Contributions on Formal Methods for Timed and Probabilistic Systems

Ocan Sankur

Contributions on Formal Methods for Timed and Probabilistic Systems

Habilitation Thesis
Université de Rennes

Ocan Sankur

Defended on October 2, 2023, before the following jury.

Nathalie Bertrand, Inria	examiner
Thao Dang, CNRS	reviewer
Thomas Henzinger, IST Austria	reviewer
Radu Iosif, CNRS	examiner
David Parker, University of Oxford	reviewer
Mihaela Sighireanu, ENS Paris-Saclay	examiner

Abstract

Formal methods is an umbrella term for rigorous techniques for specifying and verifying computer systems, or building such systems with mathematical guarantees on their correctness, or on the absence of bugs of a given kind. Model checking is a set of techniques that can formally prove properties of a given model; while controller synthesis consists, using similar techniques, in completing the decription of an open system to ensure a given property by construction. One of the targets of formal methods is systems with quantitative aspects such as real-time constraints and probabilities. In this thesis I summarize my contributions on algorithms for model checking and controller synthesis on timed and probabilistic systems, focusing on timed automaton and Markov decision process formalisms.

Table of Contents

1 Introduction

Formal methods is an umbrella term for rigorous techniques for specifying and verifying computer systems, or building such systems with mathematical guarantees on their correctness, or on the absence of bugs of a given kind. Historically, such techniques have been developed for use in the design of hardware components but also for embedded software in the aerospatial and railway domains. Among several formal methods techniques, *model checking* consists in an exhaustive analysis of the sets of configurations of a given formal model with respect to a formal specification [CES09]. Considered formal models include automata-based formalisms and their extensions, process algebra, but also (formal semantics of) programs; while formal specifications vary from simple safety properties (such as assertions) to liveness properties, and richer ones expressed in linear-time or branching-time temporal logics.

Another approach closely related to model checking is that of controller synthesis. Here, one considers a formal model of a system that is assumed to be only partially described. Some of the inputs of the system are *uncrontrollable* and their values are chosen adversarially at each step, while other inputs are *controllable* and are to be chosen by the system. Controller synthesis consists in applying techniques similar to model checking to compute a control strategy so as to ensure that *all* induced behaviors satisfy a given formal specifications, which can be safety, liveness, or expressed in temporal logics.

Model checking has been in use in several application domains. The research community has made significant progress in model checking algorithms, improving its scalability, and the expresivity of the considered features both in models and specification. There are however remaining challenges to enable model checking and more broadly formal method techniques to be adopted in a larger setting.

One direction that would help target a broader set of applications and increase the precision of the verification questions, is the consideration of quantitative aspects both in models and properties. More precisely, we target, in this thesis, the study of models with explicitly modeled time constraints and probabilities. Models with time constraints allow modeling real-time systems more precisely. This consists in using formalisms in which time constraints such as durations, communication latencies, and deadlines are modeled explicitly. This also means that properties specific to time constraints can be formulated rather than reasoning at a higher abstraction level. Second, we consider probabilistic systems, that is, systems whose behaviors follow probability distributions, and develop verification and controller synthesis techniques for such systems. Such formalisms allow one to model computer systems that are in a stochastic environment or those that use randomization in their decisions so many interesting properties do require techniques that explicitly consider probabilities.

We motivate and present our contributions in both types of systems.

Real-Time Systems It has been argued that notions of *real time* can be expressed in standard formalisms used in finite-state model checking [Lam05], for instance, using logical notions of time, and using temporal logics to reason with various events and their orders. Although many properties of real-time systems can indeed be verified in this way, timed models, that is, models in which time constraints appear explicitly have various advantages. First, timed formalisms can allow one to describe models more succinctly than using finite-state models [S11]; moreover, efficient representations, data structures, and algorithms can be designed to handle these more efficiently rather than relying on general techniques. We consider here *timed automata* [AD94] as models for real-time systems. Existing algorithms exploit the particular structure of the clocks that are used to represent time and allow one to have efficient state space exploration techniques [BY04], but also sound and complete abstractions and reductions techniques [S7]. Such optimizations would be impossible if time constraints were not distinguished in the model. Second, having timed models means that verification questions particular to real-time systems can be formulated; hence a more precise analysis can be made. For instance, specifications can refer to real-time durations, and questions related to the *robustness* of models can be asked, in order to reveal if the satisfaction of specifications is robust to small changes in observed durations.

As in many applications of formal methods, the process of verification or controller synthesis suffer from state space explosion, which limits the scalability of verification algorithms unless special state-space reduction techniques are applied. In timed automata, state space explosion can be caused by two factors: first, as in finite-state models, large discrete state-spaces, *e.g.* due to the presence of many subsystems, or several discrete variables, can impair verification performance. Second, a large number of clocks or complex time constraints can mean that the induced state space is too large to analyze. While the first factor is common to finite-state model checking, in timed automata, the state-space can grow exponentially in the size of the clocks as well [AD94][S11], so particular care must be taken when dealing with clocks.

The most popular algorithms on timed automata are based on an explicit enumeration of the discrete states, and the use of the *zone* data structure [HNSY94, BY04] to represent clock values, as witnessed by model checking tools such as Uppaal [BDL$^+$06a] and TChecker [HP]. The issue with large discrete state spaces have been known, and many attempts have been made as well to develop techniques that can handle both sources of complexity (see *e.g.* [Wan01, BLN03, EFGP10, NSL$^+$12a, TM15]). These algorithms allow handling some large models, but they could not provide tools that are systematically better than zone-based algorithms; they succeed in some instances, but underperform on others. One of the main challenges in formal verification of real-time systems is to develop algorithms that address both sources of state space explosion in a robust way.

In **Chapter 2**, we present algorithms that target model checking of real-time systems with large discrete state spaces. In these works, our approach consisted in finding a way of applying efficient finite-state model checking techniques while still benefiting from known results from zone-based model checking, such as extrapolation operators. More specifically, **Section 2.2.1** presents a technique based on predicate abstraction [GS97] applied to timed automata verification, and its application with both enumerative and symbolic techniques based on binary decision diagrams. These results were part of the **PhD thesis** of Victor Roussanaly (defended in 2020) whom I co-supervised. **Section 2.2.2** presents a technique that consists in reducing timed automata verification to finite-state model checking via automata learning techniques which are used to build a finite-state representation of the clock constraints. The latter approach is also applied to controller synthesis in **Section 2.3.2**.

Another problem addressed in this chapter is *robustness verification* in timed automata. As

for any abstract formalism, once desired properties of a system are proven on the model, a crucial question that remains is the *robustness* of these properties against the assumptions that have been made. What is the extent to which the assumptions behind the model can be relaxed while a given property still holds?

Here, we are interested in the robustness against timing imprecisions. A lot of work has been done in the timed automata literature to endow timed automata with a realistic semantics, and take imprecisions into account, *e.g.* [GHJ97, DDR05, AT05]. The works [Pur00] and [DDMR08] showed that perturbations on clocks, *i.e.* imprecisions or clock drifts, regardless of how small they are, may drastically change the behavior in some models. These observations mean that there is a need for verification tools to check the robustness of timed automata, that is, whether the behavior of a given timed automaton is preserved in the presence of perturbations, and to compute safe bounds on such perturbations.

We consider the robustness of timed automata for safety properties under timing imprecisions modeled by *guard enlargement*, consisting in relaxing each guard of the form $x \in [a, b]$ to $x \in [a - \delta, b + \delta]$ where δ is a parameter. Our goal is to decide if for some $\delta > 0$, the enlarged timed automaton satisfies its specification, and if this is the case, compute a safe upper bound on δ.

It has been argued that robustness is a modelling problem: one can indeed incorporate the imprecision δ in the model and run model checking [AT05]. However, incorporating such imprecisions explicitly in the model can enlarge the state space and impair the model checking process. In fact, modeling guard enlargement of, say, $\delta = 0.1$, requires multiplying all integer constants by 10; and because the size of the state space is in $O(M^X)$ where M is the largest constant, and X the number of clocks, model checking can become infeasible in some cases. Coming back to the two challenges for timed automata model checking mentioned at the beginning, robustness verification can suffer from both issues: the size of the discrete state space can always be an obstacle, and in addition, the presence of the guard relaxations can render time constraints overly complicated for efficient verification. In **Section 2.2.3**, we present a parameter synthesis algorithm which solves the robust verification problem more efficiently than a reduction to standard model checking by modeling the imprecisions explicitly.

The robustness question is then considered in the setting of controller synthesis in **Section 2.3.3**. Here, the goal is to study efficient algorithms to ensure a Buchi condition in a timed automaton in which each delay suggested by the controller is adversarially perturbed within a bounded amount. This cannot be modeled by guard enlargement since we assume that the guards must always be satisfied (the controller must choose delays that ensure this), and because a game semantics must be used in order to render the perturbations adversarial. While the theory was studied before and allows an elegant classification of robustly controllable and uncontrollable timed automata [S30][BA11], this chapter presents an algorithm and experiments based on zones. We also present here an extension of this setting to stochastic adversaries where the perturbations are chosen independently at random at each step.

Probabilistic Systems Probabilities are useful when modelling computer systems faithfully. These can mainly arise in two ways. First, the system under study can be in an environment that is stochastic by nature (or such that a stochastic model describes it faithfully). This can be the case in control applications where inputs to the system can contain errors or noise, or when some failures can be assumed to follow a probability distribution. In this case, the designer might want to formally verify the behaviors of the system assuming a particular stochastic model for the environment. Second, the system under study can be itself probabilistic, such as a randomized

algorithm. In this case, even under a deterministic environment, the behaviors are stochastic. We call these *probabilistic systems*.

Formal methods have been extended to analyze probabilistic systems [dA97, BK08]. Although some probabilistic systems can be analyzed using powerful Monte Carlo methods, the formal methods community has mainly brought contributions allowing one to 1) specify temporal properties such as linear temporal logic (LTL) [Pnu77] and probabilistic computation tree logic (PCTL) [HJ94]; 2) and analyzing probabilistic and *nondeterministic* systems for which simulation is difficult to apply.

Such systems are modeled as Markov decision processes (MDP) [Put94] which can express both probabilistic and nondeterministic behaviors. The main questions of interest on MDPs are the computation of minimal and maximal probabilities of satisfying a given specification, or minimal/maximal expectation of a quantitative objective. Hence, these extremal probabilities correspond to the average performance or to the probability of satisfying the specification in the system under study in the best and the worst cases. These techniques can thus be applied for formal verification of MDPs. Moreover, computing such extremal probabilities often amounts to computing a witness scheduler. Therefore, the problems at hand are also seen sometimes as controller synthesis problems, where the goal is to find a scheduler to realize a specification with maximum probability (or to optimize an objective).

My contributions detailed in this chapter apply to both points of views. In this manuscript, we put a focus on a particular objective, namely, *total payoff*, that is the sum of the rewards seen until reaching a target state. This allows us to define the *stochastic shortest path (SSP)* problem, an extension of the classical shortest path problem in probabilistic systems. This objective simply measures the sum of the costs of an execution, and has been a classical one in MDP and reinforcement learning literature.

The SSP is significantly harder than computing shortest paths in graphs. In fact, in MDPs, the object to be computed is not a single path; but, in the simplest setting, the goal is to compute a scheduler that minimizes the expectation of the total payoff. As for graphs, negative cycles are an issue, and their detection and the way these are handled are also more involved. This is one of the difficulties we address in **Section 3.2.1**: while the SSP had been studied for MDPs with nonnegative weights only, the case of general weights (including positive and negative ones) had been unsolved. We solve the general SSP problem here and provide a polynomial-time solution. In **Section 3.2.2**, we consider a multi-objective optimization view of the SSP: we consider both the expected total payoff, and its variance. We give algorithms to minimize variance among expectation-optimal schedulers, and study *variance-penalized expectation* which consists in minimizing a linear combination of the expectation and the variance. To go further in the expressivity of the objectives, we consider, in **Section 3.2.3**, *percentile* constraints on the total payoff, including multi-dimensional MDPs, that is, with multiple weight functions. A percentile constraint enforces for instance that the probability of the total payoff to be less than a threshold θ with probability at least α. Using a Boolean combination of such constraints, one can have more precision on the probability distribution induced by the computed scheduler compared to using only expectation. Last, we present in **Section 3.3** a setting where nondeterministic uncertainties on the model itself can be taken into account while giving formal guarantees on the average performance. More precisely, we consider a variant of MDPs given with a finite set of probabilistic transition functions, each corresponding to a different execution *environment*. The set of these environments is known a priori but not the actual one during execution. We seek to compute a single scheduler with guarantees under all environments.

All these works have the common objective of improving the precision of the computed schedulers by allowing the designer to have finer control on the scheduler to be computed and the probability

distribution it induces.

Other Contributions This manuscript only contains selected works from the two lines presented above. I summarize here other topics I have worked on, that are not included here.

I have been interested in applications of formal verification in collaboration with industrial partners. One direction I have been pursuing on the verification of timed systems is the study of real-time requirements with algorithms to check the consistency of a set of requirements [S21], and to repair a given inconsistent set of requirements [S22]. This line of work is pursued in collaboration with Mitsubishi Electric. Furthermore, I participated in the supervision of the **PhD thesis** of Abdul Majith (defended in 2022) on compositional verification techniques for software-defined networks, in collaboration with Nokia Bell Labs.

I have worked on the *parameterized* verification of distributed algorithms, that is, verification for *all* instantiations of the number of processes and graph topologies. This includes decidability results for round-based protocols with shared memory [S6] which is part of the ongoing **PhD thesis** of Nicolas Waldburger (started in 2021). I have also considered case studies that combine several nontrivial techniques to scale the verification to large instances for distributed protocols [S31]; and a software tool for parameterized verification based on counting and predicate abstraction [S32]. I co-supervised the **PhD thesis** of Suman Sadhukhan (defended in 2021) on *congestion games*, which are multiplayer games that are parameterized in the number of players, where each player minimizes their cost to reach their destination while the cost at each traversed edge is incurred by the congestion, i.e. the number of players that use that edge simultaneously [S4, S5].

Another topic is the controller synthesis problem on finite-state systems described succinctly by circuits. I have contributed to symbolic algorithms for solving this problem using binary decision diagrams, and using compositionality and abstraction techniques [S13, S15]. We have participated in the synthesis competition[1] with a solver developed in this setting [S20][JPA+22]. In order to extend the study of games to several players, I have studied the notion of *admissibility* in various games including concurrent and turn-based ones, and those with quantitative objectives, imperfect information, and clocks [S3, S12, S2, S14]. Admissibility is a useful notion for comparing strategies of the players and to define their rational behaviors; it can be exploited for controller synthesis in a multi-player setting [S16].

Last, a more exploratory topic is that of planning algorithms for the multi-agent path finding problem subject to connectivity constraints [S25, S19, S18, S17]. Here, the goal is to find paths for a set of agents in a graph such that they avoid pairwise collisions but also stay *connected* to each other: a pair of agents a, b is connected if their distance is within a threshold; or if a is within the threshold to another agent who is connected to b. This work was done within the **PhD thesis** of Arthur Queffelec (defended in 2021) which I co-supervised.

[1] www.syntcomp.org

2 Algorithms for Timed Systems

In this chapter, I present my contributions on model checking of systems with explicit real-time constraints modeled by timed automata [AD94]. These are an extension of finite automata using continuous clock variables and are one of the standard formalisms used in formal verification or controller synthesis.

Model checking algorithms for timed automata often represent clock constraints efficiently using convex polyhedra called *zones* [BM83]. Algorithms based on this data structure have been implemented in several tools such as Uppaal [BDL+06a] and the open source model checker TChecker [HP]. The research community has done a significant effort in improving the efficient treatment of zones using *extrapolation operators*, which are abstractions that are sound and complete for location-reachability[2] While these results significantly improved the efficiency of model checking, they only address one cause of the aforementioned state space explosion: zone-based algorithms are based on an explicit enumeration of the discrete state space, and apart from symmetry reduction in Uppaal, no reduction technique is used for these in current tools. The situation is similar for timed games used for synthesizing real-time controllers. Zone-based algorithms were developed to solve timed games and compute control strategies [CDF+05b], and are available in the Uppaal TIGA tool [BCD+07b]. These algorithms suffer from the same limitations as the zone-based model checking algorithms. Although they can be efficient on instances with small discrete state spaces, they do not scale well to large systems.

Several works have attempted at improving the performance on models with large discrete state spaces. Extensions of binary decision diagrams (BDD) with clock constraints have been considered both for continuous time [Wan01, BLN03, EFGP10] and discrete time [NSL+12a, TM15]. Another approach is to use predicate abstraction on clock variables that enables efficient finite-state verification techniques based on BDDs or SAT solvers [CGMT14, CGM+19]. Other approaches targeting the formal verification of real-time systems with large discrete state spaces include encodings of timed automata semantics in Boolean logic include [KJN11, SB03]. An extension of and-inverter graphs were used in [DDD+12] that uses predicates to represent the state space of linear hybrid automata. The only attempt we are aware of, for taming the state space of timed games was the counter-example guided abstraction refinement scheme applied to timed games in [PEM11].

We present our contributions in two parts. Section 2.2 contains our results on model checking algorithms; and Section 2.3 contains our results on controller synthesis.

[2] A survey on recent advances on this approach can be found in [S7].

2.1 Definitions

Labeled Transition Systems and Finite Automata. We denote finite *labeled transition systems (LTS)* as tuples (Q, q^0, Σ, T) where Q is the set of states, $q^0 \in Q$ is the initial state, Σ is a finite alphabet, $T \subseteq Q \times \Sigma \cup \{\epsilon\} \times Q$ is the transition relation (ϵ labeling silent transitions). A *finite automaton* is an LTS given with a set of accepting states $F \subseteq Q$, and is written (Q, q^0, Σ, T, F). A *run* of an automaton is a sequence $q_1 e_1 q_2 e_2 \dots q_n$ where $q_1 = q^0$, $e_i = (q_i, \sigma_i, q_{i+1}) \in T$ for each $1 \leq i \leq n-1$. The *trace* of the run is the sequence $\sigma_1 \sigma_2 \dots \sigma_{n-1}$. An *accepting run* starts at q^0 and ends in F. The language of $\mathcal{A}$ is the set of the traces of all accepting runs of $\mathcal{A}$, and is denoted by $\mathcal{L}(\mathcal{A})$. We will consider *deterministic finite automata (DFA)* which have at most one edge for each label from each state.

The *parallel composition* of two automata $\mathcal{A}_i = (Q_i, q_i^0, \Sigma, T_i, F_i)$, $i \in \{1, 2\}$, defined on the same alphabet, is the automaton $\mathcal{A}_1 \parallel \mathcal{A}_2 = (Q, q^0, \Sigma, T, F)$ with $Q = Q_1 \times Q_2$, $q^0 = (q_1^0, q_2^0)$, $F = F_1 \times F_2$, and T contains $((q_1, q_2), \sigma, (q_1', q_2'))$ for all $(q_1, \sigma, q_1') \in T_1$, and $(q_2, \sigma, q_2') \in T_2$; and $((q_1, q_2), \epsilon, (q_1', q_2))$ for all $(q_1, \epsilon, q_1') \in T_1$, and $q_2 \in Q_2$; and symmetrically, $((q_1, q_2), \epsilon, (q_1, q_2'))$ for all $(q_2, \epsilon, q_2') \in T_2$, and $q_1 \in Q_1$.

Timed Automata. We fix finite set of clocks $\mathcal{C}$. *Clock valuations* are the elements of $\mathbb{R}_{\geq 0}^{\mathcal{C}}$. For $R \subseteq \mathcal{C}$ and a valuation v, $v[R \leftarrow 0]$ is the valuation defined by $v[R \leftarrow 0](x) = v(x)$ for $x \in \mathcal{C} \setminus R$ and $v[R \leftarrow 0](x) = 0$ for $x \in R$. Given $d \in \mathbb{R}_{\geq 0}$ and a valuation v, $v+d$ is defined by $(v+d)(x) = v(x)+d$ for all $x \in \mathcal{C}$. We extend these operations to sets of valuations in the standard way. We write $\vec{0}$ for the valuation that assigns 0 to every clock.

We consider a clock named 0 which has the constant value 0, and let $\mathcal{C} \cup \{0\} = \mathcal{C} \cup \{0\}$. An *atomic guard* is a formula of the form $x \bowtie k$, or $x - y \bowtie k$ where $x, y \in \mathcal{C} \cup \{0\}$, $k, l \in \mathbb{N}$, and $\bowtie \in \{<, \leq, >, \geq\}$. A *guard* is a conjunction of atomic guards. A valuation v satisfies a guard g, denoted $v \models g$, if all atomic guards are satisfied when each $x \in \mathcal{C}$ is replaced by $v(x)$. Let $\Phi_{\mathcal{C}}$ denote the set of guards for $\mathcal{C}$.

A *timed automaton* $\mathcal{T}$ is a tuple $(L, \ell_0, \Sigma, \mathrm{Inv}, \mathcal{C}, E, F)$, where L is a finite set of locations, Σ is the alphabet, $\mathrm{Inv} : L \to \Phi_{\mathcal{C}}$ the invariants, $\mathcal{C}$ is a finite set of clocks, $E \subseteq \mathcal{L} \times \Sigma \times \Phi_{\mathcal{C}} \times 2^{\mathcal{C}} \times \mathcal{L}$ is a set of edges, and $\ell_0 \in L$ is the initial location. An edge $e = (\ell, g, \sigma, R, \ell')$ is also written as $\ell \xrightarrow{g, \sigma, R} \ell'$. $F \subseteq L$ is the set of accepting locations.

A *run* of $\mathcal{T}$ is a sequence $r = q_1 e_1 q_2 e_2 \dots q_n$ where $q_i \in L \times \mathbb{R}_{\geq 0}^{\mathcal{C}}$, $q_1 = (\ell_0, \vec{0})$, and writing $q_i = (\ell, v)$, we have $v \in \mathrm{Inv}(\ell)$, and either $e_i \in \mathbb{R}_{>0}$ and $v + e_i \in \mathrm{Inv}(\ell)$, in which case $q_{i+1} = (\ell, v + e_i)$, or $e_i = (\ell, g, \sigma, R, \ell') \in E$, in which case $v \models g$ and $q_{i+1} = (\ell', v[R \leftarrow 0])$. The run is accepting if the last location is in F. The *trace* of the run r is the word $\sigma_0 \sigma_1 \dots \sigma_n$ where σ_i is the label of edge e_i.

The *untimed language* of the timed automaton $\mathcal{T}$ is the set the traces of the accepting runs of $\mathcal{T}$, and is denoted by $\mathcal{L}(\mathcal{T})$.

We consider the parallel composition of a finite automaton $\mathcal{A} = (Q, q^0, \Sigma, T, F)$ and a timed automaton $\mathcal{T} = (L, \ell_0, \Sigma, \mathrm{Inv}, \mathcal{C}, E, F_{\mathcal{T}})$ which is a new timed automaton. Intuitively, a transition labeled by σ consists in an arbitrary number of silent transitions of $\mathcal{A}$, followed by a joint σ-transition of both components. The guard and the reset of the overall transition are those of the transition of $\mathcal{T}$. Formally, let $\mathcal{A} \| \mathcal{T} = (L', \ell_0', \Sigma, \mathrm{Inv}', \mathcal{C}, E', F')$ with $L' = Q \times L'$, $\mathrm{Inv}' : (q, \ell) \mapsto \mathrm{Inv}(\ell)$, $\ell_0' = (q_0, \ell_0)$, and E' contains all edges of the form $((q, \ell), g, \sigma, R, (q', \ell'))$ such that $(\ell, g, \sigma, R, \ell') \in E$, and there exists a sequence $q = q_0, q_1, \dots, q_k, q_{k+1} = q'$ of states of $\mathcal{A}$ such that $(q_0, \epsilon, q_1), \dots, (q_{k-1}, \epsilon, q_k), (q_k, \sigma, q_{k+1})$ are transitions of $\mathcal{A}$. We let $F' = F \times F_{\mathcal{T}}$.

It follows from the definition of the parallel composition that $\mathcal{L}(\mathcal{A} \| \mathcal{T}) = \mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{T})$.

An example is given in Figure 2.2 which shows how a simple scheduling setting can be modeled in this way. Here, the finite automaton is simple and only stores the mapping from machines to the tasks they are executing. Typically, if the machines or the processes executing tasks have internal states, these could be modeled in $\mathcal{A}$ as well without altering the timed automaton.

Zones A *zone* is a set of clock valuations expressible by a clock guard. Zones can be represented with *difference-bound matrices (DBM)* [BM83, BY04]. A DBM is a $\mathcal{C} \times \mathcal{C}$-matrix whose entry (x, y) contains $(\leq, k)$ or $(<, k)$ representing the constraint $x - y \leq k$ (resp. $x - y < k$). An example is given in Fig. 2.1. There are efficient algorithms to compute successors using zones and to check emptiness; see [BY03].

$$\begin{array}{c} \\ 0 \\ x \\ y \end{array} \begin{array}{c} \begin{array}{ccc} 0 & x & y \end{array} \\ \begin{pmatrix} (0,\leq) & (-1,\leq) & (-1,\leq) \\ (3,\leq) & (0,\leq) & (2,\leq) \\ (2,\leq) & (1,\leq) & (0,\leq) \end{pmatrix} \end{array}$$

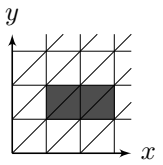


Figure 2.1: A DBM (above) representing the set $1 \leq x \leq 3 \wedge 1 \leq y \leq 2$, shown below.

Finite Automata Learning. We use finite automata learning algorithms such as L^* [Ang87, RS93] and TTT [IHS14]. In the *online* learning model, the learning algorithm interacts with a teacher in order to learn a deterministic finite automaton recognizing a hidden regular language known to the teacher. The algorithm can make two types of queries. A *membership query* consists in asking whether a given word belongs to the language, to which the teacher answers by yes or no. An *equivalence query* consists in creating a hypothesis automaton H, and asking the teacher whether H recognizes the language. The teacher either answers yes, or no and provides a counterexample word which is in the symmetric difference of $\mathcal{L}(H)$ and of the target language. Learning algorithms typically make a large number of membership queries, and a smaller number of equivalence queries.

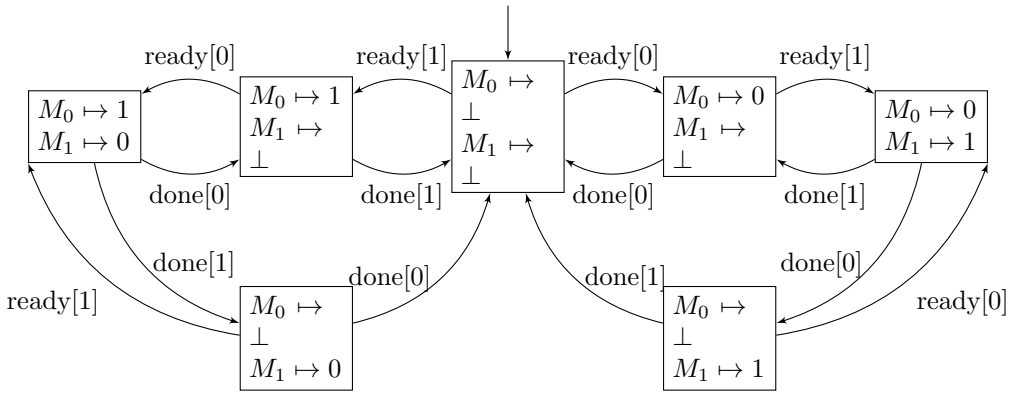
2.2 Model Checking Algorithms

2.2.1 Clock Predicate Abstraction Algorithm

One of our attempts in developing a novel algorithm that can handle both large discrete state spaces and real-time constraints was to find a good way of using *predicate abstraction* [GS97] in timed automata in order to reduce the problem to model checking in discrete state space [S27]. This section summarizes the contributions in Victor Roussanaly's PhD thesis defended in 2019 at Université de Rennes 1, which I co-supervised with Nicolas Markey.

