \infty$, we found a new minimum, and K becomes $C_2\downarrow_{\mathbb{P}}$, i. e. $p_1 > 2$. Pick $\mathbf{s}_3$ from $\mathbf{W}$: it is not a target, therefore we compute its successors $\mathbf{s}_4 = (\ell_3, x \geq 2 \land p_1 = 2 \land 1 < p_2 < 2)$ and $\mathbf{s}_5 = (\ell_3, x \geq 2 \land p_1 = p_3 = 2 \land p_2 > 1)$. Pick $\mathbf{s}_4$: it is a target, with $\mathsf{GetMin}(C_4, p_1) = (2, =)$. As $(2, =) < (2, >)$, we found a new minimum, and K is replaced with $C_4\downarrow_{\mathbb{P}}$, i. e. $p_1 = 2 \land 1 < p_2 < 2$. Pick $\mathbf{s}_5$: it is a target, with $\mathsf{GetMin}(C_4, p_1) = (2, =)$. As $(2, =) = (2, =)$, we found an

Algorithm 1: $\mathsf{MinParamSynth}(\mathcal{A}, T, p)$

```
input  : A PTA $\mathcal{A}$ with symbolic initial state $\mathbf{s}_0 = (\ell_0, C_0)$, a set of target
         locations $T$, a parameter $p$.
output : Constraint $K$ over the parameters.
1  $\mathbf{W} \leftarrow \{\mathbf{s}_0\}$                                   // waiting set
2  $\mathbf{P} \leftarrow \emptyset$                                          // passed set
3  $Opt \leftarrow \infty$                                                    // current optimum
4  $K \leftarrow \bot$                                                        // current optimum valuations
5  while $\mathbf{W} \neq \emptyset$ do
6      Pick $\mathbf{s} = (\ell, C)$ from $\mathbf{W}$
7      $\mathbf{W} \leftarrow \mathbf{W} \setminus \{\mathbf{s}\}$
8      $\mathbf{P} \leftarrow \mathbf{P} \cup \{\mathbf{s}\}$
9      if $\ell \in T$ then                                                   // s is a target state
10         $\mathbf{s}_{opt} \leftarrow \mathsf{GetMin}(C, p)$                // compute local optimum
11         if $\mathbf{s}_{opt} < Opt$ then                                   // the optimum is strictly better
12             $Opt \leftarrow \mathbf{s}_{opt}$                              // we found a new best optimum: replace it
13             $K \leftarrow C{\downarrow_{\mathbb{P}}}$                      // completely replace the found valuations
14         else if $\mathbf{s}_{opt} = Opt$ then                              // the optimum is equal to the one known
15             $K \leftarrow K \vee C{\downarrow_{\mathbb{P}}}$               // add the found valuations
16     else                                                                   // otherwise explore successors
17         for each $\mathbf{s}' \in \mathsf{Succ}(\mathbf{s})$ do
18             if $\mathbf{s}' \notin \mathbf{W} \wedge \mathbf{s}' \notin \mathbf{P}$ then $\mathbf{W} \leftarrow \mathbf{W} \cup \{\mathbf{s}'\}$
19 return $K$
```

equally good minimum, and K is improved with $C_5{\downarrow_{\mathbb{P}}}$, giving a new K equal to $(p_1 = 2 \wedge 1 < p_2 < 2) \vee (p_1 = p_3 = 2 \wedge p_2 > 1)$. As $\mathbf{W} = \emptyset$, K is returned.

Algorithm 1 is a semi-algorithm; if it terminates with result K, then K is a solution for the $\mathsf{MinParamSynth}$ problem. Correctness follows from the fact that the algorithm explores the entire parametric zone graph, except for successors of target states (from [19,20] we have that successors of a symbolic state can only restrict the parameter constraint, hence we cannot improve). Furthermore, the minimum is tracked and updated whenever a target state is reached. Synthesis can effectively be achieved for PTAs with a single clock, a decidable subclass.

Proposition 5 (synthesis for one-clock PTAs). *The solution to the minimal-parameter reachability synthesis can be computed for 1-clock PTAs using a finite union of polyhedra.*

5 Minimal time reachability synthesis

For minimal-time reachability and synthesis, we assume that the PTA contains a global clock x_{global} that is never reset. In case this is not true, we extend the PTA by simply adding a 'dummy' clock x_{global} without any associated guards or invariants for it.

Algorithm 2: $\mathsf{MinTimeSynth}(\mathcal{A}, T, x_{global})$

```
input  : A PTA $\mathcal{A}$ with symbolic initial state $\mathbf{s}_0 = (\ell_0, C_0)$, a set of target
         locations $T$, a global clock that never resets $x_{global}$.
output : Minimal time $T_{opt}$ constraint $K$ over the parameters.
1  $\mathbf{Q} \leftarrow \{(0, \mathbf{s}_0)\}$                          // priority queue ordered by time
2  $\mathbf{P} \leftarrow \emptyset$                                        // passed set
3  $K \leftarrow \bot$                               // current optimum parameter valuations
4  $T_{opt} \leftarrow \infty$                                     // current optimum time
5  while $\mathbf{Q} \neq \emptyset$ do
6      $(t, \mathbf{s} = (\ell, C)) = \mathbf{Q}.\mathsf{Pop}()$       // take head of the queue and remove it
7      $\mathbf{P} \leftarrow \mathbf{P} \cup \{\mathbf{s}\}$
8      if $t > T_{opt}$ then break
9      else if $\ell \in T$ then               // when s is a target state and $t \leq T_{opt}$
10         $K \leftarrow K \vee (C \wedge x_{global} = t){\downarrow_{\mathbb{P}}}$         // valuations for which $t = T_{opt}$
11     else                                    // otherwise explore successors
12         for each $\mathbf{s}' \in \mathsf{Succ}(\mathbf{s})$ do
13             if $\mathbf{s}' \in \mathbf{Q} \vee \mathbf{s}' \in \mathbf{P}$ then continue          // ignore seen states
14             $t' \leftarrow \mathsf{GetMin}(\mathbf{s}'.C, x_{global})$              // get minimal time of $\mathbf{s}'.C$
15             if $t' \leq T_{opt}$ then        // only add states not exceeding $T_{opt}$
16                 if $\mathbf{s}'.\ell \in T \wedge t' < T_{opt}$ then
17                     $T_{opt} \leftarrow t'$                    // new lower time to target
18                 $\mathbf{Q}.\mathsf{Push}((t', \mathbf{s}'))$              // add to the priority queue
19 return $(T_{opt}, K)$
```

We give $\mathsf{MinTimeSynth}(\mathcal{A}, T, p)$ in Algorithm 2. We maintain a *priority queue* $\mathbf{Q}$ of waiting symbolic states and order these by their minimal time (for the initial state this is 0). We further maintain a set $\mathbf{P}$ of passed states, a current time optimum T_{opt} (initially ∞), and the associated optimal valuations K. We first explain the synthesis algorithm and then the reachability variant.

Minimal-time reachability synthesis. While $\mathbf{Q}$ is not empty, the state with the lowest associated minimal time t is popped from the head of the queue (line 6). If this time t is larger than T_{opt} (line 8), then this also holds for all remaining states in $\mathbf{Q}$. Also all successor states from $\mathbf{s}$ (or successors of any state from $\mathbf{Q}$) cannot have a better minimal time, thus we can end the algorithm.

Otherwise, if $\mathbf{s}$ is a target state, we assume that $t \not< T_{opt}$ and thus $t = T_{opt}$ (we guarantee this property when pushing states to the queue). Before adding the parameter valuations to K in line 10, we intersect the constraint with $x_{global} = t$ in case the clock value depends on parameters, e. g. if C is $x_{global} = p$.[6]

[6] In case t is of the form $(c, >)$ with $c \in \mathbb{Q}_+$, then the intersection of C with the linear term $x_{global} = t$ would result in $\bot$, as the exact value t is not part of the constraint. In the implementation, we intersect C with $x_{global} = t + \varepsilon$, for a small $\varepsilon > 0$.