Abstraction is a generic approach that consists in simplifying the model under study, so as to make it easier to verify [CC77]. *Existential* abstraction may only add extra behaviors, so that when a

Finite automaton $\mathcal{A}$:



Timed automaton $\mathcal{T}$:

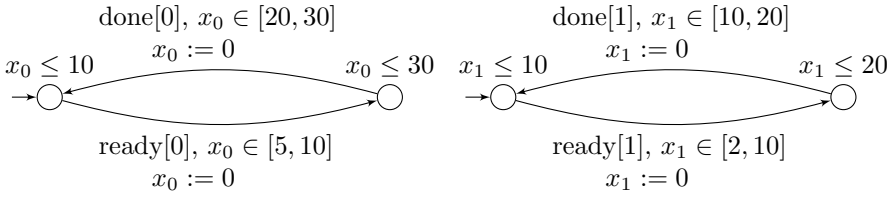


Figure 2.2: An example timed system $\mathcal{A}\|\mathcal{T}$ modeling a simple scheduling policy. The finite automaton $\mathcal{A}$ is given above and models a scheduler which schedules tasks (0 and 1) immediately when they become ready (labels read[0] and read[1]) on machines M_0 and M_1, using M_0 first if it is available. The timed automaton $\mathcal{T}$ is given below, here, as a network of the timed automata, and models interarrival times and computation times for each task.

safety property holds in an abstracted model, it also holds in the original model; if on the other hand a safety property fails to hold, the model-checking algorithms return a witness trace exhibiting the non-safe behaviour: this either invalidates the property on the original model, if the trace exists in that model, or gives information about how to automatically refine the abstraction. This approach, named counter-example guided abstraction refinement (CEGAR) [CGJ$^+$00], was further developed and used, for instance, in software verification (*e.g.* BLAST [HJMS03], SLAM [BLR11]).

The CEGAR approach has been adapted to timed automata, e.g. in [DKL07, HZH$^+$10], but the abstractions considered there only consist in removing clocks and discrete variables, and adding them back during refinement. So for most well-designed models, one ends up adding all clocks and variables which renders the method useless. Two notable exceptions are [HSW13], in which the zone extrapolation operators are dynamically adapted during the exploration, and [TM17], in which zones are refined when needed using interpolants. Both approaches define "exact" abstractions in the sense that they make sure that all traces discovered in the abstract model are feasible in the concrete model at any time. Thus, the abstractions introduced in these works may not sufficiently reduce the state space since they are not "aggressive" enough.

In our work, we consider a more general setting and study *predicate abstractions* on clock

variables. Just like in software model checking, we define abstract state spaces using these predicates, where the values of the clocks and their relations are approximately represented by these predicates. New predicates are generated if needed during the refinement step. We instantiate our approach by two algorithms. The first one is a zone-based enumerative algorithm inspired by the *lazy abstraction* in software model checking [HJMS02], where we assign a possibly different abstract domain to each node in the exploration. The second algorithm is based on binary decision diagrams (BDD): by exploting the observation that a small number of predicates was often sufficient to prove safety properties, we use an efficient BDD encoding of zones similar to one introduced in early work [SB03].

(a) Abstraction of zone $1 \leq x, y \leq 2$ (b) Abstraction of zone $y \leq 1 \wedge 1 \leq x - y \leq 2$

Figure 2.3: The abstract domain is defined by the clock constraints shown in thick red lines. In each example, the abstraction of the zone shown on the left (shaded area) is the larger zone on the right.

We apply predicate abstraction by introducing an abstraction domain that abstracts away zones by restricting the set of clock constraints that can be used to define them, while the refinement procedure computes the set of constraints that must be taken into consideration in the abstraction so as to exclude a given spurious counterexample. We implemented this idea in two ways: an enumerative algorithm where a lazy abstraction approach is adopted, meaning that possibly different abstract domains are assigned to each exploration node; and a symbolic algorithm where the abstract transition system is encoded with Boolean formulas.

Let us explain the abstract domains we consider. Assume there are two clock variables x and y. The abstraction we consider consists in restricting the clock constraints that can be used when defining zones. Assume that we only allow to compare x with 2 or 3; that y can only be compared with 2, and $x - y$ can only be compared with -1 or 2. Then any conjunction of constraints one might obtain in this manner will be delimited by the thick red lines in Fig. 2.3; one cannot define a finer region under this restriction. The figure shows the abstraction process: given a "concrete" zone, its abstraction is the smallest zone which is a superset and is definable under our restriction. For instance, the abstraction of $1 \leq x, y \leq 2$ is $0 \leq x, y \leq 2 \wedge -1 \leq x - y$ (cf. Fig. 2.3a).

Zone-based Algorithm We consider the standard zone-based enumerative algorithm with maintains a passed list (passed) of visited zones per location, and a wait list (wait) of zones to be visited at each location. We only modify it so as to apply the abstraction function (α_n) on the considered zone. Note that this abstraction function depends on the node n. In Algorithm 1, the nodes of the search tree are denoted by n, while $n.\ell$ is the location and $n.Z$ is the zone visited at that node. When a node n is covered by a node n', that is, under the condition of line 6, we stop the exploration from n and store a pointer to the covering node. When successor nodes are created for each outgoing edge from a visited node n, on line 12, choose$-$dom(n, e) returns an appropriate abstraction function for the newly created node n'.

To ensure termination, we assume that each α_n applies an abstraction which is at least the local extrapolation operator [S7]. It is clear that if the algorithm returns Safe, then safety is established,

Data: Timed automaton
while wait $\neq \emptyset$ **do**
  n := wait.pop();
  if $n.\ell = \ell_T$ **then**
    return *Trace from root to n*
  end
  if $\exists n' \in \mathsf{passed}, n.\ell = n'.\ell \wedge n.Z \subseteq n'.Z$ **then**
    $n.\mathsf{covered} := n'$
  else
    $n.Z := \alpha_n(n.Z)$;
    passed.$add(n)$;
    for $\forall e = (n.\ell, g, R, \ell')$ *s.t.* $\mathsf{Post}_e(n.Z) \neq \emptyset$ **do**
      $n' := \mathsf{node}(\ell', Z', \mathsf{choose{-}dom}(n, e))$;
      wait.$add(n')$;
    end
  end
end
return Safe;

Algorithm 1: Zone-based abstract model checking algorithm.

and the search tree describes an invariant. Otherwise we need to check the counterexample for realizability and possibly refine the abstractions.

We add an automatic refinement step to obtain a counterexample guided abstraction refinement algorithm [CGJ+03]. Realizability checks can be performed using zones (without using the abstraction operator). If the counterexample trace is spurious, then we show how to compute a set of constraints that must be added to the abstract domains of the locations in the trace so as to exclude the said trace.

This is done by computing interpolants at each step as in [HJMM04]. We do not use an SMT solver here, but directly implement the computation of interpolants using zones.

Given a pair of zones Z, Z' with $Z \cap Z = \emptyset$, an *interpolant* for Z, Z' is a zone Z'' satisfying $Z \subseteq Z''$ and $Z'' \cap Z' = \emptyset$. Specifically, we show how to compute interpolants in the form of zones which are, moreover, *minimal* in the number of constraints.

Lemma 1. *Given two zones Z, Z' with $Z \cap Z' = \emptyset$, one can compute in time $O(X^4)$ a minimal interpolant of (Z, Z').*

These results allow us to work with zones since the newly added predicates again define zones. In contrast, note that using an SMT solver to compute interpolants would yield linear arithmetic expression since the theory is needed to encode bounded executions; and we would not be able to use these in a zone-based exploration.

BDD-based Algorithm We also apply the above algorithm using Boolean techniques, namely, binary decision diagrams (BDD). We consider an order $\lhd$ on the set of clocks, and write abstract domains in the form $\mathsf{Dom} = (\mathsf{Dom}_{x,y})_{x \lhd y \in \mathcal{C} \cup \{0\}}$, where $\mathsf{Dom}_{x,y}$ is the set of pairs of constant and inequality $(k, \prec)$ (with $\prec \in \{<, \leq\}$), such that $x - y \prec k$ can be expressed precisely. In fact,

constraints of the form $x - y \leq k$ with $x \triangleright y$ are encoded using the negation of $y - x < -k$ since $(x - y \leq k) \Leftrightarrow \neg(y - x < -k)$. We thus define $\mathsf{Dom}_{x,y} = -\mathsf{Dom}_{y,x}$ for all $x \triangleright y$.

For $x, y \in \mathcal{C} \cup \{0\}$, let $\mathcal{P}_{x,y}$ denote the set of *clock predicates associated to* $\mathsf{Dom}_{x,y}$:

$$\mathcal{P}^{\mathsf{Dom}}_{x,y} = \{P_{x-y\prec k} \mid (k, \prec) \in \mathsf{Dom}_{x,y}\}.$$

Let $\mathcal{P}^{\mathsf{Dom}} = \cup_{x,y \in \mathcal{C}\cup\{0\}} \mathcal{P}_{x,y}$ denote the set of all predicates. For any Boolean formula ϕ on these predicates, let $[\![\phi]\!]$ denote the set of clock valuations that satisfy ϕ.

We also consider a set $\mathcal{B}$ of Boolean variables used to encode locations. Overall, the state space is described using Boolean formulas on these two types of variables, so states are elements of $\mathbb{B}^{\mathcal{P}\cup\mathcal{B}}$, where $\mathbb{B} = \{0, 1\}$.

We show how to implement successor computation on the abstract domain using BDD operations. Our encoding of clock constraints and semantic operations follow that of [SB03] which was given for the concrete domain. The difficulty in [SB03] was to handle a large number of predicates that appear when performing model checking. Here, thanks to the abstract domains we consider, we have a smaller number of predicates. We show how successor computation and refinement operations can be performed.

Let us consider the reduction operation, which is similar to the reduction of difference-bound matrices [BM83, BY04]. The idea is to eliminate unsatisfiable minterms from a given Boolean formula. For example, we would like to make sure that in all minterms, if $p_{x-y\leq 1}$ holds, then so does $p_{x-y\leq 2}$, when both are available predicates. Another issue is to eliminate minterms that are unsatisfiable due to triangle inequality. This is similar to the shortest path computation used to turn DBMs in canonical form.

Example 2. *Given predicates $\mathcal{P} = \{p_{x-y\leq 1}, p_{y-z\leq 1}, p_{x-z\leq 2}\}$, the formula $p_{x-y\leq 1} \wedge p_{y-z\leq 1}$ is not reduced since it contains the unsatisfiable minterm $p_{x-y\leq 1} \wedge p_{y-z\leq 1} \wedge \neg p_{x-z\leq 2}$. However, the same formula is reduced if $\mathcal{P} = \{p_{x-y\leq 1}, p_{y-z\leq 1}\}$.*

In our setting, we use a limited form of reduction; in fact, reduction is the most expensive operation in our algorithm. The following formula corresponds to 2-reduction, which intuitively amounts to applying shortest paths for paths of lengths 1 and 2:

$$\bigwedge_{\substack{(x,y)\in\mathcal{C}\cup\{0\}^2 \\ (k,\prec)\in\mathsf{Dom}_{x,y}}} \left[p_{x-y\prec k} \leftarrow \Big(\bigvee_{\substack{(l_1,\prec_1)\in\mathsf{Dom}_{x,y} \\ (l_1,\prec_1)\leq(k,\prec)}} p_{x-y\prec_1 l_1} \vee \bigvee_{\substack{z\in\mathcal{C}\cup\{0\},(l_1,\prec_1)\in\mathsf{Dom}_{x,z}, \\ (l_2,\prec_2)\in\mathsf{Dom}_{z,y} \\ (l_1,\prec_1)+(l_2,\prec_2)\leq(k,\prec)}} p_{x-z\prec l} \wedge p_{z-y\prec' l'} \Big) \right]$$

For any formula S, let us call $\mathsf{reduce}(S)$ the intersection of S with the above formula. $\mathsf{reduce}(S)$ yields a 2-reduced formula, which eliminates some of the unsatisfiable valuations of S. Because reduce does not fully capture reduction, $\mathsf{reduce}(S)$ can still have unsatisfiable valuations. Nevertheless, for any such unsatisfiable valuation, it is possible to add a new predicate to the abstract domain so that 2-reduction rules it out [S27].

To illustrate how successor computation can be done, we provide the formula for time successors. Define the following relation on $\mathcal{P} \cup \mathcal{P}'$ where $\mathcal{P}'$ is the primed versions of the predicates.

$$S_{\mathsf{Up}} = \bigwedge_{\substack{x\in\mathcal{C} \\ (k,\prec)\in\mathsf{Dom}_{x,0}}} (\neg p_{x-0\prec k} \rightarrow \neg p'_{x-0\prec k}) \bigwedge_{\substack{x,y\in\mathcal{C}\cup\{0\},x\neq 0 \\ (k,\prec)\in\mathsf{Dom}_{x,y}}} (p'_{x-y\prec k} \leftrightarrow p_{x-y\prec k}).$$

Data: $\mathcal{T} = (L, \text{Inv}, \ell_0, \mathcal{C}, E)$, ℓ_T, Dom
1 next := $\mathsf{enc}(l_0) \wedge \alpha_{\mathsf{Dom}}(\wedge_{x \in \mathcal{C}} x = 0)$;
2 layers := [];
3 reachable := false;
4 **while** $(\neg$reachable $\wedge$ next$) \neq$ false **do**
5 | reachable := reachable $\vee$ next;
6 | next := ApplyEdges(Up(next)) $\wedge \neg$reachable;
7 | layers.$push$(next);
8 | **if** (next $\wedge$ $\mathsf{enc}(l_T)) \neq$ false **then**
9 | | **return** ExtractTrace (layers);
10 **end**
11 **return** Not reachable;

Algorithm 2: Algorithm SymReach that checks the reachability of a target location l_T in a given abstract domain Dom.

S_{Up} defines a transition relation that relates abstract states. The formula says that any lower bound on a clock x must still hold after the time successor relation (the left conjunct), and that all diagonal constraints must be preserved (the right conjunct). It does not specify any other lower or upper bounds; in fact, there are several abstract states that are time successors of a given abstract state. This corresponds to non-determinism in time successor transitions.

Clock resets and guard intersection can be performed by providing appropriate transition relations using BDDs.

The overall model checking procedure is summarized in Algorithm 2. The input is a timed automaton $\mathcal{T}$, a target location ℓ_T and an initial abstract domain. The initial domain can be empty, or contain all atomic guards that appear syntactically in $\mathcal{T}$. Here, $\mathsf{enc}(\ell)$ is the Boolean encoding of the location ℓ using the variables $\mathcal{B}$. The list layers contains, at position i, the set of states that are reachable in i steps. The function ApplyEdges computes the disjunction of immediate successors by all edges. It consists in looping over all edges $e = (l_1, g, \sigma, R, l_2)$, and gathering the following image by e:

$$\mathsf{enc}(\ell_2) \wedge \mathsf{Reset}_{r_k}(\mathsf{Reset}_{r_{k-1}}(\ldots(\mathsf{Reset}_{r_1}((((\exists \mathcal{B}.A(\mathcal{B}, \mathcal{P}) \wedge \mathsf{enc}(\ell_1)) \wedge \alpha_{\mathsf{Dom}}(g)))))))),$$

where $R = \{r_1, \ldots, r_k\}$. This formula constrains the given set A to location ℓ_1, projects it to clock predicates, intersects with the abstraction of the guard of the edge e, and then applies the clock resets one by one. We thus use a partitioned transition relation and do not compute the monolithic transition relation. If the target location is reachable, ExtractTrace(layers) returns an abstract counterexample trace from the computed layers.

The full algorithm contains a refinement loop similar to the refinement in the zone-based variant of the algorithm. However, the refinement for the symbolic algorithm is more involved. In fact, due to the incomplete reduction operation, the abstract transition relation is not the most precise one: in our semantics, there may be a transition between two abstract states without any matching concrete transition between the concretizations of these states. The refinement procedure thus requires several checks in order to distinguish the source of the spurious counterexamples.

Experiments We implemented both algorithms. The symbolic version was implemented in OCaml using the CUDD library, the zone-based enumerative version was implemented in C++

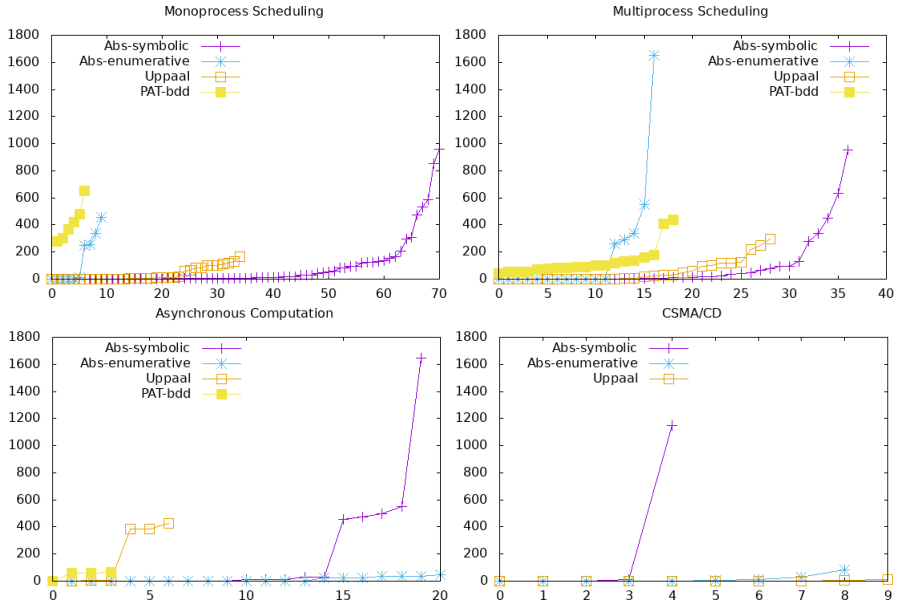


Figure 2.4: Comparison of our enumerative and symbolic algorithms (referred to as Abs-enumerative and Abs-symbolic) with Uppaal and PAT. Each figure is a cactus plot for the set of benchmarks: a point (X, Y) means X benchmarks were solved within time bound Y.

within the open-source model checker TChecker [HP]. Both prototypes take as input networks of timed automata with invariants, discrete variables, urgent and committed locations. The presented algorithms are adapted to these features without difficulty.

We evaluated our algorithms on three classes of benchmarks we believe are significant. We compare the performance of the algorithm with that of Uppaal [BDL+06a] which is based on zones, as well as the BDD-based model checker engine of PAT [NSL+12b]. The tool used in [EFGP10] was not available. We thus only provide a comparison here with the two well-maintained tools.

Figure 2.4 displays the results of our experiments. All algorithms were given 8GB of memory and a timeout of 30 minutes, and the experiments were run on laptop with an Intel i7@3.2Ghz processor running Linux. The symbolic algorithm performs best among all on the monoprocess and multiprocess scheduling benchmarks. Uppaal is the second best, but does not solve as many benchmarks as our algorithm. Our enumerative algorithm quickly fails on these benchmarks, often running out of memory. On asynchronous computation benchmarks, our enumerative algorithm performs remarkably well, beating all other algorithms. We ran our tools on the CSMA/CD benchmarks (with 3 to 12 processes); Uppaal performs the best but our enumerative algorithm is slightly behind. The symbolic algorithm does not scale, while PAT fails to terminate in all cases.

The tool used for the symbolic algorithm is open source and can be found at `https://github.com/osankur/symrob` along with all the benchmarks.

Conclusions The presented algorithm allowed performance gains in some benchmarks. However, it was not efficient in all instances. One can identify two main obstacles against general applicability. First, because the approach uses BDDs, the performance varies between instances depending on

whether the discrete part of the model is encoded efficiently or not. Unfortunately, due to the particular operations we use to compute successors on clock predicates, efficient forward image computation techniques such as [RAB+95] cannot be applied. One could study the adaptations of such techniques to our particular setting to improve the performance. The second obstacle is that the size of the transition relation is sensitive to the number of clock predicates, and the performance decreases quickly when many predicates are added. Better performance might be achievable by adapting implicit abstractions as used in [CGMT14].

Several extensions of our algorithms could be developed, *e.g.* combining our algorithms with other methods based on finer abstractions as in [HSW13], integrating predicate abstraction on discrete variables.

2.2.2 The Finite Automata Learning Approach

In this section, we introduce an approach with the goal of combining the advantages of both timed automata and finite-state model checkers. Our suggestion is to see the input model, without loss of generality, as a parallel composition between a finite-state machine $\mathcal{A}$, and a timed automaton $\mathcal{T}$. We specifically target instances where $\mathcal{A}$ is large, and $\mathcal{T}$ is relatively small but nontrivial. Note that this point of view was considered before in the verification of synchronous systems within a real-time environment [BCP+01]. As a novelty, for model checking, we apply a compositional reasoning rule on the product $\mathcal{A}\|\mathcal{T}$ by replacing the timed automaton $\mathcal{T}$ by a (small) deterministic finite automaton (DFA) H which represents the behaviors of $\mathcal{T}$. In order to select the DFA H, we adapt the algorithm [PGB+08] to our setting, and use a DFA learning algorithm (such as L* [Ang87], or TTT [IHS14]) to find an appropriate DFA either to prove the specification or to reveal a counterexanple.

Our approach enjoys the principle of *separation of concerns* in the following sense. A timed automaton model checker is used by the learning algorithm to answer *membership* and *equivalence queries*; these are answered without refering to $\mathcal{A}$, thus, by avoiding the large discrete state space. Therefore, the timed automaton model checker is used in this approach for what it is designed for: handling real-time constraints encoded in $\mathcal{T}$, not for dealing with excessive discrete state spaces. Once an appropriate DFA H is found by the learning algorithm, the system $\mathcal{A}\|H$ is model-checked using a finite-state model checker whose focus is to deal with large *discrete* state spaces. We can thus benefit from the best of the two worlds: a state-of-the-art model checker for timed automata, which is somewhat used here as a theory solver, and any finite-state model checker based on BDDs, SAT solvers, or even explicit-state enumeration.

We evaluate our algorithms in comparison with state-of-the-art tools and show that our approach is competitive with the existing tools, and allows model checking to scale to larger models in some cases. The approach thus offers an alternative treatment of timed models, which might be applied in other settings.

The results of this section appear in [S29].

Algorithm Our main motivation is to consider real-time systems that are modeled naturally as $\mathcal{A}\|\mathcal{T}$. Typically, $\mathcal{A}$ has a large (discrete) state space, and $\mathcal{T}$ is a relatively small timed automaton, but with potentially complex time constraints involving several clocks.

It should be clear however that any timed automaton $\mathcal{T}$ can be seen as such a product since both components can be defined on the same finite automaton underlying $\mathcal{T}$ with a proper synchronization mechanism, although such a translation would have little interest since $\mathcal{T}$ would have a large discrete

state space, which is what we precisely want to avoid. We are rather interested in modeling systems as products by separating the discrete state space from the real-time constraints.

The example of Figure 2.2 is given in this form. Here, the finite automaton is simple and only stores the mapping from machines to the tasks they are executing. Typically, if the machines or the processes executing tasks have internal states, these could be modeled in $\mathcal{A}$ as well without altering the timed automaton.

Our algorithm is an application of the assume-guarantee verification rule to timed automata given in the form of $\mathcal{A}\|\mathcal{T}$. The idea is to replace the timed automaton $\mathcal{T}$ by a finite-automaton overapproximation H, and obtain an overapproximation of the compound system in terms of untimed language. So if a linear property can be established on $\mathcal{A} \parallel H$ for an appropriate H, then the property also holds on the original system.

Let us present the above property as a verification rule. Assuming that we want to establish $\mathcal{A}\|\mathcal{T} \subseteq \mathsf{Spec}$ for some language Spec, we have

$$\frac{\mathcal{L}(\mathcal{T}) \subseteq \mathcal{L}(H) \quad \mathcal{L}(\mathcal{A}\|H) \subseteq \mathsf{Spec}}{\mathcal{L}(\mathcal{A}\|\mathcal{T}) \subseteq \mathsf{Spec.}}\ \mathsf{Asym} \tag{2.1}$$

Here, H serves as an *assumption* we make on $\mathcal{T}$ when verifying $\mathcal{A}$; so we can use H instead of $\mathcal{T}$ during model checking. The rule (2.1) is well known as the assume-guarantee verification rule [CHV+18], and has been used in model checking finite-state systems as well as timed automata [LAL+14]. The assumption H can either be provided by the user, or automatically computed using automata learning as in [PGB+08]. Intuitively, the model checking algorithm we present in this section is an application of [PGB+08] to our specific case.

Figure 2.5 presents the overview of the algorithm. The membership queries of the learning algorithm are answered by the membership oracle; the equivalence query with conjecture H is answered by the inclusion oracle. When the conjecture H passes the inclusion check, then we model check $\mathcal{H}\|\mathcal{A}$. When this is successful, we stop and declare that the original system $\mathcal{A}\|\mathcal{T}$ satisfies the specification. Otherwise, a counterexample $w \in \mathcal{L}(\mathcal{A}\|H) \setminus \mathsf{Spec}$ was found, and we use a realizability check to see whether $w \in \mathcal{L}(\mathcal{T})$ (this is actually done by the membership oracle). If the answer is yes, then the counterexample is confirmed, and we stop. Otherwise, we inform the learning algorithm that w must be excluded, and continue the learning process.

Experiments We built a prototype implementation of our algorithm in Scala, using the TTT automata learning algorithm [IHS14] from the learnlib library [IHS15], and the associated automatalib for manipulating finite automata. We used the TChecker [HP] model checker for implementing membership and inclusion oracles. For the latter, we complement H into H^c, and check the emptiness of the parallel composition of $\mathcal{T}$ with H^c. We use the nuXmv model checker for finite-state model checking.

Thus the overall input consists in an SMV file describing $\mathcal{A}$, and of a TChecker timed automaton describing $\mathcal{T}$. We use define expressions in SMV to define the synchronization labels Σ, while TChecker allows us to tag each transition with a label.

We compare our algorithm on a set of benchmarks with the model checkers Uppaal [BDL+06b] and nuXmv which has a timed automata model checker [CGM+19]. The former implements a zone-based enumerative algorithm, while the latter uses predicate abstraction through IC3IA which is good at handling large discrete state spaces.

The results of our experiments are given in Table 2.1. The leader election protocol is a distributed protocol that can recover from crashes [DGDF07], extended here with periodic activation times and

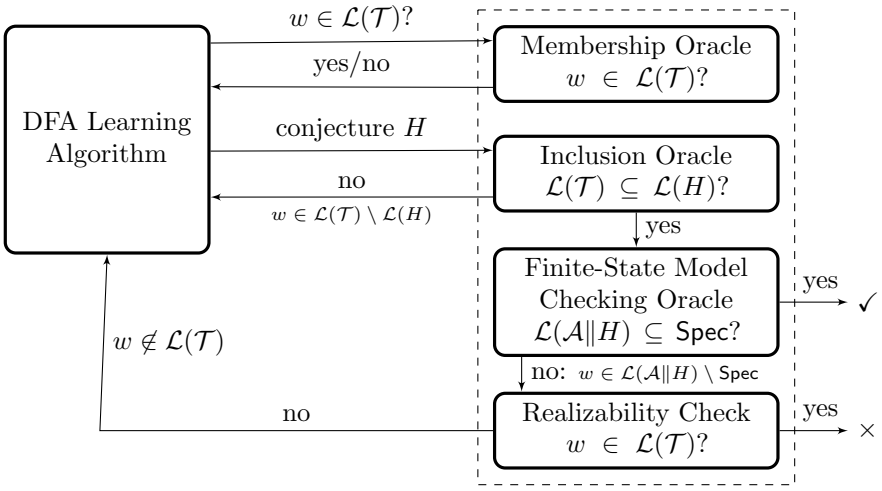


Figure 2.5: The learning-based compositional model checking algorithm. The box on the left is a DFA learning algorithm, while the oracles answering the queries of the learning algorithm are shown on the right and correspond to the teacher.

crash durations. The flooding time synchronization protocol (FTSP) is a leader election algorithm for multi-hop wireless sensor networks used for clock synchronization [MKSL04], and have been the subject of formal verification before [McI09, KA06]. We consider the abstract model used in [S31]. The STS benchmarks are programming logic controller models from [Die06] and are part of the TChecker benchmark database. The real-time broadcast protocol (rt-broadcast) implements a distributed algorithm made of $n = 3$ processes that wake up within a period interval, and stay active within a given time interval. If at least $m = 2$ of them are awake at the same time, they perform one step of a computation together and go back to sleep. The specification is whether a particular common configuration is reachable within a time bound. In the priority-based scheduling examples, a priority-based scheduling algorithm schedules tasks on a single machine. The interarrival times depend on the internal state of the processes which evolve over time.

Conclusions The results show overall that our algorithm is competitive with the state-of-the-art tools, and could solve instances that cannot be handled by these tools. Our algorithm is not uniformly better than the existing tools; since the learning process can sometimes be long, and a large overhead is observed *e.g.* when short counterexamples exist. In fact, an important limitation is due to learning being slow because of the alphabet size. Our setting could be extended to deal with large or symbolic alphabets *e.g.* [MM14, MM17]. Currently, we can only verify linear properties; one might verify branching-time properties by learning automata with a stronger notion of equivalence such as bisimulation.

2.2.3 Efficient Robustness Verification

In this section, we consider the problem of robustness verification for timed automata against safety properties. Robustness is understood here as the preservation of a safety specification when

Table 2.1: Benchmarks evaluating the compositional model checking algorithm with respect to Uppaal and nuXmv. The column #Clk is the number of clocks of the model; #C is the number of conjectures made by the DFA learning algorithm; #M is the number of membership queries; and |DFA| is the size of the final finite automaton learned. The safety specification holds on all models but those marked with *.

		Compositional				Uppaal	nuXmv
	#Clk	#C	#M	|DFA|	Time	Time	Time
Leader Election A	3	13	232	15	**157s**	—	—
Leader Election B	3	26	661	29	**198s**	—	—
Leader Election C	3	33	997	53	**149s**	—	-
Leader Election D	3				-	—	-
Leader Election (Stateless) A	3	13	232	15	15s	**6s**	—
Leader Election (Stateless) B	3	28	776	33	44s	**8s**	—
Leader Election (Stateless) C	3	33	997	53	17s	**6s**	-
Leader Election (Stateless) D *	3	134	6965	240	10m7s	**6s**	-
FTSP-abstract-2	2	3	54	8	**2s**	**2s**	-
FTSP-abstract-3	3	17	340	23	**47s**	7m8s	-
FTSP-abstract-4	4				-	-	-
STS-2	5				**7s**	19s	-
STS-3	6				-	-	-
Rt-broadcast A	4	49	1324	63	**59s**	-	87s
Rt-broadcast B	4	41	1100	63	101s	-	**90s**
Rt-broadcast C	4	21	590	39	**31s**	-	86s
Rt-broadcast D	4	27	901	52	**49s**	-	80s
Priority Scheduling 2 A	3	35	9859	49	34s	**1s**	7s
Priority Scheduling 2 B	3	29	1162	42	16s	—	**2s**
Priority Scheduling 3 C	4				-	—	**6s**
Priority Scheduling 3 D	4				-	—	**8s**
Priority Scheduling 3 E *	4				-	—	**11s**

the measure of time has bounded precision. This can be modeled in timed automata by *guard enlargement*, which consists in relaxing each guard of the form $a \leq x \leq b$ to $a - \delta \leq x \leq b + \delta$ where δ is a positive parameter.

Our goal is to decide if there exists $\delta > 0$ such that the timed automaton enlarged by δ satisfies its specification (Problem 1), and if this is the case, compute a safe upper bound on δ (Problem 2). We insist on the importance of both problems: while the first one decides the robustness of the model, the second one *quantifies* it by actually giving a bound under which the model is correct. This would allow one for instance to choose an appropriate hardware to implement the model [DDR05, AT05].

In this section, we present a symbolic procedure to simultaneously solve Problems 1 and 2 for general timed automata, based on [S28]. If the given model is robust, a safe upper bound on δ (which may not be the largest one) is output. The procedure is a semi-algorithm since we do not know whether it terminates; nevertheless, it did terminate on most of our experiments. It consists in a state-space exploration with an efficient parametric data structure which treats the enlargement δ as an unknown parameter, combined with an acceleration procedure for some of the cycles. While

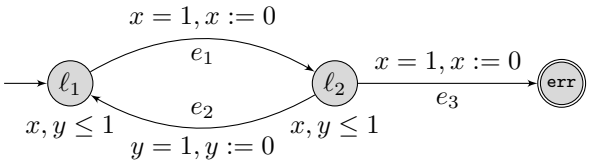


Figure 2.6: A timed automaton representing the two processes P_1 and P_2 instantiated with period $p = 1$, and a buffer size of 1. The guard under the locations are the invariants. The edge e_1 represents the arrival of a token in the buffer (period of 1) while e_2 represents process P_2 reading a token from the buffer. The error state is reached via e_3 if two tokens are pushed to the buffer without any read in between.
Without enlargement, any reachable state at location ℓ_2 satisfies $x = 0 \wedge y = 1$, so the error state is not reachable. Under enlargement by $\nu = \frac{1}{10}$, after the first transition, location ℓ_2 is reached by the set of states $1 - \nu \leq y \leq 1 + \nu \wedge 0 \leq x \leq 2\nu$ due to the enlargement of the guards and invariants. A simple calculation shows that the set of reachable states at location ℓ_2 after k cycles is $1 - (2k + 1)\nu \leq y \leq 1 + \nu \wedge 0 \leq x \leq 2k\nu \wedge 1 - (2k + 1)\nu \leq y - x \leq 1 + \nu$. Thus, for $k = 5$, we get $y \leq 1 + \nu \wedge x \leq 1 \wedge -\nu \leq y - x \leq 1 + \nu$, and $x = 1 \wedge y = 1$ is in this set, from which the error state is reachable.

the theoretical algorithm of [Pur00] is based on systematic acceleration of all encountered cycles, here, we do not systematically accelerate cycles, but rather adopt a "lazy" approach: during the exploration, when the accumulated imprecisions go beyond a threshold, we accelerate some cycles that may be responsible for this accumulation. This greatly reduces the computation overhead compared to a systematic acceleration of all cycles.

We ran experiments to evaluate the performance of our procedure. We compared our algorithm to a previously given one [KLMP14]; ours terminated faster in most cases, and sometimes with several orders of magnitude. To truly evaluate the gain of a parametric analysis, we also compared with a binary search on the values of δ using a model checker. Our procedure was often faster except against a low precision binary search (*i.e.* with few iterations).

Accumulation of Imprecisions In some timed automata even the smallest enlargement can lead to drastically different behaviors due to the accumulation of the imprecisions over long runs [Pur00]. As an example, consider the following simple problem. Two processes P_1, P_2 execute on different machines and communicate via a finite buffer. Every p time units, Process P_1 finishes a computation and pushes a token to the buffer; while P_2 reads a token from the buffer with the same period. We assume P_2 has an offset of p. The buffer will clearly not overflow in this system. However, assuming the slightest delay in the execution of P_2, or the slightest decrease in the execution time of P_1 leads to a buffer overflow since the delays will accumulate indefinitely. Figure 2.6 represents this system.

Background The formulation of Problem 1 has been studied starting with [Pur00, DDMR08] for safety properties, and extended to LTL and richer specifications, *e.g.* [BMR06, BMR08] using region-based techniques which cannot be applied efficiently. A symbolic zone-based algorithm was given in [DK06] for *flat* timed automata, that is, without nested cycles by applying *acceleration* on its cycles. Problem 2 has been answered in [JR11a] for flat timed automata, where the given algorithm computes the *largest* upper bound on δ satisfying the specification. The flatness is a

rather restrictive hypothesis since, for instance, it is easily violated when the system is obtained by composition of timed automata that contain cycles. Recently, a zone-based algorithm and a tool to solve Problem 1 for *general* timed automata was given [KLMP14]; but the algorithm does not compute any bound on δ. The latter algorithm is based, roughly, on extending the standard forward exploration of the state space augmented with the acceleration of all cycles encountered during the search, with some tricks to optimize the computations. In [LLTW11], refinements between interfaces are studied in a game-based framework including syntactic enlargement to account for imprecisions. In [SF07, SFK08] the authors use the fact that replacing all guards by closed ones allow one to verify *finite* paths (and the case of a periodic external synchronization) but this does not help in the analysis of the accumulation of imprecisions, nor can it allow one to compute a bound on δ.

Infinitesimally Enlarged DBMs We define *infinitesimally enlarged DBMs (IEDBM)*, a parameterized extension of DBMs, which we will use to explore the state space of enlarged timed automata. These were first defined in [DDMR08] to be used solely as a proof technique. Here, we extend this data structure with additional properties and explicit computations of the bounds on parameter δ, and show how it can be used to efficiently explore the state space.

We fix a clock set $\mathcal{C} \cup \{0\}$ including the 0 clock. An *infinitesimally enlarged DBM (IEDBM)* is a pair $(M, P)_{\langle 0,\delta_0\rangle}$ where M is a DBM and P is a $|\mathcal{C} \cup \{0\}| \times |\mathcal{C} \cup \{0\}|$ matrix over $\mathbb{N}$, called the *enlargement matrix*. The value $\delta_0 \in (0, \infty)$ is an upper bound on the unknown parameter δ. Intuitively, an IEDBM $(M, P)_{\langle 0,\delta_0\rangle}$ represents the *set* of DBMs $M + \nu P$ where $\nu \in [0, \delta_0)$. Figure 2.7 shows an example. We often see, abusively, an IEDBM as a matrix over pairs $(m, p) \in \mathbb{Z} \times \mathbb{N}$. The component (x, y) is denoted by $(M, P)_{\langle 0,\delta_0\rangle}[x, y]$. For simplicity, we always consider the half-open intervals of the form $[0, \delta_0)$ even though ν can be chosen equal to δ_0 in some cases. This is not a loss of generality since we are interested in the small values of ν.

We define the *width* of an IEDBM $(M, P)_{\langle 0,\delta_0\rangle}$ as the maximal value of the entries of P.

IEDBMs will allow us to reason on the parametric state space of enlarged timed automata "for small values of δ", which means that our computations on the data structure will hold for all $\nu \in [0, \delta_0)$, where $\delta_0 > 0$ is bound that is possibly updated to a smaller value after each operation. For instance, given sets $Z_1 = 1 \leq x \leq 2 + 3\nu$ and $Z_2 = x \leq 3$, for unknown δ, assume that we want to compute their intersection. We will write $Z_1 \cap Z_2 = 1 \leq x \leq 2 + 3\delta$ and chose $\delta_0 \leq \frac{1}{3}$. To make these simplifications, we need to compare pairs of IEDBM components in a similar spirit. For instance, to make the above simplification, we write $(2, 3)_{\langle 0,\frac{1}{3}\rangle} \leq (3, 0)_{\langle 0,\frac{1}{3}\rangle}$, which means that $2 + 3\nu \leq 3$ for all $\nu \in [0, \frac{1}{3})$.

$$\begin{array}{c} \\ 0 \\ x \\ y \end{array}\!\!\begin{array}{c} \begin{array}{ccc} 0 & x & y \end{array} \\ \begin{pmatrix} 0 & 1 & 1 \\ 3 & 0 & 2 \\ 2 & 1 & 0 \end{pmatrix} \end{array} + \delta \begin{pmatrix} 0 & 0 & 1 \\ 2 & 0 & 3 \\ 3 & 3 & 0 \end{pmatrix}$$

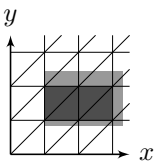


Figure 2.7: An IEDBM (above) representing the parametric set $1 \leq x \leq 3 + 2\delta \wedge 1 - \delta \leq y \leq 2 + 3\delta$. The set is represented (below) for $\delta = 0.15$.

The original robust reachability algorithm of [Pur00, DDMR08] consists in an exploration of the region graph, augmented with the addition of the images of all cycles *neighboring* reachable states. The idea is that when the guards are enlarged, these neighboring cycles become reachable, and they precisely capture all states that become reachable in the timed automaton for all values of δ. Thus, this algorithm computes the set $\cap_{\nu>0}\mathsf{reach}(\mathcal{T}_\nu)$, where $\mathsf{reach}(\mathcal{T}_\nu)$ denotes the states that are reachable in $\mathcal{T}_\nu$. A symbolic algorithm for this problem was given in [DK06] for *flat* timed automata, *i.e.*

Data: Timed automaton $\mathcal{T} = (L, \text{Inv}, \ell_0, \mathcal{C}, E)$, and target location ℓ_T.

```
1  Wait := {(ℓ0, Z0)_⟨∞⟩}, Passed := ∅, (ℓ0, Z0).K := K0;
2  while Wait ≠ ∅ do
3  |  (ℓ, Z) := pop(Wait), Add (ℓ, Z) to Passed;
4  |  if ℓ = ℓT then return Unsafe;
5  |  if width(Z) > (ℓ, Z).K then
6  |  |  Let π denote the prefix that ends in (ℓ, Z), along edges e1 e2 ... e_{|π|−1};
7  |  |  foreach cycle ρ = e_i e_{i+1} ... e_j do
8  |  |  |  if PPre*_ρ(⊤) ∩ π_i ≠ ∅ and ∀q ∈ Passed, PPost*_{(ρ)_δ}(⊤) ⋠ q then
9  |  |  |  |  Add PPost*_{(ρ)_δ}(⊤) as a successor to π_j, and to Wait;
10 |  |  |  end
11 |  |  end
12 |  |  if no fixpoint was added then (ℓ, Z).K = (ℓ, Z).K + K0 ;
13 |  foreach e ∈ E(ℓ) s.t. ∀q ∈ Passed, PExPost_{e_δ}((ℓ, Z)) ⋠ q do
14 |  |  (ℓ', Z') := PExPost_{e_δ}((ℓ, Z));
15 |  |  Add (ℓ', Z') to Wait;
16 |  |  (ℓ', Z').parent := (ℓ, Z);
17 |  |  (ℓ', Z').K := (ℓ, Z).K;
18 |  end
19 end
20 return Safe;
```

Algorithm 3: Symbolic robust safety semi-algorithm. Here (ℓ_0, Z_0) is the initial state symbolic state, and K_0 is a positive constant. We have two containers *Wait* and *Passed* storing symbolic states. The search tree is formed by assigning to each visited state (ℓ, Z) a parent denoted $(\ell, Z).parent$ (Line 16). We also associate to each symbolic state a bound K on width, denoted $(\ell, Z).K$.

without nested cycles, and later improved in [JR11a].

Let us fix a timed automaton $(L, \ell_0, \Sigma, \text{Inv}, \mathcal{C}, E)$, and a cycle ρ. Let $\mathsf{Pre}_\rho(Z)$ denote the predecessor of the zone Z along the cycle ρ; and let $\mathsf{Pre}^*_\rho(Z)$ the limit of the sequence $(\mathsf{Pre}^i_\rho(Z))_{i\geq 0}$. Let $\mathsf{Post}_{(\rho)_\nu}(Z)$ denote the successor of Z along the cycle ρ in which the guards are enlarged by ν; and $\mathsf{Post}^*_{(\rho)_\nu}(Z)$ the corresponding limit. $\top$ denotes the set of all valuations. A *progress cycle* of $\mathcal{T}$ is a cycle in which each clock is reset at least once.

The lemma below was proved in [JR11a].

Lemma 3. *Consider any zone Z and a progress cycle ρ of $\mathcal{T}$. If $\mathsf{Pre}^*_\rho(\top) \cap Z \neq \emptyset$, then starting from any state of $\mathsf{Pre}^*_\rho(\top) \cap Z$, for any $\nu > 0$, all states of $\mathsf{Post}^*_{(\rho)_\nu}(\top)$ are reachable in $\mathcal{T}_\nu$, by repeating ρ.*

As an example, consider Fig. 2.6. For the cycle $\rho = e_2 e_1$ that starts at ℓ_2, we have $\mathsf{Pre}^*_\rho(\top) = x, y \leq 1 \wedge x - y \leq 0$, and $\mathsf{Post}^*_{(\rho)_\nu}(\top) = x, y \leq 1 + \nu \wedge x - y \leq 0$. Since the point $(0, 1)$ is reachable and belongs to Pre^*_ρ, all states of $\mathsf{Post}^*_{(\rho)_\nu}(\top)$ are reachable, and in particular $(1, 1)$ from which the error state is reachable.

It is known that the above lemma does not hold for non-progress cycles; nevertheless, it was shown that in this case, $\mathsf{Post}^*_{(\rho)_\nu}(\top)$ is an *over-approximation* of the states reachable by repeating ρ under enlargement [S8]. Thus, the algorithm of [Pur00, DDMR08] may have false negatives (may answer "not robust" even though it is) but not false positives on timed automata with arbitrary cycles.

Algorithm Our procedure consists of a zone-based exploration with IEDBMs. Moreover, it uses an extension of the LU-extrapolation [BBLP06] for IEDBMs (not detailed here). It is easy to see that without cycle acceleration based on Lemma 3, an exploration based on IEDBMs alone does not terminate in general (see *e.g.* Fig. 2.6). To choose the cycles to accelerate, we adopt a lazy approach: we fix a bound K, and run the forward search using IEDBMs until the target is reached or some symbolic state has width greater than K. In the latter case, we examine the prefix of the current state, and accelerate its cycles. If no new state is obtained, then we increment the bound K for the current branch and continue the exploration. We thus interpret a large width as the accumulation of imprecisions due to cycles. No cycle may be responsible for a large width, in which case we increase the width threshold and continue the exploration.

Algorithm 3 is similar to a standard zone-based exploration with wait list and a passed list, e.g. [BY04]. We only explain the differences. In Line 5, we check whether the width of the visited zone is above the current threshold. If this is the case, then we collect all cycles along the branch from the root, and accelerate them. Here, `PPost` and `PPre` stand for parametric post and pre computation operators that work on IEDBMs. If no new zone was generated in this way, then we simply increment the threshold and continue. Line 13 corresponds to the expansion of the search tree. Here, `PExPost` stands for parametric extrapolated post operator, and $\preceq$ is the inclusion check. All predecessor and successor operations and inclusion checks may update the upper bound δ_0 in the handled IEDBMs.

If the algorithm returns Safe, then all operations are valid under the current bound δ_0, so the this is returned as a safe bound on δ.

Experimental Evaluation In this section, we evaluate the performance of our semi-algorithm on several benchmarks from the literature; most of which are available from `www.uppaal.org`, and have been considered in [KLMP14], with the exception of the scheduling tests (Sched *) which were constructed from the experiments of [GGL13]. We implemented Alg. 3 in OCaml in a tool called `Symrob` (*sym*bolic *rob*ustness, available from `https://github.com/osankur/symrob`). We consider two other competing algorithms: the first one is the previously published tool Verifix [KLMP14] which solves the infinitesimal robust safety problem but does not output any bound on δ. The second algorithm is our implementation of a binary search on the values of δ which iteratively calls an exact model checker until a given precision is reached.

The exact model checking algorithm is a forward exploration with DBMs using LU extrapolation and the inclusion test of [HKSW11] implemented in `Symrob`.

In Table 2.2, the number of visited symbolic states (as IEDBMs for `Symrob` and as DBMs for `Verifix`) and the running times are given. On most benchmarks `Symrob` terminated faster and visited less states. We also note that `Symrob` actually computed the *largest* δ below which safety holds for the benchmarks CSMA/CD and Fischer. One can indeed check that syntactically enlarging the guards by 1/3 (resp. 1/2) makes the respective classes of benchmarks unsafe (Recall that the upper bound δ_0 is always strict in IEDBMs). On one benchmark, `Verifix` wrongly classified the model as non-robust, which could be due to a bug or to the presence of non-progress cycles in the model (see [S8]).

Table 2.3 shows the performance of the binary search for varying precision $\epsilon \in \{\frac{1}{10}, \frac{1}{20}, \frac{1}{40}\}$. With precision $\frac{1}{10}$, the binary search was sometimes faster than `Symrob` (*e.g.* on CSMA/CD), and sometimes slower (*e.g.* Fischer); moreover, the computed value of δ was underestimated in some cases (*e.g.* CSMA/CD and Fischer benchmarks). With precision $\frac{1}{20}$, more precision was obtained on δ but at a cost of an execution time that is often worse than that of `Symrob` and systematically more

Table 2.2: Comparison between `Symrob` (breadth-first search, instantiated with $K_0 = 10$) and `Verifix` [KLMP14]. The running time of the exact model checking implemented in `Symrob` is given for reference in the column "Exact" (the specification was satisfied without enlargement in all models). Note that the visited number of states is not always proportional to the running time due to additional operations performed for acceleration in both cases. The experiments were performed on an Intel Xeon 2.67 GHz machine.

Benchmark	**Robust** – δ		**Visited States**		**Time**		
	Symrob	Verifix	Symrob	Verifix	Symrob	Verifix	Exact
CSMA/CD 9	Yes – 1/3	Yes	147,739	1,064,811	61s	294s	42s
CSMA/CD 10	Yes – 1/3	Yes	398,354	846,098	202s	276s	87s
CSMA/CD 11	Yes – 1/3	Yes	1,041,883	2,780,493	12m	26m	5m
Fischer 7	Yes – 1/2	Yes	35,029	81,600	11s	12s	6s
Fischer 8	Yes – 1/2	Yes	150,651	348,370	45s	240s	24s
Fischer 9	Yes – 1/2	Yes	627,199	1,447,313	4m	160m	2m20s
MutEx 3	Yes – 1000/11	Yes	37,369	984,305	3s	131s	3s
MutEx 4	No	No	195,709	146,893	16s	41s	4s
MutEx 4 fixed	Yes – 1/7	–	5,125,927	–	38m	>24h	7m
Lip Sync	–	No	–	29,647,533	>24h	14h	5s
Sched A	Yes – 1/4	**No***	9,217	16,995	11s	248s	2s
Sched B	No	–	50,383	–	105s	>24h	40s
Sched C	No	No	5,075	5,356	3s	29s	2s
Sched D	No	No	15,075	928	2s	0.5s	0.5s
Sched E	No	No	31,566	317	5s	0.5s	0.5s

Table 2.3: Performance of binary search where the initial enlargement is 8, and the required precision ϵ is either 1/10, 1/20 or 1/40. Note that when the model is not robust, the binary search is inconclusive. Nonetheless, in these cases, we do know that the model is unsafe for the smallest δ for which we model-checked the model. In these experiments the choice of the initial condition (here, $\delta = 8$) wasn't significant since the first iterations always took negligeable time compared to the case $\delta < 1$.

Benchmark	**Robust** – δ		**Visited States**		**Time**		
	$\epsilon = 1/10$	$\epsilon = 1/20$	$\epsilon = 1/10$	$\epsilon = 1/20$	$\epsilon = 1/10$	$\epsilon = 1/20$	$\epsilon = 1/40$
CSMA/CD 9	Yes – 1/4	Yes – 5/16	151,366	301,754	43s	85s	123s
CSMA/CD 10	Yes – 1/4	Yes – 5/16	399,359	797,914	142s	290s	428s
CSMA/CD 11	Yes – 1/4	Yes – 5/16	1,043,098	2,085,224	8m20s	17m	26m
Fischer 7	Yes – 3/8	Yes – 7/16	75,983	111,012	15s	21s	31s
Fischer 8	Yes – 3/8	Yes – 7/16	311,512	462,163	53s	80s	129s
Fischer 9	Yes – 3/8	Yes – 7/16	1,271,193	1,898,392	5m	7m30s	12m
MutEx 3	Yes – 8	Yes – 8	37,369	37,369	2s	2s	2s
MutEx 4	Inconclusive		1,369,963	1,565,572	1m5s	1m15s	1m30s
MutEx 4 fix'd	Yes – 5/8	Yes – 9/16	6,394,419	9,864,904	9m30s	17m	25m
Lip Sync	Inconclusive		–	–	>24h	>24h	>24h
Sched A	Yes – 7/16	Yes – 15/32	27,820	37,101	6s	9s	11s
Sched B	Inconclusive		109,478	336,394	35s	140s	20m
Sched C	Inconclusive		10,813	36,646	2s	6s	56s
Sched D	Inconclusive		27,312	182,676	2s	9s	60s
Sched E	Inconclusive		98,168	358,027	6s	17s	95s

states to visit. Increasing the precision to $\frac{1}{40}$ leads to even longer execution times. On non-robust models, a low precision analysis is often fast, but since the result is inconclusive, one rather increases the precision, leading to high execution times. The binary search can be made complete by choosing the precision exponentially small [S8] but this is too costly in practice.

Conclusion We presented a symbolic procedure to solve the quantitative robust safety problem for timed automata based on infinitesimally enlarged DBMs. A good performance is obtained thanks to the abstraction operators we lifted to the parametric setting, and to the lazy approach used to accelerate cycles. Although no termination guarantee is given, we were able to treat several benchmarks from the literature, demonstrating the feasability of robustness verification, and the running time was often comparable to that of exact model checking. Our experiments show that binary search is often fast if run with low precision; however, as precision is increased the gain of a parametric analysis becomes clear. Thus, both approaches might be considered depending on the given model.

An improvement over binary search for a problem of refinement in timed games is reported in [LLTW11]; this might be extended to our problem as well. Both our tool and Verifix fail when a large number of cycles needs to be accelerated, and this is difficult to predict. An improvement could be obtained by combining our lazy acceleration technique using the combined computation of the cycles of [KLMP14]. An extension to LTL objectives could be possible using [BMR06].

2.3 Controller Synthesis Algorithms

2.3.1 Definitions

Games. A *finite safety game* is a pair $(\mathcal{G}, \mathsf{Bad})$ where $\mathcal{G}$ is an LTS $(Q_E \cup Q_C, q_0, \Sigma, T)$ with the set of states given as a partition $Q_E \cup Q_C$, namely, *Environment states* (Q_E), and *Controller states* (Q_C), and $Bad \subseteq Q_E \cup Q_C a$ is an *objective*. The game is played between two players, namely, *Controller* and *Environment*. At each state $q \in Q_C$, Controller determines the successor by choosing an edge from q, and Environment determines the successor from states $q \in Q_E$. A *strategy* for Controller (resp. Environment) maps finite runs of $(Q_E \cup Q_C, q_0, \Sigma, T)$ ending in Q_C (resp. Q_E) to an edge leaving the last state. A pair of strategies, one for each player, induces a unique infinite run from the initial state. A run is *winning* for Controller if it does not visit Bad; it is winning for Environment otherwise. A *winning strategy* for Controller is such that for all Environment strategies, the run induced by the two strategies is winning for Controller. Symmetrically, Environment has a winning strategy if for all Controller strategies, the induced run is winning. A strategy is *positional* if it only depends on the last state of the given run.

The parallel composition of $(\mathcal{G}, \mathsf{Bad})$ and a deterministic finite automaton $\mathcal{F} = (Q', q'_0, \Sigma, T', F)$ on alphabet Σ is a new game whose LTS is $\mathcal{G} \| \mathcal{F}$ in which the Controller states are $Q_C \times Q'$, the Environment states are $Q_E \times Q'$, and the objective is $\mathsf{Bad} \times F$.