If $\mathbf{s}$ is not a target state, then we consider its successors in lines 12-18. We ignore states that have been visited before (line 13), and compute the minimal time of $\mathbf{s}'$ in line 14. We compare t' with T_{opt} in line 15. All successor states for which t' exceeds T_{opt} are ignored, as they cannot improve the result.

If $\mathbf{s}'$ is a target state and $t' < T_{opt}$, then we update T_{opt}. Finally, the successor state is pushed to the priority queue in line 18. Note that we preserve the property that $t \not< T_{opt}$ for the states in $\mathbf{Q}$.

Minimal-time reachability. When we are interested in just a single parameter valuation, we may end the algorithm early. The algorithm can be terminated as soon as it reaches line 10. We can assert at this point that T_{opt} will not decrease any further, since all remaining unexplored states have a minimal time that is larger than or equal to T_{opt}.

Algorithm 2 is a semi-algorithm; if it terminates with result (T_{opt}, K), then K is a solution for the MinTimeSynth problem. Correctness follows from the fact that the algorithm explores exactly all symbolic states in the parametric zone graph that can be reached in at most T_{opt} time, except for successors of target states. Note (again) that successors of a symbolic state can only restrict the parameter constraint. Furthermore, T_{opt} is checked and updated for every encountered successor to ensure that the first time a target state is popped from the priority queue $\mathbf{Q}$, it is reached in T_{opt} time (after which T_{opt} never changes).

6 Experiments

We implemented all our algorithms in the IMITATOR tool [9] and compared their performance with the standard (non-minimization) EFSynth parameter synthesis algorithm from [20]. For the experiments, we are interested in analysing the performance (in the form of computation time) of each algorithm, and comparing that with the performance of standard synthesis.

Benchmark models. We collected PTA models and properties from the IMITATOR benchmarks library [5] which contains numerous benchmark models from scientific and industrial domains. We selected all models with reachability properties and extended these to include: (1) a new clock variable that represents the global time x_{global}, i. e. a clock that does not reset, and (2) a new parameter p_{global} along with the linear term $x_{global} = p_{global}$ for every transition that targets a goal location, to ensure that when minimizing p_{global} we effectively minimize x_{global}. In total we have 68 models, and for every experiment we used the extended model that includes both the global time clock x_{global} and the corresponding parameter p_{global}.

Subsumption. For each algorithm that we consider, it is possible to reduce the search space with the following two reduction techniques:

- *State inclusion* [18]: Given two symbolic states $\mathbf{s}_1 = (\ell_1, C_1)$ and $\mathbf{s}_2 = (\ell_2, C_2)$ with $\ell_1 = \ell_2$, we say that $\mathbf{s}_1$ is included in $\mathbf{s}_2$ if all parameter valuations for $\mathbf{s}_1$ are also contained in $\mathbf{s}_2$, e. g. C_1 is $p > 5$ and C_2 is $p > 2$. We may then conclude that $\mathbf{s}_1$ is redundant and can be ignored. This check can be performed in the successor computation (Succ) to remove included states, without altering correctness for minimal-time (or parameter) synthesis.
- *State merging* [10]: Two states $\mathbf{s}_1 = (\ell_1, C_1)$ and $\mathbf{s}_2 = (\ell_2, C_2)$ can be merged if $\ell_1 = \ell_2$ and $C_1 \cup C_2$ is a convex polyhedron. The resulting state $(\ell_1, C_1 \cup C_2)$ replaces $\mathbf{s}_1$ and $\mathbf{s}_2$ and is an over-approximation of both states. However, reachable locations, minimality, and executable actions are preserved.