Finite games were extended to the real-time setting as *timed games* [MPS95, AMP95]. A *timed game* is a timed automaton $\mathcal{T} = (L_E \cup L_C, \ell_0, \Sigma, \mathrm{Inv}, \mathcal{C}, E, \mathsf{Bad})$ with the exception that its edges are labeled by $\Sigma \cup \{\epsilon\}$ (and not just by Σ as in the previous section), and the locations are partitioned as $L_E \cup L_C$ into *Environment locations* and *Controller locations*. The semantics is defined by letting Environment choose the delay and the edge to be taken at locations L_E, while Controller choose these from L_C. Formally, a *strategy* for Environment (resp. Controller) is a function which associates

a run that ends in L_E (resp. L_C) to a pair of delay and an edge enabled from the state reached after the delay. A run is winning for Controller if it does not visit Bad. A Controller (resp. Environment) strategy is *winning* for objective Bad if for all Environment (resp. Controller) strategies, the induced run from the initial state is winning (resp. not winning) for Controller. A run r is *compatible* with a strategy σ for Controller (resp. Environment) if there exists an Environment (resp. Controller) strategy σ' such that r is induced by σ, σ'.

The parallel composition of a finite safety game $(\mathcal{G}, \mathsf{Bad})$ and a timed automaton $\mathcal{T} = (L, \ell_0, \Sigma, \mathrm{Inv}, \mathcal{C}, E, F)$ on common alphabet Σ is the timed game $\mathcal{G} \| \mathcal{T}$ where Controller locations are $Q_C \times L$, and Environment locations are $Q_E \times L$.

2.3.2 The Finite Automaton Approach for Synthesis

This section presents the application of the finite automaton learning approach of Section 2.2.2 to the controller synthesis problem with timed games. The algorithm presented here is more involved than that for model checking.

Our objective was to find a way to exploit efficient finite-state game solvers [S20][JPA+22] in the context of timed automata even if this meant having an incomplete algorithm. We describe a setting where a one-sided abstraction is applied for controller synthesis by replacing the timed automaton component by a learned DFA. Contrarily to the model checking algorithm, our controller synthesis algorithm is sound but not complete, that is, the algorithm may fail although there exists a control strategy, while any control strategy that is output is correct. More precisely, we consider timed games in the form $\mathcal{G} \| \mathcal{T}$ where $\mathcal{G}$ is a finite-state game, and $\mathcal{T}$ is a timed automaton. We describe an algorithm that alternates between two phases. In the first phase, the goal is to find a DFA $\overline{H}$ that is an overapproximation of $\mathcal{T}$. Once this is found, we use a finite-state game solver on $\mathcal{G} \| \overline{H}$; if there is a control strategy, then we show how this strategy can be applied in the original system $\mathcal{G} \| \mathcal{T}$. If not, then we obtain a counterstrategy σ. We then switch to the second phase whose goal is to check whether the counterstrategy is spurious or not; and it does so by learning an underapproximation DFA $\underline{H}$ of $\mathcal{T}$, and checking whether σ induces runs that are all in $\underline{H}$. Accordingly, we either reject the instance or switch back to the first phase. As in the model checking algorithm, the timed automaton model checker is only used to answer queries independently from $\mathcal{G}$, and a finite-state game solver and a model checker are used to compute and analyze strategies in a discrete state-space.

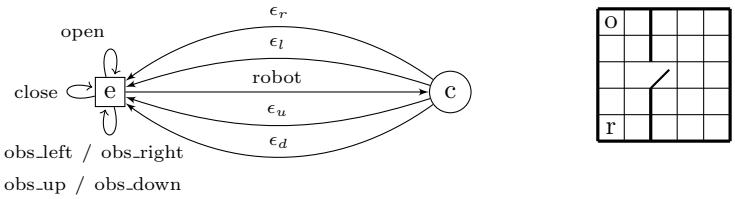
Target Timed Game Instances. Following our presentation in the model checking section, we consider controller synthesis problems described as timed games in the form of $(\mathcal{G} \| \mathcal{T}, \mathsf{Bad} \times F)$ where $(\mathcal{G}, \mathsf{Bad})$ is a finite safety game, and $\mathcal{T}$ is a timed automaton. In addition, we assume that $\mathcal{G} \| \mathcal{T}$ is *Controller-silent*, defined as follows.

Definition 4. *The timed game $(\mathcal{G} \| \mathcal{T}, \mathsf{Bad} \times F)$ on alphabet Σ is Controller-silent if 1) all Controller transitions are silent; and 2) all Controller locations in $\mathcal{T}$ are* urgent*, that is, an invariant ensures that no time can elapse.*

Hence, we again separate the game $\mathcal{G}$ defined on a possibly large discrete state space while real-time constraints are separately given in $\mathcal{T}$.

The intuition behind the semantics is the following: because the game is played in $\mathcal{G} \| \mathcal{T}$ and $\mathcal{G}$ is Controller-silent, the timed automaton model $\mathcal{T}$ is only used to disallow some of the Environment transitions according to real-time constraints, while Controller's actions are instantaneous responses to Environment's actions and thus are unaffected by the contraints of $\mathcal{T}$. One can think of the

Finite game:



Timed automaton:

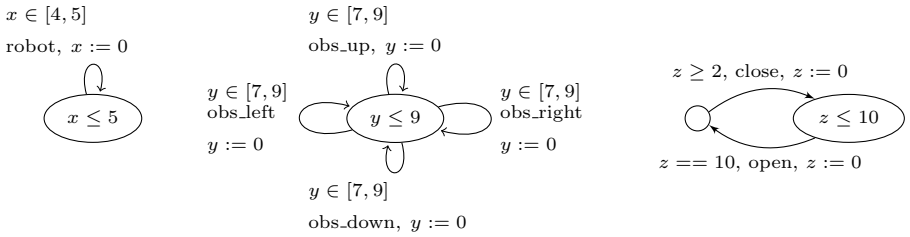


Figure 2.8: A timed game $\mathcal{G}\|\mathcal{T}$ modelling a planning problem. The finite game models a robot and an obstacle moving in a grid world as shown on top right. The cells r and o show, respectively, the initial positions of the robot and the obstacle. The robot cannot cross walls (shown in thick segments), and can only cross the door if it is open.

timed automaton as some form of scheduler that schedules uncontrollable events in the system, so the order of these is determined by Environment. This asymetric view will enable a one-sided abstraction framework which we present in the next section, where Environment transitions are approximated by a regular language.

An example is given in Figure 2.8. The finite game drawn here only shows the structure of the game. It has, in addition, integer position variables `rob_x, rob_y, obs_x, obs_y`, and a boolean variable `door` to encode the full state space. The state e belongs to Environment, which can move the obstacle in any direction, close or open the door, or let the robot move by going to state c. The state c belongs to Controller. All its leaving transitions are silent, and correspond to moving the robot in four directions. These transitions have preconditions, not shown in the figure, that check whether the moves are possible, and have updates that modify the state variables. The timed automaton, given as a network of three timed automata, determine the timings of these events. One can notice, for example, that the robot is moving faster than the obstacle, and that whenever the door is closed, it remains so for 10 time units.

One-Sided Abstraction We consider timed games with the following restriction

We show that by replacing $\mathcal{T}$ by a DFA H that is an overapproximation, we obtain an abstract game in which Controller strategies can be transferred to the original game. This is formalized in the next lemma (the proof is in the appendix).

Lemma 5. *Consider a Controller-silent timed game $(\mathcal{G}\|\mathcal{T}, \mathsf{Bad}\times F)$, and a complete DFA H with accepting states F_H, satisfying $\mathcal{L}(\mathcal{T}) \subseteq \mathcal{L}(H)$.*

- *If Controller wins $(\mathcal{G}\|H, \mathsf{Bad}\times F_H)$, then it wins $(\mathcal{G}\|\mathcal{T}, \mathsf{Bad}\times F)$.*

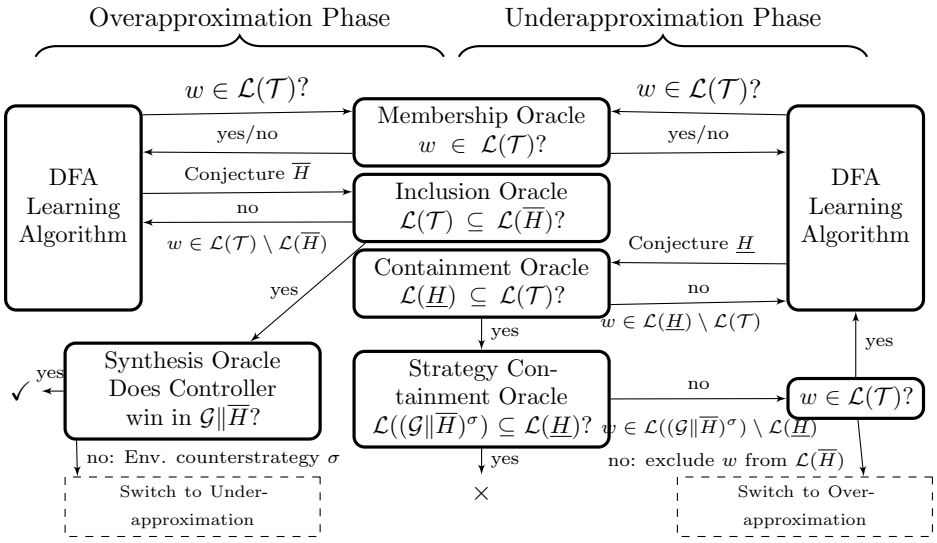


Figure 2.9: The learning-based compositional controller synthesis algorithm for the input timed game $\mathcal{G}\|\mathcal{T}$, with $\mathcal{G}$ a Controller-silent finite game, and $\mathcal{T}$ a label-deterministic timed automaton. Two automata learning algorithms run in parallel to learn under- and over-approximations $\underline{H}$ and $\overline{H}$ such that $\underline{H} \subseteq \mathcal{L}(\mathcal{T}) \subseteq \overline{H}$.

- *If Environment wins* $(\mathcal{G}\|\mathcal{T}, \mathsf{Bad}\times F)$*, then it wins* $(\mathcal{G}\|H, \mathsf{Bad}\times F_H)$*, and has a strategy in* $(\mathcal{G}\|H, \mathsf{Bad}\times F_H)$ *whose all compatible runs have traces in* $\mathcal{L}(\mathcal{T})$*.*

Note that in the above lemma, it is crucial that the game is Controller-silent. In fact, if Controller could take edges that synchronize with $\mathcal{T}$, then we may not be able to apply a strategy in $\mathcal{G}\|H$ to $\mathcal{G}\|\mathcal{T}$, since such a strategy may prescribe traces tha are not accepted in $\mathcal{T}$. Moreover, if Controller locations are not urgent, we would not know how to select the delays when mapping the strategy to $\mathcal{G}\|\mathcal{T}$.

Algorithm The algorithm is described in Figure 2.9.

The objective of the overapproximation phase is to attempt to learn a DFA $\overline{H}$ satisfying $\mathcal{L}(\mathcal{T}) \subseteq \overline{H}$, and such that Controller wins in $\mathcal{G}\|\overline{H}$. Once such a candidate DFA $\overline{H}$ is found, the synthesis oracle checks, using finite-state techniques, whether Controller has a winning strategy in $\mathcal{G}\|\overline{H}$. If this is the case, we stop and conclude that Controller wins in $\mathcal{G}\|\mathcal{T}$. Otherwise, Environment has a winning strategy σ in this game; and we switch to the underapproximation phase.

The goal of the underapproximation is to check whether the given Environment strategy σ can be proved to be spurious. Intuitively, we would like to check whether $\mathcal{L}((\mathcal{G}\|\overline{H})^\sigma) \subseteq \mathcal{L}(\mathcal{T})$ and reject if this is the case. We know that a winning Environment strategy in $\mathcal{G}\|\mathcal{T}$ implies that there is such a strategy σ. This is the source of incompleteness of our algorithm, since this condition is necessary but not sufficient for Environment to win; that is, the condition does not guarantee that Environment actually wins in $\mathcal{G}\|\mathcal{T}$.

While $\mathcal{L}((\mathcal{G}\|\overline{H})^\sigma) \subseteq \mathcal{L}(\mathcal{T})$ can be checked with a timed automaton model checker, this would mean exploring the large state space due to $\mathcal{G}$. Since we want to avoid using timed automata model checkers

on such large instances, we rather learn an underapproximation $\underline{H}$ of $\mathcal{L}(\mathcal{T})$ using the membership and containment oracles, and use a finite-state model checker to check $\mathcal{L}((\mathcal{G}\|\overline{H})^\sigma) \subseteq \mathcal{L}(\underline{H})$, which can be done with finite-state techniques. If this check passes, then we reject the instance. Otherwise, some trace w appears in $\mathcal{L}((\mathcal{G}\|\overline{H})^\sigma)$ but not in $\mathcal{L}(\underline{H})$. If $w \in \mathcal{L}(\mathcal{T})$, then we require that w be included in $\underline{H}$, and continue the learning process. Otherwise, σ is not valid since it induces w which is not in $\mathcal{L}(\mathcal{T})$. So we interrupt the current phase and switch back to the overapproximation phase requiring w to be removed from $\overline{H}$.

Membership and inclusion oracles are implemented with a timed automata model checker. Here, the synthesis oracle can be any finite game solver; we just need the capability of computing the controlled system $(\mathcal{G}\|\overline{H})^\sigma$. Such a system is finite-state, so the strategy containment oracle can be implemented using a finite-state model checker (since $\underline{H}$ is deterministic and can thus be complemented).

To implement the containment oracle, one can use testing such as the Wp-method [LvBP94] to establish the containment, as it is customary in DFA learning. In this case, the answer is approximate in the sense that the conformance test can fail to detect that containment does not hold. However, this does not affect the soundness of the overall algorithm since it can only increase false negatives for the controller synthesis problem: whenever a Controller strategy is returned, it is indeed a winning strategy for $\mathcal{G}\|\mathcal{T}$.

Experiments Our tool accepts instances $\mathcal{G}\|\mathcal{T}$ where $\mathcal{G}$ is given as a Verilog module, and $\mathcal{T}$ as a TChecker timed automaton. Some of the inputs of the Verilog module are *uncontrollable* (chosen by Environment), some others are controllable (chosen by Controller). We use outputs of the Verilog module to define the synchronization labels Σ; while TChecker models tag each transition with such a label.

Table 2.4: The results of the controller synthesis experiments. The columns #Clks, #C, #M respectively show the number of clocks in the model, the numbers of conjectures and membership queries made by the compositional algorithm; while $|\overline{H}|$, $|\underline{H}|$ show the sizes of the DFAs learned by the two phases.

		Compositional Algorithm					Uppaal TIGA	Controllable
	#Clks	#C	#M	$\lvert\overline{H}\rvert$	$\lvert\underline{H}\rvert$	Time	Time	
Scheduling genbuf A	3	50	2178	114		**26s**	—	yes
Scheduling genbuf B	3	40	1734	96		**15s**	—	yes
Scheduling genbuf C	3	45	1503	88		**4s**	—	yes
Scheduling counter64 D	3	54	2098	108		26s	**14s**	yes
Scheduling counter64 E	3	37	1454	83		**16s**	19s	yes
Scheduling counter64 F	3	19	21391	19	19	89s	**0s**	no
Planning genbuf A	2	2	17	4		**6s**	—	yes
Planning genbuf B	2	2	24	5		**9s**	—	yes
Planning genbuf C	2	9	1156	5	5	**266s**	—	no
Planning stateless D	2	3	50	9		**2s**	22s	yes
Planning stateless E	2	2	17	4		**2s**	4s	yes
Planning stateless F	2	8	973	5	5	10s	**2s**	no

Membership, inclusion, and containment queries are answered by TChecker. For the synthesis

oracle, we used the game solver Abssynthe [S13]. We use berkeley-abc and yosys to translate Verilog modules to AIG circuits which is the input format of Abssynthe. Abssynthe is able to compute the winning strategy σ for the winning player; it also computes the system controlled by σ in this case as an AIG circuit. The strategy containment oracle is implemented using nuXmv; since $\underline{H}$ is deterministic, one can complement it, and check whether the intersection with $(\mathcal{G}\|\overline{H})^\sigma$ is empty.

The tool uses two Java threads to implement both learning phases, which are interrupted and continued while switching phases. Note that the very first learning step of $\overline{H}$ and $\underline{H}$ can be parallelized since the first underapproximation conjecture $\underline{H}$ does not depend on σ.

We evaluate our algorithm with two classes of benchmarks. The only tool to which we compare is Uppaal-TIGA [BCD$^+$07a] since Synthia [PEM11] is not available anymore, and we are not aware of any other timed game solver.

In the scheduling benchmarks, there are two sporadic tasks that arrive nondeterministically, but constrained by the timed automaton model. The controller must schedule these using two machines. Each machine has an internal state, modeled here either by a simple 6-bit counter, or by a `genbuf` circuit from the SYNTCOMP database. The scheduling duration depends on the internal state: some states require executing two external tasks, some others require executing three. The external task has a nondeterministic duration constrained by the timed automaton model. The internal states of the machines change each time they finalize a task. The controller loses if all machines are busy upon the arrival of a new task, or if it schedules a task on a busy machine. Uppaal TIGA was able to solve the counter models since they induce a smaller state space, but failed at the genbuf models. The compositional algorithm could efficiently handle these models thanks to the powerful BDD-based algorithm of Abssynthe. Notice that Uppaal was generally able to determine very quickly when the model is not controllable by finding a small counterstrategy, while the compositional algorithm had a large overhead: it had to learn $\overline{H}$ and $\underline{H}$ before it can find and check the counterexample.

In the planning benchmarks, a robot and an obstacle is moving in a 6×6 grid (or 9×9 for the stateless case). Each agent can decide to move to an adjacent cell when they are scheduled, and the scheduling times are determined by a timed automaton. The goal of the robot is to avoid the obstacles. In the genbuf case, there are moreover internal states that can cause a glitch and prevent the agents from performing their moves, depending on their states. Uppaal TIGA was not able to manage the large state space unlike the compositional algorithm in this case, but both were able to solve the stateless case.

Conclusion Our algorithm is able to synthesize controllers for timed games with large discrete state spaces and real-time constraints. In our experiments, we considered a modest number of clocks which still define non-trivial behaviors.

Our setting is currently restricted by the abstractions we use since when the algorithm rejects the instace, we cannot conclude whether the system is controllable or not. Using both the under- and overapproximations within the finite-state synthesis, for instance, using a three-valued abstraction approach [dR10], might allow us to render the approach complete.

2.3.3 Robust Controller Synthesis

The semantics of timed automata is a mathematical idealization: it assumes that clocks have infinite precision and instantaneous actions. Proving that a timed automaton satisfies a property does not

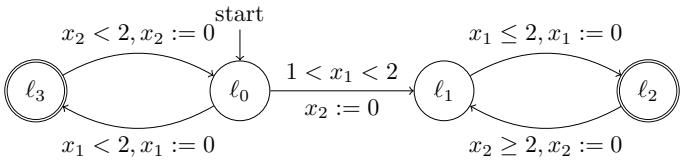


Figure 2.10: A timed automaton

ensure that a real implementation of it also does. This *robustness* issue is a challenging problem for embedded systems [HS06] and alternative semantics have been proposed, so as to ensure that the verified (or synthesized) behavior remains correct in presence of small timing perturbations.

We are interested in the controller synthesis problem in timed automata with a Büchi acceptance condition, which consists in finding an accepting infinite execution. The role of the controller here is to choose transitions and delays. This problem has been extensively studied in the exact setting (*e.g.* [HSTW20]). In the context of robustness, our goal is to distinguish timed automata where the Büchi condition can be satisfied even when the chosen delays are systematically perturbed by an adversary by a bounded amount. In fact, [CHR02] shows that in some timed automata models, an infinite run requires increasing precision at each step, which means that the behaviour may not be realizable by any controller in a real system.

More formally, the semantics we consider is defined as a game that depends on some parameter δ representing an upper bound on the amplitude of the perturbation. In this game, the controller plays against an antagonistic environment that can perturb each delay using a value chosen in the interval $[-\delta, \delta]$. We require the controller to choose delays such that the guard is satisfied uder all possible perturbations in $[-\delta, \delta]$. That is, we do not enlarge the guards as in Section 2.2.3.

The case of a fixed value of δ has been shown to be decidable in [CHP11], and also for a related model in [LLTW14]. However, these algorithms are based on regions, and as the value of δ may be very different from the constants appearing in the guards of the automaton, do not yield practical algorithms. Moreover, the maximal perturbation is not necessarily known in advance, and could be considered as part of the design process.

We are interested in determining whether for some positive value of δ, the controller wins the game. It was proven in [S30] that this problem is in PSPACE-complete, thus no harder than in the exact setting with no perturbation allowed.

Consider the timed automaton in Fig. 2.10. The controller has a winning strategy in $\mathcal{T}$ for all $\delta < 1/2$. Indeed, he can follow the cycle $\ell_0 \to \ell_3 \to \ell_0$ by always picking time delay $1/2$ so that, when arriving in ℓ_3 (resp. ℓ_0) after the perturbation of the environment, clock x_2 (resp. x_1) has a valuation in $[1/2 - \delta, 1/2 + \delta]$. Therefore, he can play forever following this memoryless strategy. For $\delta \geq 1/2$, the environment can enforce reaching ℓ_3 with a value for x_2 at least equal to 1. The guard $x_2 < 2$ of the next transition to ℓ_0 cannot be guaranteed, and therefore the controller cannot win. In [S30], it is shown that the cycle around ℓ_2 does not provide a winning strategy for the controller for any value of $\delta > 0$ since perturbations accumulate so that the controller can only play it a finite number of times in the worst case.

The algorithm of [S30] heavily relies on regions, and more precisely on *orbit graphs*, which are an abstractions that refine regions. Hence, it is not at all amenable to implementation. Our objective is to provide an efficient symbolic algorithm for solving this problem. To this end, we target the use of *zones* instead of regions, as they allow an on-demand partitioning of the state space. Moreover,

the algorithm we develop explores the reachable state-space in a *forward* manner. This is known to lead to better performances, as witnessed by the successful tool UPPAAL TIGA based on forward algorithms for solving controller synthesis problems [CDF+05a].

Our algorithm can be understood as an adaptation to the robustness setting of the standard algorithm for Büchi acceptance in timed automata [LOD+13]. This algorithm looks for an accepting lasso using a double depth-first search. A major difficulty consists in checking whether a lasso can be robustly iterated, i.e. whether there exists $\delta > 0$ such that the controller can follow the cycle for an infinite amount of steps while being tolerant to perturbations of amplitude at most δ. The key argument of [S30] was the notion of aperiodic folded orbit graph of a path in the region automaton, thus tightly connected to regions. Lifting this notion to zones seems impossible as it makes an important use of the fact that valuations in regions are time-abstract bisimilar, which is not the case for zones.

Our contributions are threefold.

- First, we provide a polynomial time procedure to decide, given a lasso, whether it can be robustly iterated. This symbolic algorithm relies on a computation of the greatest fixpoint of the operator describing the set of controllable predecessors of a path. In order to provide an argument of termination for this computation, we resort to a new notion of branching constraint graphs, extending the approach used in [JR11b, Tra16] and based on constraint graphs to check iterability of a cycle, without robustness requirements.

- Second, we show that when considering a lasso, not only can we decide robust iterability, but we can even compute the largest perturbation under which it is controllable. This problem was not known to be decidable before.

- Finally, we provide a termination criterion for the analysis of lassos. Focusing on zones is not complete: it can be the case that two cycles lead to the same zones, but one is robustly iterable while the other one is not. Robust iterability crucially depends on the real-time dynamics of the cycle and we prove that it actually only depends on the reachability relation of the path. We provide a polynomial-time algorithm for checking inclusion between reachability relations of paths in timed automata based on constraint graphs.

All our procedures can be implemented using difference bound matrices, a very efficient data structure used for timed systems. These developments were integrated in the tool TChecker [HP], and we present a case study of a train regulation network illustrating its performances.

Case Study To illustrate our approach, we present a case study on the regulation of train networks. Urban train networks in big cities are often particularly busy during rush hours: trains run in high frequency so even small delays due to incidents or passenger misbehavior can perturb the traffic and end up causing large delays. Train companies thus apply regulation techniques: they slow down or accelerate trains, and modify waiting times in order to make sure that the traffic is fluid along the network. Computing robust schedules with provable guarantees is a difficult problem.

We study here a simplified model of a train network and aim at automatically synthesizing a controller that regulates the network despite perturbations, in order to ensure performance measures on total travel time for each train. Consider a circular train network with m stations $s_0, \ldots, s_{m-1}$ and n trains. We require that all trains are at distinct stations at all times. There is an interval of delays $[\ell_i, u_i]$ attached to each station which bounds the travel time from s_i to $s_{i+1 \bmod m}$. Here the lower bound comes from physical limits (maximal allowed speed, and travel distance) while

the upper bound comes from operator specification (e.g. it is not desirable for a train to remain at station for more than 3 minutes). The objective of each train i is to cycle on the network while completing each tour within a given time interval $[t_1^i, t_2^i]$.

All timing requirements are naturally encoded with clocks. Given a model, we solve the robust controller synthesis problem in order to find a controller choosing travel times for all trains ensuring a Büchi condition (visiting s_1 infinitely often). Given the fact that trains cannot be at the same station at any given time, it suffices to state the Büchi condition only for one train, since its satisfaction of the condition necessarily implies that of all other trains.

Scenario	m	n	#Clocks	robust?	time
A	6	2	4	yes	4s
B	6	2	4	no	2s
C	6	3	5	no	263s
D	6	3	4	yes	125s
E	6	4	2	yes	53s
F	6	4	2	yes	424s
G	6	4	8		TO
H	6	4	8		TO
I	20	2	2	yes	76s
J	20	2	2	yes	55s
K	30	2	2	yes	579s

Figure 2.11: Summary of experiments with different sizes. In each scenario, we assign a different objective to a subset of trains. The answer is *yes* if if a robust controller was found, *no* if none exists. TO stands for a time-out of 30 minutes.

Let us present two representative instances and then comment the performance of the algorithm on a set of instances. Consider a network with two trains and m stations, with $[\ell_i, u_i] = [200, 400]$ for each station i, and the objective of both trains is the interval $[250 \cdot m, 350 \cdot m]$, that is, an average travel time between stations that lies in $[250, 350]$. The algorithm finds an accepting lasso: intuitively, by choosing δ small enough so that $m\delta < 50$, perturbations do not accumulate too much and the controller can always choose delays for both trains and satisfy the constraints. This case corresponds to scenario A in Figure 2.11. Consider now the same network but with two different objectives: $[0, 300 \cdot m]$ and $[300 \cdot m, \infty)$. Thus, one train needs to complete each cycle in at most $300 \cdot m$ time units, while the other one in at least $300 \cdot m$ time units. A classical Büchi emptiness check reveals the existence of an accepting lasso: it suffices to move each train in exactly 300 time units between each station. This controller can even recover from perturbations for a bounded number of cycles: for instance, if a train arrives late at a station, the next travel time can be chosen smaller than 300. However, such corrections will cause the distance between the two trains to decrease and if such perturbations happen regularly, the system will eventually enter a deadlock. Our algorithm detects that there is no robust controller for the Büchi objective. This corresponds to the scenario B in Figure 2.11.

Figure 2.11 summarizes the outcome of our prototype implementation on other scenarios. The tool was run on a 3.2Ghz Intel i7 processor running Linux, with a 30 minute time out and 2GB of memory. The performance is sensitive to the number of clocks: on scenarios with 8 clocks the algorithm ran out of time.

Case of Random Perturbations In the above results, we adopted a worst-case view assuming that the perturbations were chosen adversarially at each step. One could be critical about this view since such perturbations are random in many situations, and it is highly unlikely that a sequence of perturbations is chosen in a very particular manner in order to make the system fail. Thus, an interesting question is whether the robustness questions have different answers assuming a stochastic model of perturbations rather than adversarial one.

We investigated these questions in [S23] where the Buchi emptiness problem was studied in the presence of perturbations follow a probability distribuion (such as the uniform distribution) among the perturbation set $[-\delta, \delta]$, and assuming perturbations at each step are independent. We prove that in such a setting robust controllers achieving a Buchi condition are precisely those achieving the Buchi condition in the adversarial perturbation model. This is a strong result showing that controllers that are not robust will eventually fail whether the perturbations are chosen by an adversary with computational power or just random noise.

Conclusions The presented case study illustrates the application of robust controller synthesis in small or moderate size problems. Our prototype relies on the DBM libraries that we use with twice as many clocks to store the constraints of the normalised constraint graphs. Our algorithms must be further extended in order to scale to larger models. First, better representations for specific constraints we use can be an interesting direction. Second, we plan to study extrapolation operators and their integration in the computation of reachability relations, which seems to be a challenging task. Different strategies can also be adopted for the double forward analysis, switching between the two modes using heuristics, a parallel implementation, etc.

3 Algorithms for Probabilistic Systems

This chapter presents some of my contributions on the analysis of probabilistic systems. We first focus on the stochastic shortest path problem. More precisely, Section 3.2.1 presents a polynomial-time algorithm for MDPs with general weights; Section 3.2.2 explores the introduction of the variance in the objective; and Section 3.2.3 studies percentile queries over multi-dimensional MDPs for the shortest path objectives but also various other objectives. Section 3.3 presents results on MDPs with multiple probability transition functions.

3.1 Definitions

A finite *Markov decision process (MDP)* is a tuple $\mathcal{M} = (S, A, \delta)$, where S is a finite set of *states*, A a finite set of *actions*, and $\delta : S \times A \to \mathcal{D}(S)$ a partial function, where $\mathcal{D}(S)$ is the set of *probability distributions* on S. For any state $s \in S$, we denote by $A(s)$ the set of actions available from s. We define a *run* of $\mathcal{M}$ as a finite sequence $s_1 a_1 \dots a_{n-1} s_n$ of states and actions such that $\delta(s_i, a_i, s_{i+1}) > 0$ for all $1 \leq i \leq n-1$. Finite runs are also called *histories* and denoted $\mathcal{H}(\mathcal{M})$. We write $pref(\rho, i)$ for its prefix up to state s_i. let $\mathcal{H}_\infty(\mathcal{M})$ denote the set of infinite runs. We might omit $\mathcal{M}$ if it is clear from the context.

We define a *Markov chain* as an MDP in which $|A(s)| = 1$ for all states s.

Sub-MDPs and End-components For the following definitions, we fix an MDP $\mathcal{M} = (S, A, \delta)$. A *sub-MDP* $\mathcal{M}'$ *of* $\mathcal{M}$ is an MDP (S', A', δ') with $S' \subseteq S$, $A' \subseteq A$, and such that for all $s \in S'$, $A'(s) \neq \emptyset$ and for all $a \in A'(s)$, we have $\mathsf{Supp}(\delta(s,a)) \subseteq S'$, and $\delta'(s,a) = \delta(s,a)$, where Supp denotes the support of the given distribution. For all subsets $S' \subseteq S$ with the property that for all $s \in S'$, there exists $a \in A(s)$ with $\mathsf{Supp}(\delta(s,a)) \subseteq S'$, we define the *sub-MDP of* $\mathcal{M}$ *induced by* S' as the maximal sub-MDP whose states are S', and denote it by $M|_{S'}$. Specifically, the sub-MDP induced by S' contains all actions of S' whose supports are inside S'. An MDP is strongly connected if between any pair of states s, t, there is a run. An *end-component* of $\mathcal{M} = (S, A, \delta)$ is a sub-MDP $\mathcal{M}' = (S', A', \delta')$ with $S' \neq \emptyset$ that is strongly connected. It is known that the union of two end components with non-empty intersection is an end-component; one can thus define *maximal* end-components. We let $\mathsf{MEC}(\mathcal{M})$ denote the set of maximal end-components of $\mathcal{M}$, computable in polynomial time [dA97]. An *absorbing state* s is such that for all $a \in A(s)$, $\delta(s, a, s) = 1$.

We will consider the intersections of end-components $(S', A') \cap (S'', A'')$ to mean the pair $(S' \cap S'', A''')$ defined by $A'''(s) = A'(s) \cap A''(s)$ for all $s \in S' \cap S''$. Note that the intersection may not be an end-component. We say that (S', A') is included in (S'', A'') if $S' \subseteq S''$ and $A'(s) \subseteq A''(s)$ for all $s \in S' \cap S''$.

Histories and Schedulers A *scheduler* σ is a function $(SA)^*S \to \mathcal{D}(A)$ such that for all $h \in (SA)^*S$ ending in s, we have $\mathsf{Supp}(\sigma(h)) \in A(s)$. A scheduler is *deterministic* if all histories are mapped to *Dirac distributions*. A scheduler σ is *finite-memory* if it can be encoded with a *stochastic Moore machine*, $(\mathcal{M}, \sigma_a, \sigma_u, \alpha)$ where $\mathcal{M}$ is a finite set of memory elements, α the *initial distribution on* $\mathcal{M}$, σ_u the *memory update function* $\sigma_u : \mathcal{M} \times S \times A \to \mathcal{D}(\mathcal{M})$, and $\sigma_a : S \times \mathcal{M} \to \mathcal{D}(A)$ the *next action function* where $\mathsf{Supp}(\sigma_a(s,m)) \subseteq A(s)$ for any $s \in S$ and $m \in \mathcal{M}$. A *K-memory scheduler* is such that $|\mathcal{M}| = K$. A *memoryless scheduler* is such that $|\mathcal{M}| = 1$, and thus only depends on the last state of the history. We define such schedulers as functions $s \mapsto \mathcal{D}(A(s))$ for $s \in S$. An *MD-scheduler* is a memoryless deterministic scheduler.

An MDP $\mathcal{M}$, a finite-memory scheduler σ encoded by $(\mathcal{M}, \sigma_a, \sigma_u, \alpha)$, and a state s determine a finite Markov chain $\mathcal{M}_s^\sigma$ defined on the state space $S \times \mathcal{M}$ as follows. The initial distribution is such that for any $m \in \mathcal{M}$, state (s,m) has probability $\alpha(m)$, and 0 for other states. For any pair of states (s,m) and (s',m'), the probability of the transition $(s,m), a, (s',m')$ is equal to $\sigma_a(s,m)(a) \cdot \delta(s,a,s') \cdot \sigma_u(s,m,a)(m')$. A *run* of $\mathcal{M}_s^\sigma$ is a finite or infinite sequence of the form $(s_1,m_1), a_1, (s_2,m_2), a_2, \ldots$, where each $(s_i,m_i), a_i, (s_{i+1},m_{i+1})$ is a transition with nonzero probability in $\mathcal{M}_s^\sigma$, and $s_1 = s$. In this case, the run $s_1 a_1 s_2 a_2 \ldots$, obtained by projection to $\mathcal{M}$, is said to be *compatible with* σ. Given $E \subseteq (SA)^*$, we denote by $\mathbb{P}^\sigma_{M,s}[E]$ the probability of the runs of $\mathcal{M}_s^\sigma$ whose projection to $\mathcal{M}$ is in E, provided these sets are measurable.

For any scheduler σ in a MDP $\mathcal{M}$, and a sub-MDP $\mathcal{M}' = (S', A', \delta')$, we say that σ is *compatible with* $\mathcal{M}'$ if for any $h \in (SA)^*S'$, $\mathsf{Supp}(\sigma(h)) \subseteq A'(s)$.

Let $\mathsf{Inf}(\rho)$ denote the disjoint union of states and actions that occur infinitely often in the run ρ; Inf is thus seen as a random variable. By a slight abuse of notation, we say that $\mathsf{Inf}(\rho)$ is equal to a sub-MDP D whenever it contains exactly the states and actions of D. It was shown that for any MDP $\mathcal{M}$, state s, and scheduler σ it holds that Inf is equal to an end-component of $\mathcal{M}$ with probability 1 [dA97]. We call a subset S' of states *transient* if for all schedulers, runs starting in S' leave S' eventually with probability 1.

Quantitative Objectives Given an MDP $\mathcal{M} = (S, A, \delta)$, a weight function $\mathsf{wgt} : S \times A \to \mathbb{Z}$, we define the weight of a finite run $\rho = s_1 a_2 s_2 a_2 \ldots s_n$ as the sum of the weights of its state-action pairs: $\mathsf{wgt}(\rho) = \sum_{i=1}^{n-1} \mathsf{wgt}(s_i, a_i)$. For a finite or infinite run $\rho = s_1 a_2 s_2 a_2 \ldots$, we define the *total payoff until reaching* a target set *goal* $\subseteq S$ as $\Diamond\!\!\!\!+ goal(\rho) = \mathsf{wgt}(s_1 a_2 \ldots s_n)$ where s_n is the first visit of a state in *goal*. If *goal* is never reached, then we assign $\Diamond\!\!\!\!+ goal(\rho) = \infty$.

Given a random variable f, $\mathbb{E}^{\sup}_{\mathcal{M},s}(f) = \sup_\sigma \mathbb{E}^\sigma_{\mathcal{M},s}(f)$ and $\mathbb{E}^{\inf}_{\mathcal{M},s}(f) = \inf_\sigma \mathbb{E}^\sigma_{\mathcal{M},s}(f)$ denote the extremal expectations of f. We will denote the former $\mathbb{E}^{\max}_{\mathcal{M},s}(f)$ when the maximum exists.

We will also consider *mean payoff* objectives defined on infinite paths by

$$\mathrm{MP}(\rho) = \liminf_{n\to\infty} \frac{\mathsf{wgt}(pref(\rho, n))}{n}.$$

Recall that the maximal expected mean payoff in strongly connected MDPs does not depend on the starting state and that there exist MD-schedulers with a single *bottom strongly connected component* (BSCC) maximizing the expected mean payoff. When $\mathcal{M}$ is strongly connected, we omit the starting state and write $\mathbb{E}^{\max}_{\mathcal{M}}(\mathrm{MP})$.

3.2 The stochastic shortest path problem

3.2.1 The Stochastic Shortest Path Algorithm on General MDPs

We study the *stochastic shortest path (SSP) problem* on Markov decision processes which consists in computing schedulers that minimize the total payoff until reaching a target set. This problem is well understood and supported by various tools for finite-state MDPs with nonnegative weights only, for which the algorithms can rely on the monotonicity of accumulated weights along the prefixes of paths. In this case, schedulers that maximize or minimize the expected accumulated weight until reaching the target can be determined in polynomial time based on a preprocessing of end components (i.e. strongly connected sub-MDPs) and linear programs [BT91, dA99]. One can compute schedulers maximizing the probability for reaching the target within a given cost in pseudo-polynomial time using an iterative approach that successively increases the weight bound and treats zero-weight loops by linear-programming techniques [UB13, BDD$^+$14]. The corresponding decision problem is PSPACE-hard, even for acyclic MDPs [HK15].

For MDPs with arbitrary integer weights, the lack of monotonicity of accumulated weights makes analogous questions much harder. Even for finite-state Markov chains with integer weights, the set of relevant configurations (i.e. states augmented with the weight that has been accumulated so far) can be infinite and, in MDPs with integer weights optimal or ε-optimal schedulers might require an infinite amount of memory. The latter is known from energy-MDPs [CD11, BKN16, MSTW17] where one aims at finding a scheduler under which the system never runs out of energy (i.e. the accumulated weight plus some initial credit is always positive) and satisfies an ω-regular property (e.g. a parity condition) with probability 1 or maximizes the expected mean payoff. Another indication for the additional difficulties that arise when switching from nonnegative weights to integers is given by the work on one-counter MDPs [BBE$^+$10], which can be seen as MDPs where all weights are in $\{-1, 0, +1\}$ and that terminate as soon as the counter value is 0. Among others, [BBE$^+$10] establishes PSPACE-hardness and an EXPTIME upper bound for the almost-sure termination problem under some scheduler, while the corresponding weight-bounded (control-state) reachability problem in nonnegative MDPs is in PTIME [UB13].

Here, we address a fundamental problem for MDPs with integer weights. Our main contribution is to show that the classical stochastic shortest path problem, where the task is to *minimize the expected weight* until reaching a target, is solvable in polynomial time for arbitrary integer-weighted MDPs. We hereby extend previous results for restricted classes of MDPs [BT91, dA99] that left the general case open.

Although several other problems for integer-weighted MDPs are known to be in NP $\cap$ coNP and as hard as nonstochastic two-player mean-payoff games (see, e.g. [CD11, MSTW17, BFRR17]), our techniques crucially depart from previous work by heavily relying on new algorithms to classify end components (ECs) of MDPs. We see these results on the *classification of ECs* as a further contribution as it provides a useful vehicle for reasoning about different problems for integer-weighted MDPs.

Our classification of ECs is according to the existence of schedulers that increase the weight to infinity (*pumping ECs*), or ensure that the weight eventually exceeds any threshold possibly without having $+\infty$ as a limit (*weight-divergent ECs*), or have oscillating behavior (*gambling ECs*), or keep the accumulated weights within a compact interval (*bounded ECs*).

A sufficient and necessary criterion for the pumping property is that the maximal expected mean payoff is positive, which is decidable in polynomial time by computing the maximal expected

mean payoff using linear-programming techniques [Put94, Kal11]. While this observation has been made by several other authors, we are not aware of earlier algorithms for checking the gambling or boundedness property. For checking weight-divergence, the results of [BBE+10] on one-counter MDPs without boundary yield a polynomial time bound for the special case of MDPs where all weights are in $\{+1, 0, -1\}$ and a pseudo-polynomial time bound in the general case. We improve this result by presenting a polynomial-time algorithm for deciding weight-divergence for MDPs with arbitrary integer weights. Moreover, in case that the given MDP $\mathcal{M}$ is not weight-divergent, the algorithm generates a new MDP $\mathcal{N}$ with the same state space that has no 0-ECs (i.e. end components where the accumulated weight of all cycles is 0) and that is equivalent to $\mathcal{M}$ for all properties that are invariant with respect to behaviors inside 0-ECs. The generation of such an MDP $\mathcal{N}$ relies on an iterative technique to flatten 0-ECs. This new technique, called *spider construction*, can be seen as a generalization of the method proposed in [dA97, dA99] to eliminate 0-ECs in nonnegative MDPs. There, all states that belong to some maximal end component of the sub-MDP built by state-action pairs with weight 0 are collapsed. This technique obviously fails for integer-weighted MDPs as 0-ECs can contain state-action pairs with negative and positive weights. The spider construction maintains the state space, but turns the graph structure of maximal 0-ECs into an acyclic graph with a single sink state that captures the original behavior of all other states in the same maximal 0-EC. Besides deciding weight-divergence, the spider construction will be the key to solve the classical shortest path problem for arbitrary integer-weighted MDPs.

Checking the gambling property is NP-complete in the general case, but can be decided in polynomial time using the spider construction, provided that the maximal expected mean payoff is 0. We establish an analogous result for the boundedness property, shown to be equivalent to the existence of 0-ECs in cases where the given end component has maximal expected mean payoff 0.

These results were obtained in collaboration with Christel Baier and her group [S1]. She was the driving force in this work.

Classification of End Components As basic building blocks of our algorithms, we define four types of schedulers and end components of MDPs. The *pumping* end components have a scheduler that let the accumulated weight almost surely diverge to infinity; positively (resp. negatively) *weight-divergent* ones have a scheduler where almost surely the limsup (resp. liminf) of the accumulated sum is infinity (resp. minus infinity); the *gambling* ones have schedulers with expected mean payoff 0 and where the accumulated weight approaches both plus and minus infinity with probability 1; while the *zero end components* only have 0 cycles, so the weight stays bounded with probability 1.

Definition 6. *An infinite path ρ in an MDP $\mathcal{M}$ is called*

- pumping *if* $\liminf_{n\to\infty} \mathsf{wgt}(\mathit{pref}(\rho, n)) = +\infty$,
- positively weight-divergent*, or briefly* weight-divergent*,* *if* $\limsup_{n\to\infty} \mathsf{wgt}(\mathit{pref}(\rho, n)) = +\infty$,
- negatively weight-divergent *if* $\liminf_{n\to\infty} \mathsf{wgt}(\mathit{pref}(\rho, n)) = -\infty$,
- gambling *if ρ is positively and negatively weight-divergent,*
- bounded from below *if* $\liminf_{n\to\infty} \mathsf{wgt}(\mathit{pref}(\rho, n)) \in \mathbb{Z}$.

A scheduler σ for $\mathcal{M}$ is called *pumping from state s* if $\Pr^{\sigma}_{\mathcal{M},s}\{\rho \in \mathcal{H}_\infty : \rho \text{ is pumping}\} = 1$, *i.e.*almost all σ-paths from s are pumping. σ is called *pumping* if it is pumping from all states s. The MDP

$\mathcal{M}$ itself is said to be *pumping* if it has at least one pumping scheduler. $\mathcal{M}$ is called *universally pumping* if all schedulers of $\mathcal{M}$ are pumping.

The notions of weight-divergent (or negatively weight-divergent or bounded from below) schedulers and MDPs are defined analogously, requiring these properties on runs to hold with probability 1. *Gambling* schedulers are those where almost all paths are gambling and where the expected mean payoff is 0. A strongly connected MDP $\mathcal{M}$ is called *gambling* if $\mathbb{E}^{\max}_{\mathcal{M}}(\mathrm{MP}) = 0$ and $\mathcal{M}$ has a gambling scheduler (see Fig. 3.1).

Obviously, a strongly connected MDP $\mathcal{M}$ is pumping (universal pumping or weight-divergent or gambling, respectively) from some state iff $\mathcal{M}$ is pumping (universal pumping or weight-divergent or gambling, respectively).

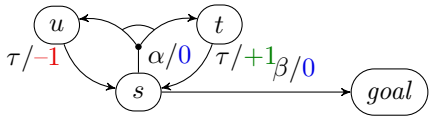


Figure 3.1: EC $\mathcal{E} = \{(s,\alpha),(u,\tau),(t,\tau)\}$ is gambling in case all distributions are uniform. The MD scheduler that always takes (s,α) is gambling. Moreover, *goal* can be reached almost surely for any weight threshold, using the infinite-memory scheduler that takes (s,α) if below the threshold, and (s,β) otherwise. One can show that this cannot be achieved with a finite-memory scheduler.

A cycle ξ in $\mathcal{M}$ is called positive if $\mathsf{wgt}(\xi) > 0$, and negative if $\mathsf{wgt}(\xi) < 0$.

A *zero end component* (0-EC) is an end component $\mathcal{E}$ where $\mathsf{wgt}(\xi) = 0$ for each cycle ξ in $\mathcal{E}$ and use the term *0-BSCC* when $\mathcal{E}$ contains at most one state-action pair (s,α) for each state s in $\mathcal{E}$. Thus, each 0-BSCC is a bottom strongly connected component of an MD-scheduler.

We recall characterizations of these notions for Markov chains (see e.g. [KSBD15]):

Lemma 7. *Let $\mathcal{C}$ be a strongly connected finite Markov chain.*

(a) $\mathcal{C}$ is pumping iff $\mathbb{E}_{\mathcal{C}}(\mathrm{MP}) > 0$.

(b) $\mathbb{E}_{\mathcal{C}}(\mathrm{MP}) = 0$ iff $\mathcal{C}$ is a 0-BSCC or $\mathcal{C}$ is gambling.

(c) If $\mathbb{E}_{\mathcal{C}}(\mathrm{MP}) = 0$ then the following statements are equivalent: (1) $\mathcal{C}$ is gambling, (2) $\mathcal{C}$ is positively weight-divergent, (3) $\mathcal{C}$ is negatively weight-divergent, (4) $\mathcal{C}$ has a positive cycle, (5) $\mathcal{C}$ has a negative cycle.

(d) If $\mathbb{E}_{\mathcal{C}}(\mathrm{MP}) = 0$ then the following are equivalent: (1) $\mathcal{C}$ is a 0-BSCC, (2) $\mathcal{C}$ is bounded from below, (3) the set of paths bounded from below has positive measure.

We provide an analogous characterization for strongly connected MDPs, and provide algorithms to decide the type of a given end-component.

This is simple for the existential and universal pumping property, checkable in polynomial time.

Lemma 8. *Let $\mathcal{M}$ be a strongly connected MDP. Then, $\mathcal{M}$ is pumping iff $\mathcal{M}$ has a pumping MD-scheduler iff $\mathbb{E}^{\max}_{\mathcal{M}}(\mathrm{MP}) > 0$. Likewise, $\mathcal{M}$ is universally pumping iff all MD-schedulers are pumping iff $\mathbb{E}^{\min}_{\mathcal{M}}(\mathrm{MP}) > 0$.*

We will show how to check weight-divergence, the gambling property, and how to compute the states belonging to a 0-EC.[3] We start with an observation on weight-divergence.

Lemma 9. *Let $\mathcal{M}$ be a strongly connected MDP. If $\mathcal{M}$ is positively weight-divergent then $\mathbb{E}^{\max}_{\mathcal{M}}(\mathrm{MP}) \geqslant 0$. Conversely, if $\mathbb{E}^{\max}_{\mathcal{M}}(\mathrm{MP}) > 0$, then $\mathcal{M}$ is positively weight-divergent.*

Notice that the nontrivial case is when $\mathbb{E}^{\max}_{\mathcal{M}}(\mathrm{MP}) = 0$. In this case, the MDP can be weight-divergent or bounded from below, depending on the presence of positive or negative cycles.

Eliminating 0-ECs We now present a method to eliminate a given 0-EC from an MDP by "flattening" it. This so-called *spider construction*[4] preserves the state space and all properties of interest, in particular, those that are invariant by adding or removing path segments of weight 0.

We will omit the details of the construction but only summarize the main properties below.

Lemma 10. *Given an MDP $\mathcal{M}$, and 0-EC $\mathcal{E}$, one can compute an MDP $\mathsf{Spider}_{\mathcal{E}}(\mathcal{M})$, in polynomial time, that satisfies the following properties:*

(S1) $\mathcal{M}$ and $\mathsf{Spider}_{\mathcal{E}}(\mathcal{M})$ have the same state space and $\|\mathsf{Spider}_{\mathcal{E}}(\mathcal{M})\| = \|\mathcal{M}\|-1$.

(S2) If $\mathcal{E} \neq \mathcal{M}$ and $\mathcal{M}$ is strongly connected then $\mathsf{Spider}_{\mathcal{E}}(\mathcal{M})$ has a single MEC that is reachable from all states.

(S3) $\mathcal{M}$ is negatively (resp. positively) weight-divergent iff $\mathsf{Spider}_{\mathcal{E}}(\mathcal{M})$ is. Moreover, for each goal, there is $\overline{goal}$ such that $\mathbb{E}^{\inf}_{\mathcal{M}}[\diamondsuit goal] = \mathbb{E}^{\inf}_{\mathsf{Spider}_{\mathcal{E}}(\mathcal{M})}[\diamondsuit \overline{goal}]$.

(S4) Suppose that $\mathcal{E}$ is contained in an MEC $\mathcal{G}$ of $\mathcal{M}$ with $\mathbb{E}^{\max}_{\mathcal{G}}(\mathrm{MP}) = 0$. Then for each state s with $s \notin \mathcal{E}$: s belongs to a 0-EC of $\mathcal{M}$ iff s belongs to a 0-EC of $\mathsf{Spider}_{\mathcal{E}}(\mathcal{M})$. Likewise, for each state-action pair (s,α) of $\mathcal{M}$: (s,α) belongs to a 0-EC of $\mathcal{M}$ iff $(s,\alpha) \in \mathcal{E}$ or (s_0,α) belongs to a 0-EC of $\mathsf{Spider}_{\mathcal{E}}(\mathcal{M})$.

The main property of the spider construction is that it eliminates the given 0-BSCC while maintaining all other 0-EC, as stated in (S4). (S3) states an equivalence between $\mathcal{M}$ and $\mathsf{Spider}_{\mathcal{E}}(\mathcal{M})$ with respect to properties of interest.

Checking Weight-Divergence We present an algorithm to check the weight-divergence of an end component (see Algorithm 4).

Given a strongly connected MDP $\mathcal{M}$ we first compute $\mathbb{E}^{\max}_{\mathcal{M}}(\mathrm{MP})$ and an MD-scheduler σ maximizing the expected mean payoff. If $\mathbb{E}^{\max}_{\mathcal{M}}(\mathrm{MP}) > 0$ then $\mathcal{M}$ is pumping (Lemma 8) and therefore positively weight-divergent. If $\mathbb{E}^{\max}_{\mathcal{M}}(\mathrm{MP}) < 0$ then all schedulers for $\mathcal{M}$ are negatively weight-divergent (Lemma 8 with weights multiplied by -1), and hence, $\mathcal{M}$ is not positively weight-divergent. If $\mathbb{E}^{\max}_{\mathcal{M}}(\mathrm{MP}) = 0$ and σ has a gambling BSCC then $\mathcal{M}$ is gambling and therefore positively weight-divergent. Otherwise, each BSCC of the Markov chain induced by σ is a 0-BSCC (Lemma 7) and we pick such a 0-BSCC $\mathcal{E}$ of σ. In case $\mathcal{M} = \mathcal{E}$ then $\mathcal{M}$ is a 0-EC, hence not weight-divergent, and the algorithm terminates. If $\mathcal{M} \neq \mathcal{E}$, we apply the spider construction to generate the MDP

[3] We focus here on results for (positive) weight-divergence. The negative case can be obtained analogously by multiplying all weights with -1.

[4] The name is due to the shape obtained after application of the transformation.

input : strongly connected MDP $\mathcal{M}$
output : "yes" if $\mathcal{M}$ is weight divergent and "no" otherwise

Compute $e := \mathbb{E}^{\max}_{\mathcal{M}}(\text{MP})$ and σ with $\mathbb{E}^{\sigma}_{\mathcal{M}}(\text{MP}) = e$
if $e < 0$ **then return** "no"
if $e > 0$ *or* σ *has a gambling BSCC* **then return** "yes"
Pick a 0-BSCC $\mathcal{E}$ of σ
if $\mathcal{M} = \mathcal{E}$ **then return** "no"
Compute the MEC $\mathcal{F}$ of $\mathsf{Spider}_{\mathcal{E}}(\mathcal{M})$ that is reachable from all states and **return** $\mathsf{Wgtdiv}(\mathcal{F})$

Algorithm 4: $\mathsf{Wgtdiv}(\cdot)$ that checks the weight-divergence of $\mathcal{M}$.

$\mathsf{Spider}_{\mathcal{E}}(\mathcal{M})$ that contains a unique maximal end component $\mathcal{F}$ ((S2) in Lemma 10). Repeating the procedure recursively on $\mathcal{F}$ etc. thus generates a sequence of MDPs $\mathcal{M}_0 = \mathcal{M}, \mathcal{M}_1, \ldots, \mathcal{M}_\ell$ with $\mathcal{M}_{i+1} = \mathsf{Spider}_{\mathcal{E}_i}(\mathcal{M}_i)$ for some 0-BSCC $\mathcal{E}_i$ of $\mathcal{M}_i$. All $\mathcal{M}_i$'s have the same state space and the number of state-action pairs is strictly decreasing, *i.e.*we have $\|\mathcal{M}_0\| > \|\mathcal{M}_1\| > \ldots > \|\mathcal{M}_\ell\|$ by property (S1) in Lemma 10. Moreover, $\mathcal{M}_i$ is weight-divergent iff $\mathcal{M}$ is weight-divergent.