State inclusion is a relatively inexpensive computational task and preliminary results showed that it caused the algorithm to perform equally fast or faster than without the check. Checking for merging is however a computationally expensive procedure and thus should not be performed for every newly found state. For all BFS-based algorithms (standard synthesis and minimal-parameter synthesis) we merge every BFS layer. For the minimal-time synthesis algorithm We empirically studied various merging heuristics and found that merging every ten iterations of the algorithm yielded the best results. We assume that both the inclusion and merging state-space reductions are used in all experiments, unless otherwise mentioned.

Run configurations. For the experiments we used the following configurations:

- MTReach: Minimal-time reachability,
- MTSynth: Minimal-time synthesis,
- MTSynth-noRed: Minimal time synthesis, without reductions,
- MPReach: Minimal-parameter reachability (of p_{global}), and
- MPSynth: Minimal-parameter synthesis (of p_{global}), and
- EFSynth: Classical reachability synthesis.

Experimental setup. We performed all our experiments on an Intel® Core™ i7-4710MQ processor with 2.50GHz and 7.4GiB memory, using a single thread. The six run configurations were executed on each benchmark model, with a timeout of 3600 seconds. All our models, results, and information on how to reproduce the results are available on https://github.com/utwente-fmt/OptTime-TACAS19.

Results. The results of our experiments are displayed in Fig. 4. We discuss each graph individually.

MTSynth vs EFSynth. We observe that for most of the models MTSynth clearly outperforms EFSynth. This is to be expected since all states that take more than the minimal time can be ignored. Note that the experiments that appear on a vertical line between $0.1s < x < 1s$ are a scaled-up variant of the same model, indicating that this scaling does not affect minimal-time synthesis. Finally, the model plotted at $(1346, 52)$ does not heavily modify the clocks. As a consequence, MTSynth has to explore most of the state space while continuously having to extract the time constraints, making it inefficient.

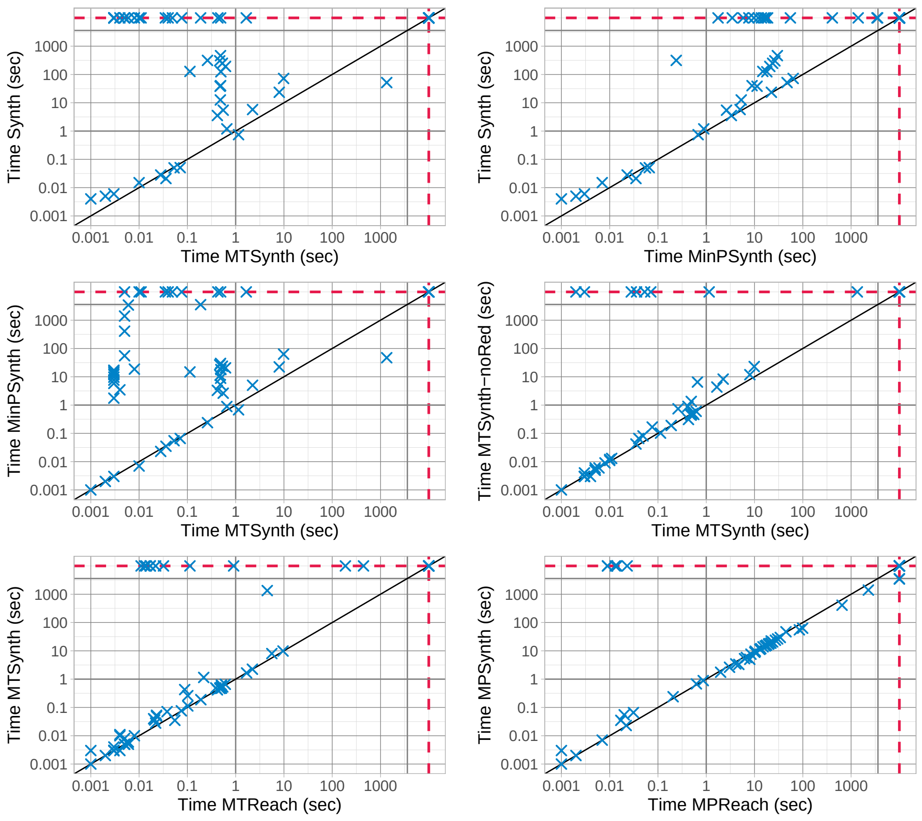


Fig. 4: Scatterplot comparisons of different algorithm configurations. The marks on the red dashed line did not finish computing within the allowed time (3600s).

MPSynth vs EFSynth. We can see that MPSynth performs more similar to EFSynth than MTSynth, which is to be expected as the algorithms differ less. Still, MPSynth significantly outperforms EFSynth. This is also because fewer states have to be explored to guarantee optimality (once a parameter exceeds the minimal value, all its successors can be ignored).

MTSynth vs MPSynth. Here, we find that MTSynth outperforms MPSynth, similar to the comparison with EFSynth. The results also show a second scalable model around $(0.003, 10)$ and we see that MPSynth is able to solve the 'bad performing model' for MTSynth as quickly as EFSynth. Still, we can conclude that the minimal-time synthesis problem is in general more efficiently solved with the MTSynth algorithm.

MTSynth vs MTSynth-noRed. Here we can see the advantage of using the inclusion and merging reductions to reduce the search space. For most models there is a non-existent to slight improvement, but for others it makes a large difference. While there is some computational overhead in performing these reductions, we conclude that this overhead is not significant enough to outweigh their benefits.

MTReach vs MTSynth. With MTReach we expect faster execution times as the algorithm terminates once a parameter valuation is found. The experiments show that this is indeed the case (mostly visible from the timeout line). However, we also observe that for quite a few models the difference is not as significant, implying that synthesis results can often be quickly obtained once a single minimal-time valuation is found.

MPReach vs MPSynth. Here we also expect MPReach to be faster than its synthesis variant. While it does quickly solve six instances for which MPSynth timed out, other than that there is no real performance gain. We also argue here that synthesis is obtained quickly when a minimal parameter bound is found. Of course we are effectively computing a minimal global time, so results may change when a different parameter is minimized.

7 Conclusion

We have designed and implemented several algorithms to solve the minimal-time parameter synthesis and related problems for PTAs. From our experiments we observed in general that minimal-time reachability synthesis is in fact faster to compute compared to standard synthesis. We further show that synthesis while minimizing a parameter is also more efficient, and that existing search space reductions apply well to our algorithms.

Aside from the performance improvement, we deem minimal-time reachability synthesis to be useful in practice. It allows for evaluating which parameter valuations guarantee that the goal is reached in minimal time. We consider it particularly valuable when reasoning about real-time systems.

On the theoretical side, we did not address the minimal-parameter reachability problem for L/U-PTAs (we only showed intractability of the synthesis). While finding the minimal valuation of a given lower-bound parameter is trivial (the answer is 0 iff the target location is reachable), finding the minimum of an upper-bound parameter boils down to reachability-synthesis for U-PTAs, a problem that remains open in general (it is only solvable for integer-valued parameters [15]), as well as to shrinking timed automata [23], but with 0-coefficients in the shrinking vector—not allowed in [23].

A direction for future work is to improve performance by exploiting parallelism. Parallel random search or swarming-based solutions could significantly speed up the computation process, as demonstrated for timed automata [25,24]. Another interesting research direction is to look at maximizing the time to reach the target, or to minimize the *upper-bound* time to reach the target (e. g. for minimizing the worst-case response-time in real-time systems); a preliminary study suggests that the latter problem is significantly more complex than the minimal-time synthesis problem.