As each iteration takes polynomial time and the size of each $\mathcal{M}_i$ is polynomially bounded by the size of $\mathcal{M}$ the algorithm runs in polynomial time. Using an inductive argument and Lemma 10, we obtain:

Theorem 11. *The algorithm for checking weight-divergence of a strongly connected MDP $\mathcal{M}$ runs in polynomial time. If $\mathcal{M}$ is weight-divergent then it finds a pumping or a gambling MD-scheduler. If $\mathcal{M}$ is not weight-divergent, then it generates an MDP $\mathcal{N}$ without 0-ECs on the same state space as $\mathcal{M}$, and is equivalent to $\mathcal{M}$ in the sense of (S3) in Lemma 10.*

Application to Stochastic Shortest Path We present an algorithm to solve the stochastic shortest path problem that relies on the classification of end components presented above. The classical shortest path problem for MDPs is to compute the *minimal expected accumulated weight* until reaching a goal state *goal*. Here, the infimum is taken over all *proper* schedulers. These are schedulers σ that reach *goal* almost surely, *i.e.* $\Pr^{\sigma}_{\mathcal{M},s}(\Diamond \mathit{goal}) = 1$ for all states $s \in S$.

We assume, w.l.o.g., that *goal* is a trap state, and that all states s are reachable from an initial state s_{init} and can reach *goal*. The *stochastic shortest path problem* aims at computing the minimal expected accumulated weight until reaching *goal*:

$$\mathbb{E}^{\inf}_{\mathcal{M},s_{\mathit{init}}}(\oplus\, \mathit{goal}) \;=\; \inf_{\sigma \text{ proper}} \mathbb{E}^{\sigma}_{\mathcal{M},s_{\mathit{init}}}(\oplus\, \mathit{goal}) \;.$$

Although for each proper scheduler this quantity is finite, the infimum may be $-\infty$. We describe a polynomial-time algorithm to check whether $\mathbb{E}^{\inf}_{\mathcal{M},s_{\mathit{init}}}(\oplus\, \mathit{goal})$ is finite and to compute it, both using our classification of end components.

It is well known (see, *e.g.*[Kal11]) that if $\mathcal{M}$ is *contracting*, *i.e.* if all schedulers are proper, then $\mathbb{E}^{\inf}_{\mathcal{M},s_{\mathit{init}}}(\oplus\, \mathit{goal}) > -\infty$ and one can compute $\mathbb{E}^{\inf}_{\mathcal{M},s_{\mathit{init}}}(\oplus\, \mathit{goal})$ using linear-programming techniques. To relax the assumption of $\mathcal{M}$ being contracting, Bertsekas and Tsitsiklis [BT91] identified conditions that guarantee the finiteness of the values $\mathbb{E}^{\inf}_{\mathcal{M},s_{\mathit{init}}}(\oplus\, \mathit{goal})$, the existence of a minimizing MD-scheduler, and the computability of the vector $(\mathbb{E}^{\inf}_{\mathcal{M},s}(\oplus\, \mathit{goal}))_{s\in S}$ as the unique solution of a linear program (or using value and policy iteration). The assumptions of [BT91], written (BT) in the sequel, are: (i) existence of a proper scheduler, and (ii) under each non-proper scheduler the expected accumulated weight is $+\infty$ from at least one state. While these assumptions are sound, they are

incomplete in the sense that there are MDPs where $\mathbb{E}^{\inf}_{\mathcal{M},s}(\diamondsuit\, goal)$ is finite for all states s, but (BT) does not hold.

Orthogonally, De Alfaro [dA99] showed that in MDPs where the weights are either all nonnegative or all nonpositive, one can decide in polynomial time whether $\mathbb{E}^{\inf}_{\mathcal{M},s_{init}}(\diamondsuit\, goal)$ is finite. Moreover, when this is the case, $\mathcal{M}$ can be transformed into another MDP that has proper schedulers, satisfies (BT) and preserves the minimal expected accumulated weight. Using the classification of end components, we generalize De Alfaro's result and provide a characterization of finiteness of the minimal expected accumulated weight.

Lemma 12. *Let $\mathcal{M}$ be an MDP with a distinguished initial state s_{init} and a trap state goal such that all states are reachable from s_{init} and can reach goal. Then, $\mathbb{E}^{\inf}_{\mathcal{M},s_{init}}(\diamondsuit\, goal)$ is finite iff $\mathcal{M}$ has no negatively weight-divergent end component. If so, then $\mathcal{M}$ satisfies (BT) iff $\mathcal{M}$ has no 0-EC.*

The above lemma allows us to derive our algorithm by first determining if $\mathbb{E}^{\inf}_{\mathcal{M},s_{init}}(\diamondsuit\, goal)$ is finite, and then using the iterative spider construction to transform $\mathcal{M}$ into an equivalent new MDP satisfying (BT).

More precisely, one can check in polynomial time whether $\mathbb{E}^{\inf}_{\mathcal{M},s_{init}}(\diamondsuit\, goal) > -\infty$ by applying the weight divergence algorithm to the maximal end components of $\mathcal{M}$ (in fact, checking negative weight-divergence reduces to checking positive weight-divergence after multiplication of all weights by -1). If so, by the iterative spider construction to flatten 0-ECs. we obtain in polynomial time an MDP $\mathcal{N}$ such that $\mathcal{N}$ satisfies condition (BT) and $\mathbb{E}^{\inf}_{\mathcal{N},s}(\diamondsuit\, goal) = \mathbb{E}^{\inf}_{\mathcal{M},s}(\diamondsuit\, goal)$ for each state s. This yields:

Theorem 13. *Given an arbitrary MDP $\mathcal{M}$, one can compute in polynomial time $\mathbb{E}^{\inf}_{\mathcal{M},s_{init}}(\diamondsuit\, goal)$ as well as an MD scheduler achieving the minimum when this value is finite.*

Analogous results are obtained for $\mathbb{E}^{\sup}_{\mathcal{M},s_{init}}(\diamondsuit\, goal)$ by multiplying all weights in $\mathcal{M}$ with -1.

3.2.2 Variance-Penalized Stochastic Shortest Path Problem

While a solution to the stochastic shortest path problem provides guarantees on the behavior of a system in all environments or indicates the optimal control to maximize expected rewards, it completely disregards all other aspects of the resulting probability distribution of the accumulated weight besides the expected value. In almost all practical applications, however, the uncertainty coming with the probabilistic behavior cannot be neglected. In traffic control systems or energy grids, for example, large variability in the throughput comes at a high cost due to the risk of traffic jams or the difficulty of storing surplus energy. Also a probabilistic program employed in a complex environment might be of more use with a higher expected termination time in exchange for a lower chance of extreme termination times.

To overcome these shortcomings of the SSP problem, various additional optimization problems have been studied in the literature: Optimizing conditional expected accumulated weights under the condition that certain system states are reached allows for a more fine-grained system analysis by making it possible to determine the worst- or best-case expectation in different scenarios [BKKW17, PB19]. Given a probability p, quantiles on the accumulated weight in MDPs, also called *values-at-risk* in the context of risk analysis, are the best bound B such that the accumulated weight exceeds B with probability at most p in the worst or best case [HK15, UB13]. The *conditional value-at-risk* and the *entropic value-at-risk* are more involved measures that have been studied in this context [ADBA21, KM18]. They quantify how far the probability mass of the tail of the probability

distribution lies above the value-at-risk. The arguably most prominent measure for the deviation of a random variable from its expected value is the *variance*. The computation of the variance of accumulated weights has been studied in Markov chains [Ver04] and in MDPs [Man71, MT11]. The investigations of variance in MDPs in the literature is discussed in more detail in the 'Related Work' section below.

In this section, we investigate a variant of the SSP problem in which the costs caused by probabilistic uncertainty are priced in to the objective function: We study the optimization of the *variance-penalized expectation* (VPE), a well-known measure that combines the expected value μ and the variance σ^2 into the single objective function $\mu - \lambda \cdot \sigma^2$ where λ is a parameter that can be varied to aim for different tradeoffs between expectation and variance. These results were published in [S24].

In the context of optimization problems on MDPs, the VPE has been studied, e.g., in [FKL89, Col97]. Furthermore, the VPE finds use in an area of research primarily concerned with the tradeoffs between expected performance and risks, namely, the theory of financial markets and investment decision-making: In 1952, Harry Markowitz introduced *modern portfolio theory* that evaluates portfolios in terms of expected returns and variance of the returns [Mar52], for which he was later awarded the Nobel Prize in economics. A portfolio lies on the *Markowitz efficient frontier* if the expected return cannot be increased without increasing the variance and, vice versa, the variance cannot be decreased without decreasing the expectation. The final choice of a portfolio on the efficient frontier depends on the investors preferences. In this context, the VPE $\mu - \lambda \cdot \sigma^2$ is a simple, frequently used way to express the preference of an investor using the single parameter λ capturing the risk-aversion of the investor (see, e.g., [GBGE14]). In more involved accounts, the investor's preference is described in terms of a utility function mapping returns to utilities. For the commonly used exponential utility function $u(x) = -e^{-\alpha x}$ and normally distributed returns, the objective of an investor trying to maximize expected utility turns out to be equivalent to the maximization of the VPE with parameter $\lambda = \alpha/2$ [Arr70, Pra64].

For an illustration of the VPE, consider the following example:

Example 14. *Consider the MDP $\mathcal{M}$ depicted in Figure 3.2 where non-trivial probability values as well as the weights accumulated are denoted next to the transitions. We want to analyze the possible trade-offs between the variance and the expected value of the accumulated weight that we can achieve in this MDP.*

The only non-deterministic choice is in the state s_{init}. Choosing action α leads to goal with expected weight and variance 0*. For the remaining actions, the accumulated weight follows a geometric distribution where in each step some weight k is accumulated and goal is reached with some probability p after the step. For such a distribution, it is well-known that the expected accumulated weight is k/p and the variance is $(k/p)^2 \cdot (1-p)$. Plugging in the respective values for the distributions reached after actions β, γ, and δ, we obtain the pairs of expectations and variances as depicted on the right-hand side of Figure 3.2. In particular, choosing γ leads to an expectation of* $10/3$ *and a variance of* $10/9$*.*

Making use of randomization over two different actions τ and σ with probability p and $1-p$, respectively, for some $p \in (0,1)$, we show that the expected values and variances under the resulting schedulers lie on a parabolic line segment depicted in black that is uniquely determined by the expected values and variances under τ and σ. By further randomization over multiple actions, combinations of expectation and variance in the gray region in Figure 3.2 can be realized.

Consider now the VPE with parameter $\lambda = 1$. The dashed blue line in Figure 3.2 marks all points at which $\mu - 1 \cdot \sigma^2 = 20/9$. The arrow indicates in which direction the value of the VPE

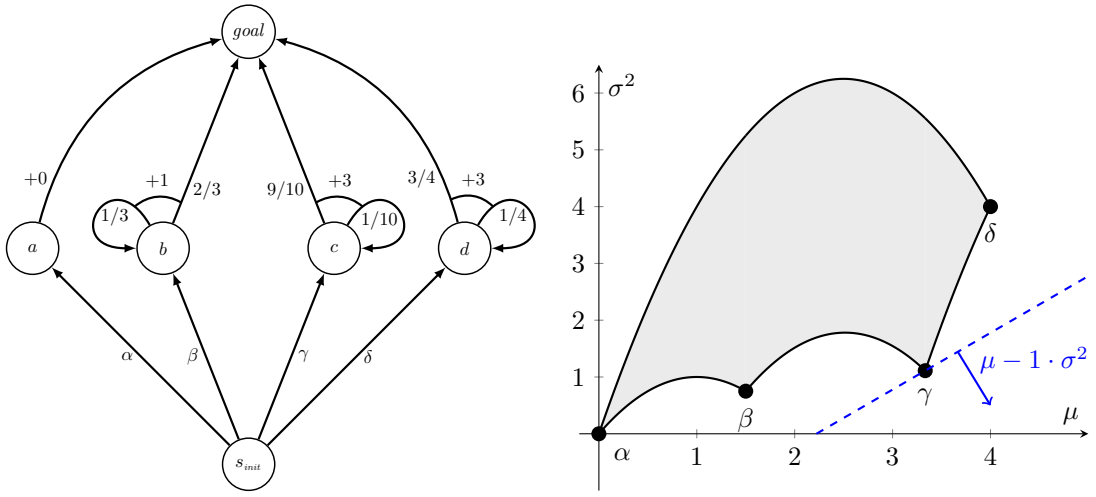


Figure 3.2: The left hand side shows the MDP $\mathcal{M}$ for Example 14. On the right hand side, all possible combinations of expected accumulated weight and variance for schedulers for $\mathcal{M}$ are depicted. The points corresponding to the four deterministic schedulers are marked by the corresponding action. Furthermore, the blue line indicates all points at which $\mu - 1 \cdot \sigma^2 = 20/9$ and the arrow indicates the direction in which the value of this objective function increases.

increases. So, it turns out that choosing action γ maximizes the VPE in this case; the slightly lower expectation compared to δ is compensated by a significantly lower variance. Geometrically, we can observe that the optimal point for the VPE for any parameter will always lie on the border of the convex hull of the region of feasible points in the μ-σ^2-plane as the VPE is a linear function of expectation and variance. For varying values of λ, also α (for $\lambda \geq 3$) and δ (for $\lambda \leq 1/13$) can constitute the optimal choice in s_{init} for the maximization of the VPE, while β is not optimal for any choice of λ as it lies in the interior of the convex hull of the feasible region. Our results show that in general, the optimal point for the VPE can be achieved by a deterministic finite-memory scheduler.

Contributions. Our results on this problem can be summarized as follows [S24].

1. Among all schedulers that optimize the expected accumulated weight before reaching a target, a variance-minimal scheduler can be computed in polynomial time and chosen to be memoryless and deterministic.

2. The maximal VPE in MDPs with non-negative weights can be computed in exponential space. The maximum is obtained by a deterministic scheduler that can be computed in exponential space as well. As memory, an optimal scheduler only needs to keep track of the accumulated weight up to a bound computable in polynomial time. As soon as the bound is reached, optimal schedulers can switch to the behavior of a variance-minimal scheduler among the expectation-minimal schedulers that can be computed by result 1.

3. The threshold problem whether the maximal VPE is greater than or equal to a rational ϑ is in NEXPTIME and EXPTIME-hard.

Minimizing Variance among Expectation-Optimal Schedulers The variance of a random variable X under the probability measure determined by σ and s in $\mathcal{M}$ is denoted by $\mathbb{V}^{\sigma}_{\mathcal{M},s}(X)$ and defined by

$$\mathbb{V}^{\sigma}_{\mathcal{M},s}(X) = \mathbb{E}^{\sigma}_{\mathcal{M},s}((X - \mathbb{E}^{\sigma}_{\mathcal{M},s}(X))^2) = \mathbb{E}^{\sigma}_{\mathcal{M},s}(X^2) - \mathbb{E}^{\sigma}_{\mathcal{M},s}(X)^2.$$

Let us call a scheduler *expectation-optimal* if it maximizies the expectation of $\oplus goal$ from a given state s. Here we present a result that is of interest in its own right and that plays a crucial role in our investigation of the optimization of the VPE. Namely, we show how to compute a scheduler that minimizes the variance among expectation-optimal schedulers in polynomial time. Note that in MDPs with weights in $\mathbb{Z}$, the mimimization of the expectation of $\oplus goal$ can be reduced to the maximization by multiplying all weights with -1. This change of weights does not affect the variance and hence all results of this section also apply to expectation-minimal schedulers.

We assume that in a given MDP $\mathcal{M} = (S, A, P)$, with initial state s_{init}, weight function wgt, and goal states *goal*, the maximal achievable expectation of $\oplus goal$ is finite. This can be checked in polynomial time (see Section 3.2.1, or [S1]) and, when this value is finite, it is achievable by memoryless deterministic schedulers. By Section 3.2.1, all end components E of $\mathcal{M}$ are then either 0-end components or satisfy $\mathbb{E}^{\max}_{E}(\mathrm{MP}) < 0$. In fact, a reachable end component E satisfying $\mathbb{E}^{\max}_{E}(\mathrm{MP}) > 0$ implies that the optimal expectation is infinite; so does a weight-divergent E with $\mathbb{E}^{\max}_{E}(\mathrm{MP}) = 0$.

The algorithm proceeds as follows. First, a transformation is applied so as to ensure that the only end-components in $\mathcal{M}$ are such that the maximal achievable expected mean payoff is negative; while preserving the expectation and the variance of $\oplus goal$ (Lemma 15). We then prune the MDP so that all actions are optimal for maximizing the expected $\oplus goal$. It follows that all schedulers then achieve the same expected $\oplus goal$. We then derive an equation system in which the variances at each state are unknowns, while the expectations are known constants (Lemma 17). We conclude by showing that this equation system admits a unique solution and is solvable in polynomial time.

The following lemma is a consequence of Lemma 10.

Lemma 15. *Let $\mathcal{M} = (S, A, \delta)$ with weight function wgt, and target state goal, satisfying $\mathbb{E}^{\max}_{\mathcal{M},s_{init}}(\oplus goal) < \infty$ from state state s_{init}. There is a polynomial transformation which outputs an MDP $\mathcal{M}'$ with the following properties:*

1. *$\mathcal{M}'$ has no 0-end-components,*
2. *there is a mapping f from schedulers of $\mathcal{M}$ to those of $\mathcal{M}'$ such that for all* proper *schedulers σ for $\mathcal{M}$, $\mathbb{E}^{\sigma}_{\mathcal{M},s_{init}}(\oplus goal) = \mathbb{E}^{f(\sigma)}_{\mathcal{M}',s_{init}}(\oplus goal)$, and $\mathbb{V}^{\sigma}_{\mathcal{M},s_{init}}(\oplus goal) = \mathbb{V}^{f(\sigma)}_{\mathcal{M}',s_{init}}(\oplus goal)$.*
3. *there is a mapping g from schedulers of $\mathcal{M}'$ to those of $\mathcal{M}$ such that for all* proper *schedulers σ for $\mathcal{M}'$, $\mathbb{E}^{g(\sigma)}_{\mathcal{M},s_{init}}(\oplus goal) = \mathbb{E}^{\sigma}_{\mathcal{M}',s_{init}}(\oplus goal)$, and $\mathbb{V}^{g(\sigma)}_{\mathcal{M},s_{init}}(\oplus goal) = \mathbb{V}^{\sigma}_{\mathcal{M}',s_{init}}(\oplus goal)$.*

From now on, by the previous lemma, we assume that $\mathcal{M}$ only has end-components E with $\mathbb{E}^{\max}_{\mathcal{M},s_{init}}(E) < 0$. We start by computing $\mathbb{E}^{\max}_{\mathcal{M}}(\oplus goal)$ with the following equation:

$$\mu_s = \begin{cases} 0 & \text{if } s = goal, \\ \max_{a \in A(s)} \sum_{s' \in S} \delta(s,a,s')(\mathsf{wgt}(s,a) + \mu_{s'}) & \text{otherwise.} \end{cases} \qquad (*)$$

By [BT91], $(*)$ has a unique solution $\mu_s = \mathbb{E}^{\max}_{\mathcal{M}}(\oplus goal)$ and this solution is computable in polynomial time via linear programming. Let us define $A^{\max}(s)$ as the set of actions from s which satisfy $(*)$ with equality, i.e. $A^{\max}(s) = \{a \in A(s) \mid \mu_s = \mathsf{wgt}(s,a) + \sum_{s' \in S} \delta(s,a,s')\mu_{s'}\}$, and

let $\mathcal{M}'$ be obtained by restricting $\mathcal{M}$ to actions from $A^{\max}$. By standard arguments, we can show the following lemma:

Lemma 16. *Let $(\mu_s)_{s\in S}$ be the solution of $(*)$ for an MDP $\mathcal{M}$. Let $\mathcal{M}'$ obtained from $\mathcal{M}$ as above. Then, $\mathcal{M}'$ has no end-components. Moreover, for all $s \in S$, all schedulers σ of $\mathcal{M}'$ achieve $\mathbb{E}^{\sigma}_{\mathcal{M}'}[\oplus goal] = \mu_s$.*

So, in order to find the variance-minimal scheduler among expectation optimal schedulers for $\mathcal{M}$, it is sufficient to find a variance-minimal scheduler for $\mathcal{M}'$. We derive the following lemma by adapting [Ver04] to MDPs.

Lemma 17. *Consider an MDP $\mathcal{M}$, and assume that there is a vector $(\mu_s)_{s\in S}$ of values such that all schedulers σ satisfy $\forall s \in S, \mathbb{E}^{\sigma}_{\mathcal{M},s}(\oplus goal) = \mu_s$. Then, $(\mathbb{V}^{\inf}_{\mathcal{M},s}(\oplus goal))_{s\in S}$ is the unique solution of the following equation:*

$$V_s = \begin{cases} 0 & \text{if } s = goal, \\ \min_{a\in A(s)} \sum_{t\in S} \delta(s,a,t)\left((\mathsf{wgt}(s,a) + \mu_t - \mu_s)^2 + V_t\right) & \text{otherwise.} \end{cases} \qquad (**)$$

Note that the equation system $(**)$ is the same as the equation system used to minimize the expected accumulated weight before reaching *goal* under the weight function wgt' that assigns the non-negative weight $(\mathsf{wgt}(s,a) + \mu_t - \mu_s)^2$ to the transition (s,α,t). So, this equation system is solvable in polynomial time [dA99]. Using that all schedulers in $\mathcal{M}'$ achieve an expected accumulated weight of μ_s when starting in state s, the results of this section can be combined to the following theorem.

Theorem 18. *Given an MDP $\mathcal{M}$ such that $\mathbb{E}^{\max}_{\mathcal{M}}[\oplus goal] < \infty$, a memoryless deterministic, expectation-optimal scheduler σ such that $\mathbb{V}^{\sigma}_{\mathcal{M},s}[\oplus goal]$ is minimal among all expectation-optimal schedulers for any state s is computable in polynomial time.*

Variance-penalized expectation The goal of this section is to develop an algorithm to compute the optimal *variance-penalized expectation* (VPE). Given a rational $\lambda > 0$, we define the VPE with parameter λ under a scheduler σ as

$$\mathbb{VPE}[\lambda]^{\sigma}_{\mathcal{M}} = \mathbb{E}^{\sigma}_{\mathcal{M}}(\oplus goal) - \lambda \cdot \mathbb{V}^{\sigma}_{\mathcal{M}}(\oplus goal) = \mathbb{E}^{\sigma}_{\mathcal{M}}(\oplus goal) - \lambda \cdot \mathbb{E}^{\sigma}_{\mathcal{M}}(\oplus goal^2) + \lambda \cdot (\mathbb{E}^{\sigma}_{\mathcal{M}}(\oplus goal))^2.$$

Task: Compute the maximal variance-penalized expectation

$$\mathbb{VPE}[\lambda]^{\max}_{\mathcal{M}} = \sup_{\sigma} \mathbb{VPE}[\lambda]^{\sigma}_{\mathcal{M}}$$

where the supremum ranges over all proper schedulers. Furthermore, compute an optimal scheduler σ with $\mathbb{VPE}[\lambda]^{\sigma}_{\mathcal{M}} = \mathbb{VPE}[\lambda]^{\max}_{\mathcal{M}}$.

We restrict ourselves to MDPs with a weight function $\mathsf{wgt}\colon S \times A \to \mathbb{N}$, i.e., we only consider MDPs with non-negative weights. As before, we are only interested in schedulers that reach the goal with probability 1. If the maximal expectation $\mathbb{E}^{\max}_{\mathcal{M}}(\oplus goal) < \infty$, it is well-known that in this case of non-negative weights, all end components of $\mathcal{M}$ are 0-end components [dA99, S1]. Hence, w.l.o.g., we can assume that $\mathcal{M}$ has no end components throughout this section by Lemma 15. In this case, $\oplus goal$ is defined on almost all paths under any scheduler. So, in particular the values $\mathbb{E}^{\sigma}_{\mathcal{M}}(\oplus goal)$ and $\mathbb{V}^{\sigma}_{\mathcal{M}}(\oplus goal)$ are defined for all schedulers σ. The main result concerning VPE is the following:

Theorem 19. *Given an MDP $\mathcal{M}$ and λ as above, the optimal value $\mathbb{VPE}[\lambda]^{\max}_{\mathcal{M}}$ and an optimal scheduler σ can be computed in exponential space. Given a rational ϑ, the threshold problem whether $\mathbb{VPE}[\lambda]^{\max}_{\mathcal{M}} \geq \vartheta$ is in NEXPTIME and EXPTIME-hard.*

To obtain the main result, we first prove that the maximal VPE is obtained by a deterministic scheduler. This result can then be used for the EXPTIME-hardness proof for the threshold problem. The key step to obtain the upper bounds of the main result is to show that optimal schedulers have to *minimize* the weight that is expected to still be accumulated after a computable bound of accumulated weight has been exceeded. We call such a bound a *saturation point*. Finally, we show how to utilize the saturation point result to solve the threshold problem and to compute the optimal VPE.

Remark 20. *In the formulation presented here, the goal is to maximize the expected accumulated weight with a penalty for the variance. All results and proofs in this section, however, hold analogously for the variant $\sup_\sigma -\mathbb{E}^\sigma_{\mathcal{M}}(\oplus goal) - \lambda \cdot \mathbb{V}^\sigma_{\mathcal{M}}(\oplus goal)$ of the maximal VPE in which the goal is to minimize the expected accumulated weight while receiving a penalty for the variance. In particular, the same saturation point works and optimal schedulers still have to minimize the expected accumulated weight as soon as the accumulated weight exceeds the saturation point.*

Related work. *Accumulated rewards.* In [Man71], a characterization of variance-minimal scheulders among the schedulers maximizing the expected accumulated weight in MDPs is given. Here, we provide a simpler proof based on the calculations of [Ver04]; we moreover show how to compute such schedulers in polynomial time. [Man71] also contains hints for a similar characterization of discounted reward, and developments for mean payoff. Another closely related work is [MT11] which study the following multi-objective problem for the accumulated weight in finite-horizon MDPs: given η, ν is there a scheduler achieving an expectation of at least η, and a variance of at most ν? This problem is shown to be NP-hard, and exact pseudo-polynomial time algorithm is given for the existence of a scheduler with expectation η and variance $\leq \nu$. Furthermore, pseudo-polynomial approximation algorithms are given for optimizing the expectation under a constraint on the variance, and optimizing the variance under a constraint on the expectation.

Discounted rewards. In [Jaq73], the author proves that memoryless *moment-optimal* schedulers exist for the discounted reward, that is, schedulers that maximize the expectation, minimize the variance, maximize the third moment, and so on. Moreover, an algorithm is described to compute such schedulers. In [Sob82], a formula for the variance of the discounted reward is given for memoryless schedulers and for the finite-horizon case, in MDPs and semi-MDPs. Variance-minimal schedulers among those maximizing the expected discounted reward until a target set is reached are studied in [WG15] for MDPs with varying discount factors. [Xia18a] presents a policy iteration algorithm to minimize variance of the discounted weight among schedulers achieving an expectation equal to a given constant.

Mean payoff. For mean payoff objectives, variance was studied in [Sob94] for memoryless schedulers, and algorithms were given to compute schedulers that achieve given bounds on the expectation and the variance [BCFK17]. The latter paper also considers the minimization of the variability, which is the average of the squared differences between the expected mean-payoff and each observed one-step reward. In [Kur87], the author considers optimizing the expected mean payoff and the average variance. Average variance is defined as the limsup of the variances of the partial sums. They show how to minimize average variance among ϵ-optimal schedulers for

the expected mean payoff. Policy iteration algorithms were given in [Xia16, Xia18b] to minimize variance or variability of the mean payoff (without constraints on the expectation).

Variance-penalized expectation. The VPE was studied for finite-horizon MDPs with terminal rewards in [Col97]. In [FKL89], this notion was studied for the expectation and the variability of both mean payoff and discounted rewards. [Xia20] presents a policy iteration algorithm converging against *local* optima for a similar measure.

Conclusion In our results, there remains a complexity gap between the EXPTIME-lower bounds and the exponential-space and NEXPTIME-upper bounds for the optimization of the VPE in MDPs with non-negative weights and the corresponding threshold problem, respectively.

Further, we restricted our attention to MDPs with non-negative weights. When allowing positive and negative weights, the key result, the existence of a saturation point, does not hold anymore. For conditional expectations and other problems relying on the existence of a saturation point, the switch to integer weights makes the problems even at least as hard as the Positivity problem for linear recurrence sequences, a number theoretic problem whose decidability has been open for many decades (see [PB20, Pir21]). The question whether such a hardness result for the threshold problem of the VPE, rendering decidability impossible without a breakthrough in number theory, can be established remains as future work.

Further possible directions of research include the investigation of the following multi-objective threshold problem: Given η and ν, is there a scheduler with expectation at least η and variance at most ν? As the variance treats good and bad outcomes symmetrically, replacing the variance in the VPE by a one-sided deviation measure, such as the lower semi-variance that only takes the outcomes worse than the expected value into account, constitutes another natural extension of this work.

3.2.3 Percentiles in the Multi-Dimensional Case

The classical setting in Markov decision processes considers optimizing the *expectation* of the cost, such the expected shortest path distance (as seen above), or other measures such as discounted sum. If we are risk-averse, we may consider variance as in the previous section; but we may also want to search instead for strategies that ensure that the performance measure along time is larger than a given value with high probability, i.e., a probability that exceeds a given threshold. See for example [FKR95] for a solution.

Some works on MDPs have been exploring several natural extensions of those problems. First, there is a series of works that investigate MDPs with multi-dimensional weights [CMH06, BBC$^+$14] rather than single-dimensional as it is traditionally the case. Multi-dimensional MDPs are useful to analyze systems with *multiple objectives* that are potentially conflicting and make the analysis of trade-offs necessary. For instance, we may want to build a control strategy that both ensures some good quality of service and, at the same time, minimizes the energy consumption of the system. Second, there are works that aim at synthesizing strategies which enforce *richer properties*. For example, we may want to construct a strategy that both ensures some minimal threshold with certainty (or probability one) together with a good expectation. See [BFRR14] for results in that direction.

Our results participate in this general effort by providing algorithms and complexity results on the synthesis of strategies that enforce *multiple percentile constraints*. We consider MDP $\mathcal{M} = (S, A, \delta)$ with multiple weight functions $\mathsf{wgt}_1, \ldots, \mathsf{wgt}_d$. A *multi-percentile query* and the associated synthesis problem is as follows: given a multi-dimensionally weighted MDP M, an initial state s_{init}, synthesize

a unique strategy σ such that it satisfies the conjunction of q constraints:

$$\mathcal{Q} \coloneqq \bigwedge_{i=1}^{q} \mathbb{P}^{\sigma}_{M,s_{init}}\big[f_{l_i} \geq v_i\big] \geq \alpha_i.$$

where each l_i refers to a dimension of the weight vectors (that is wgt_i), each v_i is a value threshold, and α_i is a probability threshold, and f is a payoff function. We consider classical payoff functions: sup, inf, limsup, liminf, mean-payoff, truncated sum and discounted sum.

Let us consider some examples. In an MDP that models a stochastic shortest path problem, we may want to obtain a strategy that ensures that the probability to reach the target within d time units exceeds 50 percent: this is a single-constraint percentile query. With a *multi-constraint percentile query*, we can impose richer properties on strategies, for instance, enforcing that the duration is less than d_1 in at least 50 percent of the cases, and less than d_2 in 95 percent of the cases, with $d_1 < d_2$. We may also consider percentile queries in multi-dimensional systems. If in the model, we add information about fuel consumption, we may also enforce that we arrive within d time units in 95 percent of the cases, and that in half of the cases the fuel consumption is below some threshold c.

Payoff Functions We are interested in quantitative evaluation of runs in weighted MDPs according to *payoff functions* among inf, sup, $\liminf$, $\limsup$, mean-payoff, total payoff (seen in previous sections), and discounted sum. For any infinite run $\rho = s_1 a_1 s_2 a_2 \dots$ and dimension $l \in \{1, \dots, d\}$, we define these functions as follows.

- $\inf_l(\rho) = \inf_{j \geq 1} w_l(a_j)$, $\sup_l(\rho) = \sup_{j \geq 1} w_l(a_j)$,
- $\liminf_l(\rho) = \liminf_{j \to \infty} w_l(a_j)$, $\limsup_l(\rho) = \limsup_{j \to \infty} w_l(a_j)$,
- $\underline{\mathsf{MP}}_l(\rho) = \liminf_{n \to \infty} \frac{1}{n} \sum_{j=1}^{n} w_l(a_j)$,
- $\overline{\mathsf{MP}}_l(\rho) = \limsup_{n \to \infty} \frac{1}{n} \sum_{j=1}^{n} w_l(a_j)$,
- $\mathsf{DS}_l^{\lambda_l}(\rho) = \sum_{j=1}^{\infty} \lambda_l^j \cdot w_l(a_j)$, with $\lambda_l \in]0,1[$ a rational discount factor,

Contributions. We study percentile problems for a range of classical payoff functions: we establish algorithms and prove complexity and memory bounds. Our algorithms can handle multi-constraint multi-dimensional queries, but we also study interesting subclasses, namely, multi-constraint single-dimensional queries, single-constraint queries, and other classes depending on the payoff functions. We present an overview of our results in Table 3.1. For all payoff functions but the discounted sum, they only require *polynomial time in the size of the model* when the query size is fixed. In most applications, the query size can be reasonably bounded while the model can be very large, which suggests that percentile problems could be solved efficiently in practice.

We give a list of contributions from [**?**, **?**] and highlight some links with related problems.

A) We show the PSPACE-hardness of the multiple reachability problem with exponential dependency on the query size only, and the PSPACE-completeness of the almost-sure case, refining the results of [EKVY08]. We also prove that in the case of *nested* target sets, the problem admits polynomial-time solution, and we use it to solve some of the multi-constraint percentile problems.

	Reachability	$f \in \mathcal{F}$	$\overline{\mathsf{MP}}$	$\underline{\mathsf{MP}}$	SP	ε-gap DS
single constraint	P[Put94]	P[CH09]	P[Put94]		$\mathrm{P}(M)\cdot\mathrm{P}_{ps}(\mathcal{Q})$[HK14] PSPACE-h [HK14]	$\mathrm{P}_{ps}(M,\mathcal{Q},\varepsilon)$ NP-h
dim = 1. multi const.	$\mathrm{P}(M)\cdot\mathrm{E}(\mathcal{Q})$ [EKVY08] PSPACE-h	P	P	$\mathrm{P}(M)\cdot\mathrm{E}(\mathcal{Q})$	$\mathrm{P}(M)\cdot\mathrm{P}_{ps}(\mathcal{Q})$ (1 target) PSPACE-h [HK14]	$\mathrm{P}_{ps}(M,\varepsilon)\cdot\mathrm{E}(\mathcal{Q})$ NP-h
dim > 1. n const.	—	$\mathrm{P}(M)\cdot\mathrm{E}(\mathcal{Q})$ PSPACE-h	P	$\mathrm{P}(M)\cdot\mathrm{E}(\mathcal{Q})$	$\mathrm{P}(M)\cdot\mathrm{E}(\mathcal{Q})$ PSPACE-h [HK14]	$\mathrm{P}_{ps}(M,\varepsilon)\cdot\mathrm{E}(\mathcal{Q})$ PSPACE-h

Table 3.1: Overview of some of our results for percentile queries. Here $\mathcal{F} = \{\inf, \sup, \liminf, \limsup\}$. SP stands for shortest path, and DS for discounted sum. Parameters M and $\mathcal{Q}$ resp. represent the size of the model and the size of the query; $\mathrm{P}(x)$, $\mathrm{E}(x)$ and $\mathrm{P}_{ps}(x)$ resp. denote polynomial, exponential and pseudo-polynomial time in parameter x. All results without reference are our contributions.

B) For payoff functions inf, sup, lim inf and lim sup, we establish a polynomial-time algorithm for the single-dimension case, and an algorithm that is only exponential in the size of the query for the general case. We prove the PSPACE-hardness of the problem for sup, and derive a polynomial time algorithm for lim sup.

C) In the mean-payoff case, we distinguish $\overline{\mathsf{MP}}$ defined by the limsup of the average weights, and $\underline{\mathsf{MP}}$ by their liminf. For the former, we give a polynomial-time algorithm for the general case. For the latter, our algorithm is polynomial in the model size and exponential in the query size.

D) The total payoff function computes the *sum* of weights until a target is reached and is used to model *shortest path* problems. We prove the multi-dimensional percentile problem to be undecidable when both negative and positive weights are allowed. Therefore, we concentrate on the case of non-negative weights, and establish an algorithm that is polynomial in the model size and exponential in the query size. We derive from recent results that even the single-constraint percentile problem is PSPACE-hard [HK14].

E) The discounted sum case turns out to be difficult, and linked to a long-standing open problem, not known to be decidable. Nevertheless, we give algorithms for an approximation of the problem, called ε-gap percentile problem. Our algorithm guarantees correct answers up to an arbitrarily small zone of uncertainty. We also prove that this ε-gap problem is PSPACE-hard in general, and already NP-hard for single-constraint queries.

In all cases, we also study the memory requirement for strategies satisfying percentile queries.

We build our algorithms using different techniques. Here are a few tools we exploit. For inf and sup payoff functions, we reduce percentile queries to multiple reachability queries, and rely on the algorithm of [EKVY08]. For lim inf, lim sup and $\overline{\mathsf{MP}}$, we additionally need to resort to maximal end-component decomposition of MDPs. For $\underline{\mathsf{MP}}$, we use linear programming techniques to characterize winning strategies, borrowing ideas from [EKVY08, BBC+14]. For shortest path and discounted sum, we consider unfoldings of the MDP, with particular care to bound their sizes, and for the latter, to analyze the cumulative error due to necessary roundings.

Related work There are several works in the literature that study multi-dimensional MDPs: for discounted sum, see [CMH06], and for mean-payoff, see [BBC+14, FKR95]. In the latter papers, the following threshold problem is studied in multi-dimensional MDPs: given a threshold vector $\vec{v}$ and a probability threshold ν, does there exist a strategy σ such that $\mathbb{P}^{\sigma}_{\mathcal{M},s}[\vec{r} \geq \vec{v}] \geq \nu$. The work [FKR95] solves this problem for the single dimensional case, and the multi-dimensional for the *non-degenerate* case (referring to the solutions of a linear program). A general algorithm was later given in [BBC+14]. This problem asks for a bound on the *joint probability* of the thresholds, that is, the probability of satisfying *all* constraints simultaneously. In contrast, in our problem we bound the *marginal probabilities* separately, which may allow for more modeling flexibility. The problem of maximizing the *expectation vector* was also solved in [BBC+14].

Multiple reachability objectives in MDPs were considered in [EKVY08]: given an MDP and multiple targets T_i, thresholds α_i, decide if there exists a strategy that forces each T_i with a probability larger than α_i. This work is the closest to our work and we show here that their problem is inter-reducible with our problem for the sup measure. In [EKVY08] the complexity results are given only for the size of the model and not for the size of the query: we refine those results here and answer questions that were left open in that paper.

Several works consider percentile queries but only for *one* dimension and *one* constraint (while we consider multiple constraints and possibly multiple dimensions) and particular payoff functions. Single-constraint queries for lim sup and lim inf were studied in [CH09]. The treshold probability problem for truncated sum was studied in MDPs with either all non-negative or all non-positive weights in [Oht04, SO13]. The related notion of *quantile queries* for the shortest path over non-negatively weighted MDPs was studied in [UB13] in the single-constraint case. It has been recently extended to *cost problems* [HK14], in a direction orthogonal to ours. For fixed finite horizon, [XM11] considers the problem of ensuring a single-contraint percentile query for the discounted sum, and that of maximizing the expected value subject to a single percentile constraint. Still for the discounted case, there is a series of works studying *threshold problems* [Whi93, WL99] and *value-at-risk problems* [BCF+13]. All can be related to single-constraint percentiles queries.

3.3 Optimality under Uncertainties: Multiple-Environment MDPs

Given an MDP, we call the probabilistic transition function an *environment*. This terminology comes from the following point of view: when the system under study is not probabilistic, but the probabilistic behavior is due to the environmet in which it is executed (such as an external communication medium), then the probabilities are a model of the environment. In this setting, it is thus possible to study a given system under various environments. Environments are also known as *models* in the reinforcement learning literature.

In an MDP, the environment is *unique*, and this may not be realistic: we may want to design a control strategy that exhibits good performances under several hypotheses formalized by different models for the environment, and those environments may *not* be distinguishable or we may *not want* to distinguish them (e.g. because it is too costly to design several control strategies.) As an illustration, consider the design of guidelines for a medical treatment that needs to work adequately for two populations of patients (each given by a different stochastic model), even if the patients cannot be diagnosed to be in one population or in the other. An appropriate model for this case would be an MDP with two different models for the responses of the patients to the sequence of actions taken during the cure. We want a therapy that possibly makes decisions by observing the reaction of the patient and that works well (say reaches a good state for the patient with high

probability) no matter if the patient belongs to the first of the second population.

Facing two potentially indistinguishable environments can be easily modelled with a partially observable MDPs. Unfortunately, this model is particularly intractable [CCT13] (e.g. qualitative and quantitative reachability, safety and parity objectives are undecidable.) To remedy to this situation, in [S26], we introduce *multiple-environment MDPs (MEMDP)* which are MDPs with a *set* of probabilistic transition functions, rather than a *single one*. The goal in a MEMDP is to synthesize a single controller with guaranteed performances against *all* environments even though the environment at play is unknown a priori (it may be discovered during interaction but not necessarily.) We show that verification problems that are undecidable for partially observable MDPs, are decidable for MEMDPs and sometimes have even efficient solutions.

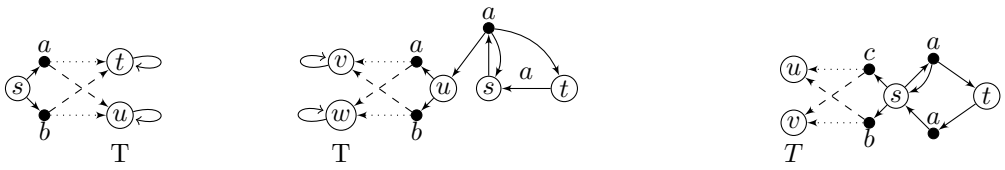


Figure 3.3: We adopt the following notation in all examples: edges that only exist in M_1 are drawn in dashed lines, and those that only exist in M_2 by dotted ones. To see that randomization may be necessary, observe that in the MEMDP M in Fig. 3.3a, the vector $(0.5, 0.5)$ of reachability probabilities for target T can only be achieved by a strategy that randomizes between a and b. In the MEMDP in Fig. 3.3b, where action a from s has the same support in M_1 and M_2 but different distributions. Any strategy almost surely reaches u in both M_i, since action a from s has nonzero probability of leading to u. Intuitively, the best strategy is to sample the distribution of action a from s, and to choose, upon arrival to u, either a or b according to the most probable environment. We prove that such an infinite-memory strategy achieves a Pareto-optimal vector which cannot be achieved by any finite-memory strategy. Last, in Fig. 3.3c, the MEMDP is similar to that of Fig. 3.3b except that action a from s only leads to s or t. We prove that for any $\epsilon > 0$, there exists a strategy ensuring reaching T with probability $1 - \epsilon$ in each M_i. The strategy consists in sampling the distribution of action a from s a sufficient number of times and estimating the actual environment against which the controller is playing. However, the vector $(1, 1)$ is not achievable.

Results We study MEMDPs with three types of objectives: reachability, safety and parity objectives. For each of those objectives, we study both *qualitative* and *quantitative* threshold decision problems.

We show that winning strategies may need infinite memory as well as randomization, and we provide algorithms to solve the decision problems. This can be seen in Figure 3.3. As it is classical, we consider two variants for the qualitative threshold problems. For simplicity, we limit our study to MEMDPs with two environments only. The first variant, asks to determine the existence of a *single* strategy that wins the objective with probability one (almost surely winning) in all the environments of the MEMDP. The second variant asks to determine the existence of a family of strategies such that for all $\epsilon > 0$, there is one strategy in the family that wins the objective with probability larger than $1 - \epsilon$ (limit sure winning) in all the environments of the MEMDP. For both almost sure winning and limit sure winning, and for all three types of objectives, we provide efficient polynomial-time algorithmic solutions. Then we turn to the quantitative threshold problem that asks for the existence of a single strategy that wins the objective with a probability that exceeds a

given rational threshold in all the environments. We show the problem to be NP-hard (already for two environments and acyclic MEMDPs), and so classical quantitative analysis techniques based on LP cannot be applied here. Instead, we show that finite memory strategies are sufficient to approach achievable thresholds and we reduce the existence of bounded memory strategies to solving quadratic equations, leading to solutions in polynomial space.

Related Work In addition to partially observable MDPs, our work is related to the following research lines.

Interval Markov chains are Markov chains in which transition probabilities are only known to belong to given intervals (see *e.g.* [KU02, KS05, CHK13]). Similarly, Markov decision processes with uncertain transition matrices for finite-horizon and discounted cases were considered [NEG05]. The latter work also mentions the finite scenario-case which is similar to our setting. However, the precise distributions of actions at each round are assumed to be independent while in our work we consider it to be fixed but unknown. Independence is a *simplifying assumption* that only provides pessimistic guarantees. However this approach does not use the information one obtains on the system along observed histories, and so the results tend to be overly pessimistic.

Our work is related to reinforcement learning, where the goal is to develop strategies which ensure good performance in unknown environments, by learning and optimizing simultaneously [SB18]. In particular, it is related to the multi-armed bandit problem where one is given a set of *stateless* systems with unknown reward distributions, and the goal is to choose the best one while optimizing the overall cost incured while learning. The problem of finding the optimal one (without optimizing) with high confidence was considered in [EDMM02, MT04]. However, our problems differ from this one as in multi-armed bandit problem models of the bandits are unknown while our environments are known but we do not know a priori against which one we are playing.

MEMDPs are also related to multi-objective reachability in MDPs considered in [EKVY08], where a strategy is to be synthesized so as to ensure the reachability of a set of targets, each with a possibly different probability. If we allow multiple environments and possibly different reachability objectives for each environment, this problem can be reduced to reachability in MEMDPs. Note however that the general reachability problem is harder in MEMDPs; it is NP-hard even for acyclic MEMDPs with absorbing targets, while polynomial-time algorithms exist for absorbing targets in the setting of [EKVY08].

Future Work Our study focused on the theoretical complexity of the optimal controller synthesis problems on MEMDPs. An extension of these results for an arbitrary number of environments is under way. For the almost-sure case, this has been done recently [vdVJJ23]. It would be interesting to extend these results to quantitative objectives such as the total payoff and mean payoff. This would extend the stochastic shortest path problem presented in previous sections to the multiple environments setting. An investigation of efficient algorithms to solve MEMDPs using reinforcement learning can be found in [CCK$^+$20].

4 Perspectives

The first set of contributions in this manuscript was on new model checking and synthesis algorithms for timed automata that target large discrete state spaces still being able to handle real-time constraints. We believe this is an important objective in the research on model checking of real-time systems. In fact, in many applications, the source of the state space explosion is the discrete state space rather than the number of clocks.

There are of course verification problems where an enumerative approach is well adapted for the discrete part, and the difficulty lies in the handling of real-time constraints. Due to the success of zone-based verification techniques and tools, the timed automata community has focused on benchmarks that are real-time protocols in which the discrete state space is modest, but the number of processes, the number of clocks, and the clock constraints are the source of complexity. Thus, protocols such as Fischer's mutual exclusion protocol [Lam87], and various models for CSMA/CD were used extensively, and have become a criterion to compare performances, as if the community's objective was to find efficient algorithms tailored for these particular protocols. Zone-based model checking tools such as Uppaal, and TChecker are indeed efficient in verifying such protocols, and are difficult to beat by symbolic algorithms; see for instance for nuXmv in [CGM+19].

In our work, our objective was to target other types of systems such as large synchronous systems within real-time environments, or distributed protocols, as above, but with a large number of discrete states in which real-time constraints are used to constrain the activation periods of the processes and their communication. Such systems were targeted before of course, *e.g.* [BCP+01] or in other algorithms attempting to develop a symbolic approach for timed automata. However, these have not integrated actively maintained model checking tools, and none of them have become a standard technique. We believe the community should make an effort to push timed automata verification outside of its comfort zone, which is the verification of real-time protocols of modest sizes. Further work is needed to develop algorithms robust enough so that they have predictable and convincing performance on particular types of systems with large state spaces. This would push the use of the timed automata technology further.

An important goal for future work is to develop algorithms that combine the efficient techniques used in zone-based algorithms (such as extrapolations), and symbolic techniques used in alternative algorithms, as presented in this manuscript. In fact, there is little overlap between the two approaches, which means that the community's effort on zone-based algorithms are not currently reused in symbolic algorithms; and vice versa. Looking for ways to combine the two approaches might allow us to go further in efficiency.

The second contribution presented in this chapter was on the handling of time imprecisions both in verification and synthesis: robustness verification, and robust controller synthesis problems. Both have the motivation of defining a more realistic notion of non-emptiness in the former problem, and controllability in the latter. This has thus a theoretical interest in its own; the theory was

studied before, *e.g.* [S9]. The question addressed in this chapter was on practical aspects: how can we handle robustness efficiently, and how do specialized algorithms improve over a treatment of robustness by modeling?

The robustness verification problems are not relevant for all verification tasks. In fact, models of systems under study are often abstracted significantly for model checking to scale, and detailed considerations about small clock imprecisions are not always relevant in such abstract models. Moreover, many case studies considered in the literature could actually use a discrete notion of time, conveniently and succinctly modeled by timed automata, so robustness, as we consider here, is not appropriate to consider. Yet, some distributed systems are indeed sensitive to small variations in the activation periods; *e.g.* an industrial case study is presented in [AFMS19] where timed automata techniques were used to deal with these real-time constraints. We believe that the robustness problems we solve here can further be relevant for analyzing a detailed low-level model of modest size, say, an infrared controller as studied in [BS13]. Analyzing a model with real-time constraints derived from the specification sheet of such a device might be sufficiently low-level so that the robustness questions make sense. By identifying such applications, one might find new uses for formal methods to solve non-trivial problems.

The application of formal methods to probabilistic systems have targeted rich sets of properties to be handled when synthesizing schedulers, and by the way uncertainties, modeled both with probabilities and nondeterminism are handled are taken into account. In fact, we believe that the contributions of the formal methods community is meaningful, and complementary to those from the discrete mathematics, planning, and reinforcement learning communities in the extent to which these help one 1) to work with schedulers with large memory requirements, both in theory and in practice; 2) to mix temporal logic properties with various quantitative objectives, either their expectations or their probabilities of satisfaction; 3) to handle nondeterministic uncertainties when analyzing systems, where the goal is not to optimize the average performance of a scheduler but to ensure that all schedulers have guaranteed average behaviors.

These results have the advantage of providing finer guarantees in the synthesized controllers. In fact, a single objective function to optimize is often insufficient to express complex requirements one might ask for the controller. One often needs to add temporal constraints with worst-case or average-case guarantees, and have finer control on the probability distribution of the objective values. Optimizing just the expectation might be insufficient; one might want to optimize the variance as well, have constraints on percentiles, and impose worst-case guarantees either on performance or on temporal properties that must be respected at all times. As for model checking, in practice, controller synthesis is difficult to apply in one-shot because the first objectives that are written might not capture the intent of the designer. Having a rich set of objectives and conditions helps the designer to refine the objectives so as to apply controller synthesis in real applications.

The study of multiple-environment MDPs participate in the effort of solving synthesis problems under nondeterministic uncertainties of the model. One could pursue this approach to target the analysis of systems with a clear separation of both types of uncertainties, and a mix of worst-case and probabilistic guarantees. A line of work [BFRR17], [S10] consists in giving such mix guarantees on quantitative objectives, that is, ensuring some performance guarantee in the worst-case, and other guarantees in average.

Another important direction is the development of data structures and solvers for handling such uncertainties, but also developing algorithms to work efficiently with schedulers with large memory requirements and various quantitative objectives. In many theoretical works, including some of ours, quantitative objectives such as total payoff are handled by encoding integer quantities in the state

space. Although such an approach might help achieve optimal worst-case complexity, this is not satisfactory in practice. Efforts are being made to classify the complexity of schedulers depending on the type and quantity of memory requirements [MM21]; these might lead to finer algorithms perhaps representing these quantities symbolically, and focusing the state space search on the control part of the model. In short, in the same way finite-state model checking has seen over the years several data structures and algorithms to handle larger and larger state spaces, perhaps an important objective for the community is the development of robust and efficient data structures and algorithms for quantitative verification over probabilistic systems that can handle various quantitative objectives and large schedulers, in order to be able to routinely apply this rich theory to real-world applications.

References to Author's Publications

[S1] Christel Baier, Nathalie Bertrand, Clemens Dubslaff, Daniel Gburek, and Ocan Sankur. Stochastic shortest paths and weight-bounded properties in markov decision processes. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '18, pages 86–94, New York, NY, USA, 2018. ACM.

[S2] Nicolas Basset, Gilles Geeraerts, Jean-François Raskin, and Ocan Sankur. Admissiblity in concurrent games. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, pages 123:1–123:14, 2017.

[S3] Nicolas Basset, Jean-François Raskin, and Ocan Sankur. Admissible strategies in timed games. In Luca Aceto, Giorgio Bacci, Giovanni Bacci, Anna Ingólfsdóttir, Axel Legay, and Radu Mardare, editors, *Models, Algorithms, Logics and Tools: Essays Dedicated to Kim Guldstrand Larsen on the Occasion of His 60th Birthday*, pages 403–425, Cham, 2017. Springer International Publishing.

[S4] Nathalie Bertrand, Nicolas Markey, Suman Sadhukhan, and Ocan Sankur. Dynamic network congestion games. In *40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020)*, Goa, India, December 2020.

[S5] Nathalie Bertrand, Nicolas Markey, Suman Sadhukhan, and Ocan Sankur. Semilinear representations for series-parallel atomic congestion games. In Anuj Dawar and Venkatesan Guruswami, editors, *Proceedings of the 42nd Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'22)*, volume 250 of *Leibniz International Proceedings in Informatics*. Leibniz-Zentrum für Informatik, December 2022.

[S6] Nathalie Bertrand, Nicolas Markey, Ocan Sankur, and Nicolas Waldburger. Parameterized Safety Verification of Round-Based Shared-Memory Systems. In Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming (ICALP 2022)*, volume 229 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 113:1–113:20, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[S7] Patricia Bouyer, Paul Gastin, Frédéric Herbreteau, Ocan Sankur, and B. Srivathsan. Zone-based verification of timed automata: Extrapolations, simulations and what next? In Sergiy Bogomolov and David Parker, editors, *Formal Modeling and Analysis of Timed Systems - 20th International Conference, FORMATS 2022, Warsaw, Poland, September 13-15, 2022, Proceedings*, volume 13465 of *Lecture Notes in Computer Science*, pages 16–42. Springer, 2022.

[S8] Patricia Bouyer, Nicolas Markey, and Ocan Sankur. Robust model-checking of timed automata via pumping in channel machines. In Uli Fahrenberg and Stavros Tripakis, editors, *Proceedings of the 9th International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS'11)*, volume 6919 of *Lecture Notes in Computer Science*, pages 97–112, Aalborg, Denmark, September 2011. Springer.

[S9] Patricia Bouyer, Nicolas Markey, and Ocan Sankur. Robustness in timed automata. In Parosh Aziz Abdulla and Igor Potapov, editors, *Proceedings of the 7th Workshop on Reachability Problems in Computational Models (RP'13)*, volume 8169 of *Lecture Notes in Computer Science*, pages 1–18, Uppsala, Sweden, September 2013. Springer.

[S10] Romain Brenguier, Lorenzo Clemente, Paul Hunter, Guillermo A. Pérez, Mickael Randour, Jean-François Raskin, Ocan Sankur, and Mathieu Sassolas. Non-zero sum games for reactive synthesis. In Adrian-Horia Dediu, Jan Janoušek, Carlos Martín-Vide, and Bianca Truthe, editors, *Language and Automata Theory and Applications: 10th International Conference (LATA 2016), Prague, Czech Republic, March 14-18, 2016*, volume 9618 of *Lecture Notes in Computer Science*, pages 3–23, Cham, 2016. Springer International Publishing.