References

1. Abdeddaïm, Y., Asarin, E., Maler, O.: Scheduling with timed automata. Theoretical Computer Science 354(2), 272–300 (2006)
2. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science 126(2), 183–235 (1994)
3. Alur, R., Henzinger, T.A., Vardi, M.Y.: Parametric real-time reasoning. In: STOC. pp. 592–601. ACM, New York, NY, USA (1993)
4. Alur, R., La Torre, S., Pappas, G.J.: Optimal paths in weighted timed automata. Theoretical Computer Science 318(3), 297–322 (2004)
5. André, É.: A benchmarks library for parametric timed model checking. In: FTSCS. LNCS, Springer (2018), models: https://www.imitator.fr/library.html
6. André, É.: What's decidable about parametric timed automata? International Journal on Software Tools for Technology Transfer (2018), to appear
7. André, É., Bloemen, V., Van de Pol, J., Petrucci, L.: Minimal-time synthesis for parametric timed automata (long version). https://www.dropbox.com/s/7dnszaqm79n4yoe/MTS-PTAs-long-version.pdf (Nov 2018)
8. André, É., Chatain, Th., Encrenaz, E., Fribourg, L.: An inverse method for parametric timed automata. IJFCS 20(5), 819–836 (2009)
9. André, É., Fribourg, L., Kühne, U., Soulat, R.: IMITATOR 2.5: A tool for analyzing robustness in scheduling problems. In: FM. LNCS, vol. 7436, pp. 33–36. Springer (2012)
10. André, É., Fribourg, L., Soulat, R.: Merge and conquer: State merging in parametric timed automata. In: ATVA. LNCS, vol. 8172, pp. 381–396. Springer (2013)
11. André, É., Markey, N.: Language preservation problems in parametric timed automata. In: FORMATS. LNCS, vol. 9268, pp. 27–43. Springer (2015)
12. Behrmann, G., Fehnker, A.: Efficient guiding towards cost-optimality in UPPAAL. In: TACAS. LNCS, vol. 2031, pp. 174–188. Springer (2001)
13. Behrmann, G., Larsen, K.G., Rasmussen, J.I.: Optimal scheduling using priced timed automata. SIGMETRICS Perform. Eval. Rev. 32(4), 34–40 (2005)
14. Beneš, N., Bezděk, P., Larsen, K.G., Srba, J.: Language emptiness of continuous-time parametric timed automata. In: ICALP, Part II. LNCS, vol. 9135, pp. 69–81. Springer (2015)
15. Bozzelli, L., La Torre, S.: Decision problems for lower/upper bound parametric timed automata. Formal Methods in System Design 35(2), 121–151 (2009)
16. Bundala, D., Ouaknine, J.: Advances in parametric real-time reasoning. In: Mathematical Foundations of Computer Science 2014 - 39th International Symposium, MFCS 2014, Budapest, Hungary, August 25-29, 2014. Proceedings, Part I. LNCS, vol. 8634, pp. 123–134. Springer (2014)
17. Courcoubetis, C., Yannakakis, M.: Minimum and maximum delay problems in real-time systems. Formal Methods in System Design 1(4), 385–415 (1992)
18. Daws, C., Tripakis, S.: Model checking of real-time reachability properties using abstractions. In: Steffen, B. (ed.) TACAS. LNCS, vol. 1384, pp. 313–329. Springer (1998)
19. Hune, T., Romijn, J., Stoelinga, M., Vaandrager, F.W.: Linear parametric model checking of timed automata. JLAP 52-53, 183–220 (2002)
20. Jovanović, A., Lime, D., Roux, O.H.: Integer parameter synthesis for timed automata. IEEE Transactions on Software Engineering 41(5), 445–461 (2015)
21. Miller, J.S.: Decidability and complexity results for timed automata and semi-linear hybrid automata. In: HSCC. LNCS, vol. 1790, pp. 296–309. Springer (2000)

22. Niebert, P., Tripakis, S., Yovine, S.: Minimum-time reachability for timed automata. In: IEEE Mediteranean Control Conference (2000)
23. Sankur, O., Bouyer, P., Markey, N.: Shrinking timed automata. Information and Computation 234, 107–132 (2014)
24. Zhang, Z., Nielsen, B., Larsen, K.G.: Distributed algorithms for time optimal reachability analysis. In: FORMATS. LNCS, vol. 9884, pp. 157–173. Springer (2016)
25. Zhang, Z., Nielsen, B., Larsen, K.G.: Time optimal reachability analysis using swarm verification. In: SAC. pp. 1634–1640. ACM (2016)