[S11] Romain Brenguier, Stefan Göller, and Ocan Sankur. A comparison of succinctly represented finite-state systems. In Maciej Koutny and Irek Ulidowski, editors, *Proceedings of the 23rd International Conference on Concurrency Theory (CONCUR'12)*, volume 7454 of *Lecture Notes in Computer Science*, pages 147–161, Newcastle, UK, September 2012. Springer.

[S12] Romain Brenguier, Arno Pauly, Jean-François Raskin, and Ocan Sankur. Admissibility in Games with Imperfect Information (Invited Talk). In Roland Meyer and Uwe Nestmann, editors, *28th International Conference on Concurrency Theory (CONCUR 2017)*, volume 85 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 2:1–2:23, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[S13] Romain Brenguier, Guillermo A. Pérez, Jean-François Raskin, and Ocan Sankur. Abssynthe: abstract synthesis from succinct safety specifications. In Krishnendu Chatterjee, Rüdiger Ehlers, and Susmit Jha, editors, *Proceedings 3rd Workshop on Synthesis (SYNT'14)*, volume 157 of *Electronic Proceedings in Theoretical Computer Science*, pages 100–116. Open Publishing Association, 2014.

[S14] Romain Brenguier, Guillermo A. Pérez, Jean-François Raskin, and Ocan Sankur. Admissibility in quantitative graph games. In *36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, Chennai, India, December 2016.

[S15] Romain Brenguier, Guillermo A. Pérez, Jean-François Raskin, and Ocan Sankur. Compositional algorithms for succinct safety games. In Pavol Černý, Viktor Kuncak, and Madhusudan Parthasarathy, editors, *Proceedings Fourth Workshop on Synthesis (SYNT'15), San Francisco, CA, USA, 18th July 2015*, volume 202 of *Electronic Proceedings in Theoretical Computer Science*, pages 98–111. Open Publishing Association, 2016.

[S16] Romain Brenguier, Jean-François Raskin, and Ocan Sankur. Assume-admissible synthesis. *Acta Informatica*, 54(1):41–83, 2017.

[S17] Isseïnie Calviac, Ocan Sankur, and François Schwarzentruber. Improved complexity results and an efficient solution for connected multi-agent path finding. In *22nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS'23)*, May 2023. To appear.

[S18] Tristan Charrier, Arthur Queffelec, Ocan Sankur, and François Schwarzentruber. Complexity of planning for connected agents. *Auton. Agents Multi Agent Syst.*, 34(2):44, 2020.

[S19] Tristan Charrier, Arthur Queffelec, Ocan Sankur, and François Schwarzentruber. Reachability and coverage planning for connected agents. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pages 144–150, 2019.

[S20] Swen Jacobs, Roderick Bloem, Romain Brenguier, Rüdiger Ehlers, Timotheus Hell, Robert Könighofer, Guillermo A. Pérez, Jean-François Raskin, Leonid Ryzhyk, Ocan Sankur, Martina Seidl, Leander Tentrup, and Adam Walker. The first reactive synthesis competition (syntcomp 2014). *International Journal on Software Tools for Technology Transfer*, pages 1–24, 2016.

[S21] Thierry Jéron, Nicolas Markey, David Mentré, Reiya Noguchi, and Ocan Sankur. Incremental methods for checking real-time consistency. In *18th International Conference on Formal Modeling and Analysis of Timed Systems, FORMATS 2020.*, 2020.

[S22] Reiya Noguchi, Ocan Sankur, Thierry Jéron, Nicolas Markey, and David Mentré. Repairing real-time requirements. In Ahmed Bouajjani, Lukáš Holík, and Zhilin Wu, editors, *Proceedings of the 20th International Symposium on Automated Technology for Verification and Analysis (ATVA'22)*, Lecture Notes in Computer Science. Springer-Verlag, October 2022. To appear.

[S23] Youssouf Oualhadj, Pierre-Alain Reynier, and Ocan Sankur. Probabilistic robust timed games. In Paolo Baldan and Daniele Gorla, editors, *Proceedings of the 25th International Conference on Concurrency Theory (CONCUR'14)*, volume 8704 of *Lecture Notes in Computer Science*, pages 203–217. Springer, 2014.

[S24] Jakob Piribauer, Ocan Sankur, and Christel Baier. The Variance-Penalized Stochastic Shortest Path Problem. In Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming (ICALP 2022)*, volume 229 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 129:1–129:19, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[S25] Arthur Queffelec, Ocan Sankur, and François Schwarzentruber. Planning for Connected Agents in a Partially Known Environment. In *AI 2021 - 34th Canadian Conference on Artificial Intelligence*, pages 1–23, Vancouver / Virtual, Canada, May 2021.

[S26] Jean-Francois Raskin and Ocan Sankur. Multiple-Environment Markov Decision Processes. In Venkatesh Raman and S. P. Suresh, editors, *34th International Conference on Foundation of Software Technology and Theoretical Computer Science (FSTTCS 2014)*, volume 29 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 531–543. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2014.

[S27] Victor Roussanaly, Ocan Sankur, and Nicolas Markey. Abstraction refinement algorithms for timed automata. In Isil Dillig and Serdar Tasiran, editors, *Computer Aided Verification (CAV'19)*, pages 22–40, Cham, 2019. Springer International Publishing.

[S28] Ocan Sankur. Symbolic quantitative robustness analysis of timed automata. In Christel Baier and Cesare Tinelli, editors, *Tools and Algorithms for the Construction and Analysis of Systems (TACAS'15)*, volume 9035 of *Lecture Notes in Computer Science*, pages 484–498. Springer Berlin Heidelberg, 2015.

[S29] Ocan Sankur. Timed Automata Verification and Synthesis via Finite Automata Learning. In *TACAS 2023 - 29th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Paris, France, April 2023.

[S30] Ocan Sankur, Patricia Bouyer, Nicolas Markey, and Pierre-Alain Reynier. Robust controller synthesis in timed automata. In Pedro R. D'Argenio and Hernán Melgratti, editors, *Proceedings of the 24th International Conference on Concurrency Theory (CONCUR'13)*, volume 8052 of *Lecture Notes in Computer Science*, pages 546–560, Buenos Aires, Argentina, August 2013. Springer.

[S31] Ocan Sankur and Jean-Pierre Talpin. An abstraction technique for parameterized model checking of leader election protocols: Application to FTSP. In *Tools and Algorithms for the Construction and Analysis of Systems - 23rd International Conference, TACAS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings, Part I*, pages 23–40, 2017.

[S32] Bastien Thomas and Ocan Sankur. PyLTA: A Verification Tool for Parameterized Distributed Algorithms. In *TACAS 2023 - 29th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Paris, France, April 2023.

Bibliography

[AD94] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[ADBA21] Mohamadreza Ahmadi, Anushri Dixit, Joel W Burdick, and Aaron D Ames. Risk-averse stochastic shortest path planning. *arXiv:2103.14727*, 2021.

[AFMS19] Étienne André, Laurent Fribourg, Jean-Marc Mota, and Romain Soulat. Verification of an industrial asynchronous leader election algorithm using abstractions and parametric model checking. In Constantin Enea and Ruzica Piskac, editors, *VMCAI*, Lecture Notes in Computer Science, pages 409–424. Springer, 2019.

[AMP95] Eugene Asarin, Oded Maler, and Amir Pnueli. Symbolic controller synthesis for discrete and timed systems. In *Hybrid Systems II*, volume 999 of *LNCS*, pages 1–20. Springer, 1995.

[Ang87] Dana Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, 1987.

[Arr70] Kenneth J. Arrow. *Essays in the Theory of Risk-Bearing*. Amsterdam, North-Holland Pub. Co., 1970.

[AT05] Karine Altisen and Stavros Tripakis. Implementation of timed automata: An issue of semantics or modeling? In Paul Pettersson and Wang Yi, editors, *Proceedings of the 3rd International Conferences on Formal Modelling and Analysis of Timed Systems, (FORMATS'05)*, volume 3829 of *Lecture Notes in Computer Science*, pages 273–288. Springer, 2005.

[BA11] Nicolas Basset and Eugene Asarin. Thin and thick timed regular languages. In Uli Fahrenberg and Stavros Tripakis, editors, *Formal Modeling and Analysis of Timed Systems*, volume 6919 of *Lecture Notes in Computer Science*, pages 113–128. Springer, 2011.

[BBC+14] Tomáš Brázdil, Václav Brozek, Krishnendu Chatterjee, Vojtech Forejt, and Antonín Kucera. Markov decision processes with multiple long-run average objectives. *Logical Methods in Computer Science*, 10(13):1–29, 2014.

[BBE+10] Tomás Brázdil, Václav Brozek, Kousha Etessami, Antonín Kucera, and Dominik Wojtczak. One-counter Markov decision processes. In *Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 863–874. SIAM, 2010.

[BBLP06] Gerd Behrmann, Patricia Bouyer, Kim G. Larsen, and Radek Pelánek. Lower and upper bounds in zone-based abstractions of timed automata. *International Journal on Software Tools for Technology Transfer*, 8(3):204–215, June 2006.

[BCD+07a] Gerd Behrmann, Agnes Cougnard, Alexandre David, Emmanuel Fleury, Kim G Larsen, and Didier Lime. Uppaal-tiga: Time for playing games! In *International Conference on Computer Aided Verification*, pages 121–125. Springer, 2007.

[BCD+07b] Gerd Behrmann, Agnès Cougnard, Alexandre David, Emmanuel Fleury, Kim Guldstrand Larsen, and Didier Lime. UPPAAL-Tiga: Time for playing games! In *Proc. 19th International Conference on Computer Aided Verification (CAV'07)*, volume 4590 of *Lecture Notes in Computer Science*, pages 121–125. Springer, 2007.

[BCF+13] Tomás Brázdil, Taolue Chen, Vojtech Forejt, Petr Novotný, and Aistis Simaitis. Solvency Markov decision processes with interest. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2013, December 12-14, 2013, Guwahati, India*, pages 487–499, 2013.

[BCFK17] Tomáš Brázdil, Krishnendu Chatterjee, Vojtěch Forejt, and Antonín Kučera. Trading performance for stability in Markov decision processes. *Journal of Computer and System Sciences*, 84:144–170, 2017.

[BCP+01] V. Bertin, E. Closse, M. Poize, J. Pulou, J. Sifakis, P. Venier, D. Weil, and S. Yovine. Taxys=esterel+kronos. a tool for verifying real-time properties of embedded systems. In *Proceedings of the 40th IEEE Conference on Decision and Control (Cat. No.01CH37228)*, volume 3, pages 2875–2880 vol.3, 2001.

[BDD+14] Christel Baier, Marcus Daum, Clemens Dubslaff, Joachim Klein, and Sascha Klüppelholz. Energy-utility quantiles. In *Proc. of the 6th NASA Formal Methods Symposium (NFM)*, volume 8430 of *Lecture Notes in Computer Science*, pages 285–299. Springer, 2014.

[BDL+06a] Gerd Behrmann, Alexandre David, Kim Guldstrand Larsen, John Håkansson, Paul Pettersson, Wang Yi, and Martijn Hendriks. Uppaal 4.0. 2006.

[BDL+06b] Gerd Behrmann, Alexandre David, Kim Guldstrand Larsen, John Håkansson, Paul Pettersson, Wang Yi, and Martijn Hendriks. UPPAAL 4.0. In *Third International Conference on the Quantitative Evaluation of Systems (QEST 2006), 11-14 September 2006, Riverside, California, USA*, pages 125–126, 2006.

[BFRR14] Véronique Bruyère, Emmanuel Filiot, Mickael Randour, and Jean-François Raskin. Meet your expectations with guarantees: Beyond worst-case synthesis in quantitative games. In Ernst W. Mayr and Natacha Portier, editors, *31st International Symposium on Theoretical Aspects of Computer Science (STACS 2014), STACS 2014, March 5-8, 2014, Lyon, France*, volume 25 of *LIPIcs*, pages 199–213. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2014.

[BFRR17] Véronique Bruyère, Emmanuel Filiot, Mickael Randour, and Jean-François Raskin. Meet your expectations with guarantees: Beyond worst-case synthesis in quantitative games. *Information and Computation*, 254:259–295, 2017.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.

[BKKW17] Christel Baier, Joachim Klein, Sascha Klüppelholz, and Sascha Wunderlich. Maximizing the conditional expected reward for reaching the goal. In Axel Legay and Tiziana Margaria, editors, *23rd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 10206 of *Lecture Notes in Computer Science*, pages 269–285. Springer, 2017.

[BKN16] Tomás Brázdil, Antonín Kucera, and Petr Novotný. Optimizing the expected mean payoff in energy Markov decision processes. In *14th International Symposium on Automated Technology for Verification and Analysis (ATVA)*, volume 9938 of *Lecture Notes in Computer Science*, pages 32–49, 2016.

[BLN03] Dirk Beyer, Claus Lewerentz, and Andreas Noack. Rabbit: A tool for BDD-based verification of real-time systems. In *Proc. 15th International Conference on Computer Aided Verification (CAV'03)*, volume 2725 of *Lecture Notes in Computer Science*, pages 122–125. Springer, 2003.

[BLR11] Thomas Ball, Vladimir Levin, and Sriram K Rajamani. A decade of software model checking with slam. *Communications of the ACM*, 54(7):68–76, 2011.

[BM83] Bernard Berthomieu and Miguel Menasche. An enumerative approach for analyzing time Petri nets. In R. E. A. Mason, editor, *Information Processing 83 – Proceedings of the 9th IFIP World Computer Congress (WCC'83)*, pages 41–46. North-Holland/IFIP, September 1983.

[BMR06] Patricia Bouyer, Nicolas Markey, and Pierre-Alain Reynier. Robust model-checking of linear-time properties in timed automata. In José R. Correa, Alejandro Hevia, and Marcos Kiwi, editors, *Proceedings of the 7th Latin American Symposium on Theoretical INformatics (LATIN'06)*, volume 3887 of *Lecture Notes in Computer Science*, pages 238–249. Springer, 2006.

[BMR08] Patricia Bouyer, Nicolas Markey, and Pierre-Alain Reynier. Robust analysis of timed automata via channel machines. In Roberto Amadio, editor, *Proceedings of the 11th International Conference on Foundations of Software Science and Computation Structure (FoSSaCS'08)*, volume 4962 of *Lecture Notes in Computer Science*, pages 157–171. Springer, 2008.

[BS13] Timothy Bourke and Arcot Sowmya. Analyzing an embedded sensor with timed automata in Uppaal. *ACM Transactions on Embedded Computing Systems*, 13(3):Article 44, December 2013.

[BT91] Dimitri P. Bertsekas and John N. Tsitsiklis. An analysis of stochastic shortest path problems. *Mathematics of Operations Research*, 16(3):580–595, 1991.

[BY03] Johan Bengtsson and Wang Yi. Timed automata: Semantics, algorithms and tools. In *Advanced Course on Petri Nets*, pages 87–124. Springer, 2003.

[BY04] Johan Bengtsson and Wang Yi. Timed automata: Semantics, algorithms and tools. In Jörg Desel, Wolfgang Reisig, and Grzegorz Rozenberg, editors, *Lectures on Concurrency*

and Petri Nets, volume 2098 of *Lecture Notes in Computer Science*, pages 87–124. Springer-Verlag, 2004.

[CC77] Patrick Cousot and Radhia Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 238–252, 1977.

[CCK+20] Krishnendu Chatterjee, Martin Chmelík, Deep Karkhanis, Petr Novotný, and Amélie Royer. Multiple-environment markov decision processes: Efficient analysis and applications. *Proceedings of the International Conference on Automated Planning and Scheduling*, 30(1):48–56, Jun. 2020.

[CCT13] Krishnendu Chatterjee, Martin Chmelik, and Mathieu Tracol. What is decidable about partially observable markov decision processes with omega-regular objectives. In *CSL*, volume 23 of *LIPIcs*. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013.

[CD11] Krishnendu Chatterjee and Laurent Doyen. Energy and mean-payoff parity Markov decision processes. In *Proc. of the 36th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, volume 6907 of *Lecture Notes in Computer Science*, pages 206–218. Springer, 2011.

[CDF+05a] Franck Cassez, Alexandre David, Emmanuel Fleury, Kim G. Larsen, and Lmei Didier. Efficient on-the-fly algorithms for the analysis of timed games. In *CONCUR 2005*, pages 66–80, London, UK, 2005. Springer-Verlag.

[CDF+05b] Franck Cassez, Alexandre David, Emmanuel Fleury, Kim Guldstrand Larsen, and Didier Lime. Efficient on-the-fly algorithms for the analysis of timed games. In *Proc. 16th International Conference on Concurrency Theory (CONCUR'05)*, volume 3653 of *Lecture Notes in Computer Science*, pages 66–80. Springer, 2005.

[CES09] Edmund M. Clarke, E. Allen Emerson, and Joseph Sifakis. Model checking: Algorithmic verification and debugging. *Commun. ACM*, 52(11):74–84, nov 2009.

[CGJ+00] Edmund Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement. In *International Conference on Computer Aided Verification*, pages 154–169. Springer, 2000.

[CGJ+03] Edmund Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement for symbolic model checking. *Journal of the ACM (JACM)*, 50(5):752–794, 2003.

[CGM+19] Alessandro Cimatti, Alberto Griggio, Enrico Magnago, Marco Roveri, and Stefano Tonetta. Extending nuxmv with timed transition systems and timed temporal properties. In *International Conference on Computer Aided Verification*, pages 376–386. Springer, 2019.

[CGMT14] Alessandro Cimatti, Alberto Griggio, Sergio Mover, and Stefano Tonetta. IC3 modulo theories via implicit predicate abstraction. In *Proc. 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'14)*, volume 8413 of *Lecture Notes in Computer Science*, pages 46–61, 2014.

[CH09] Krishnendu Chatterjee and Thomas A. Henzinger. Probabilistic systems with limsup and liminf objectives. In Margaret Archibald, Vasco Brattka, Valentin Goranko, and Benedikt Löwe, editors, *Infinity in Logic and Computation*, volume 5489 of *Lecture Notes in Computer Science*, pages 32–45. Springer Berlin Heidelberg, 2009.

[CHK13] Taolue Chen, Tingting Han, and Marta Z. Kwiatkowska. On the complexity of model checking interval-valued discrete time markov chains. *Inf. Process. Lett.*, 113(7):210–216, 2013.

[CHP11] Krishnendu Chatterjee, Thomas A. Henzinger, and Vinayak S. Prabhu. Timed parity games: Complexity and robustness. *Logical Methods in Computer Science*, 7(4), 2011.

[CHR02] Franck Cassez, Thomas A. Henzinger, and Jean-François Raskin. A comparison of control problems for timed and hybrid systems. In Claire Tomlin and Mark R. Greenstreet, editors, *Proceedings of the 5th International Workshop on Hybrid Systems: Computation and Control (HSCC'02)*, volume 2289 of *Lecture Notes in Computer Science*, pages 134–148. Springer, 2002.

[CHV+18] Edmund M Clarke, Thomas A Henzinger, Helmut Veith, Roderick Bloem, et al. *Handbook of model checking*, volume 10. Springer, 2018.

[CMH06] Krishnendu Chatterjee, Rupak Majumdar, and Thomas A. Henzinger. Markov decision processes with multiple objectives. In Bruno Durand and Wolfgang Thomas, editors, *STACS 2006*, volume 3884 of *Lecture Notes in Computer Science*, pages 325–336. Springer Berlin Heidelberg, 2006.

[Col97] EJ Collins. Finite-horizon variance penalised Markov decision processes. *Operations-Research-Spektrum*, 19(1):35–39, 1997.

[dA97] Luca de Alfaro. *Formal Verification of Probabilistic Systems*. PhD thesis, Stanford University, Department of Computer Science, 1997.

[dA99] Luca de Alfaro. Computing minimum and maximum reachability times in probabilistic systems. In *10th International Conference on Concurrency Theory (CONCUR)*, volume 1664 of *Lecture Notes in Computer Science*, pages 66–81, 1999.

[DDD+12] Werner Damm, Henning Dierks, Stefan Disch, Willem Hagemann, Florian Pigorsch, Christoph Scholl, Uwe Waldmann, and Boris Wirtz. Exact and fully symbolic verification of linear hybrid automata with large discrete state spaces. *Science of Computer Programming*, 77(10):1122–1150, 2012.

[DDMR08] Martin De Wulf, Laurent Doyen, Nicolas Markey, and Jean-François Raskin. Robust safety of timed automata. *Formal Methods in System Design*, 33(1-3):45–84, 2008.

[DDR05] Martin De Wulf, Laurent Doyen, and Jean-François Raskin. Almost ASAP semantics: From timed models to timed implementations. *Formal Aspects of Computing*, 17(3):319–341, 2005.

[DGDF07] Carole Delporte-Gallet, Stéphane Devismes, and Hugues Fauconnier. Robust stabilizing leader election. In Toshimitsu Masuzawa and Sébastien Tixeuil, editors, *Stabilization, Safety, and Security of Distributed Systems*, pages 219–233, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[Die06] Henning Dierks. *Time, abstraction and heuristics - automatic verification and planning of timed systems using abstraction and heuristics*, volume 01-06 of *Berichte aus dem Department für Informatik / Universität Oldenburg / Fachbereich Informatik*. 2006.

[DK06] Conrado Daws and Piotr Kordy. Symbolic robustness analysis of timed automata. In Eugene Asarin and Patricia Bouyer, editors, *Proceedings of the 4th International Conferences on Formal Modelling and Analysis of Timed Systems, (FORMATS'06)*, volume 4202 of *Lecture Notes in Computer Science*, pages 143–155. Springer, September 2006.

[DKL07] Henning Dierks, Sebastian Kupferschmid, and Kim G Larsen. Automatic abstraction refinement for timed automata. In *International Conference on Formal Modeling and Analysis of Timed Systems*, pages 114–129. Springer, 2007.

[dR10] Luca de Alfaro and Pritam Roy. Solving games via three-valued abstraction refinement. *Information and Computation*, 208(6):666–676, 2010. Special Issue: 18th International Conference on Concurrency Theory (CONCUR 2007).

[EDMM02] Eyal Even-Dar, Shie Mannor, and Yishay Mansour. Pac bounds for multi-armed bandit and markov decision processes. In *COLT'02*, volume 2375 of *LNCS*, pages 255–270. Springer, 2002.

[EFGP10] Rudiger Ehlers, Daniel Fass, Michael Gerke, and Hans-Jorg Peter. Fully symbolic timed model checking using constraint matrix diagrams. In *Proc. 31th IEEE Real-Time Systems Symposium (RTSS'10)*, pages 360–371. IEEE Computer Society Press, 2010.

[EKVY08] Kousha Etessami, Marta Z. Kwiatkowska, Moshe Y. Vardi, and Mihalis Yannakakis. Multi-objective model checking of Markov decision processes. *Logical Methods in Computer Science*, 4(4), 2008.

[FKL89] Jerzy A Filar, Lodewijk CM Kallenberg, and Huey-Miin Lee. Variance-penalized Markov decision processes. *Mathematics of Operations Research*, 14(1):147–161, 1989.

[FKR95] Jerzy A. Filar, Dmitry Krass, and Kirsten W. Ross. Percentile performance criteria for limiting average Markov decision processes. *Automatic Control, IEEE Transactions on*, 40(1):2–10, Jan 1995.

[GBGE14] William N Goetzmann, Stephen J Brown, Martin J Gruber, and Edwin J Elton. *Modern portfolio theory and investment analysis*. John Wiley & Sons, 2014.

[GGL13] G. Geeraerts, J. Goossens, and M. Lindström. Multiprocessor schedulability of arbitrary-deadline sporadic tasks: Complexity and antichain algorithm. *Real-Time Systems, The International Journal of Time-Critical Computing Systems*, 48(2), 2013.

[GHJ97] Vineet Gupta, Thomas A. Henzinger, and Radha Jagadeesan. Robust timed automata. In *Proc. International Workshop on Hybrid and Real-Time Systems (HART'97)*, volume 1201 of *Lecture Notes in Computer Science*, pages 331–345. Springer, 1997.

[GS97] Susanne Graf and Hassen Saidi. Construction of abstract state graphs with pvs. In *International Conference on Computer Aided Verification*, pages 72–83. Springer, 1997.

[HJ94] Hans Hansson and Bengt Jonsson. A logic for reasoning about time and reliability. *Formal aspects of computing*, 6(5):512–535, 1994.

[HJMM04] Thomas A Henzinger, Ranjit Jhala, Rupak Majumdar, and Kenneth L McMillan. Abstractions from proofs. In *Proceedings of the 31st ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 232–244, 2004.

[HJMS02] Thomas A Henzinger, Ranjit Jhala, Rupak Majumdar, and Gregoire Sutre. Lazy abstraction. In *Proceedings of the 29th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 58–70, 2002.

[HJMS03] Thomas A Henzinger, Ranjit Jhala, Rupak Majumdar, and Gregoire Sutre. Software verification with blast. In *International SPIN Workshop on Model Checking of Software*, pages 235–239. Springer, 2003.

[HK14] Christoph Haase and Stefan Kiefer. The odds of staying on budget. *CoRR*, abs/1409.8228, 2014.

[HK15] Christoph Haase and Stefan Kiefer. The odds of staying on budget. In *Proc. of the 42nd International Colloquium on Automata, Languages, and Programming (ICALP), Part II*, volume 9135 of *Lecture Notes in Computer Science*, pages 234–246. Springer, 2015.

[HKSW11] Frédéric Herbreteau, Dileep Kini, B. Srivathsan, and Igor Walukiewicz. Using non-convex approximations for efficient analysis of timed automata. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2011, December 12-14, 2011, Mumbai, India*, pages 78–89, 2011.

[HNSY94] Thomas A Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Symbolic model checking for real-time systems. *Information and computation*, 111(2):193–244, 1994.

[HP] Frédéric Herbreteau and Gérald Point. The TChecker tool and librairies. `https://github.com/ticktac-project/tchecker`.

[HS06] Thomas A. Henzinger and Joseph Sifakis. The embedded systems design challenge. In *Formal Methods, 14th International Symposium on Formal Methods*, volume 4085 of *Lecture Notes in Computer Science*, pages 1–15, Hamilton, Canada, 2006. Springer.

[HSTW20] Frédéric Herbreteau, B Srivathsan, Thanh-Tung Tran, and Igor Walukiewicz. Why liveness for timed automata is hard, and what we can do about it. *ACM Transactions on Computational Logic (TOCL)*, 21(3):1–28, 2020.

[HSW13] Frédéric Herbreteau, Balaguru Srivathsan, and Igor Walukiewicz. Lazy abstractions for timed automata. In *International Conference on Computer Aided Verification*, pages 990–1005. Springer, 2013.

[HZH+10] Fei He, He Zhu, William NN Hung, Xiaoyu Song, and Ming Gu. Compositional abstraction refinement for timed systems. In *2010 4th IEEE International Symposium on Theoretical Aspects of Software Engineering*, pages 168–176. IEEE, 2010.

[IHS14] Malte Isberner, Falk Howar, and Bernhard Steffen. The ttt algorithm: a redundancy-free approach to active automata learning. In *International Conference on Runtime Verification*, pages 307–322. Springer, 2014.

[IHS15] Malte Isberner, Falk Howar, and Bernhard Steffen. The open-source learnlib. In Daniel Kroening and Corina S. Păsăreanu, editors, *Computer Aided Verification*, pages 487–495, Cham, 2015. Springer International Publishing.

[Jaq73] Stratton C. Jaquette. Markov Decision Processes with a New Optimality Criterion: Discrete Time. *The Annals of Statistics*, 1(3):496 – 505, 1973.

[JPA+22] Swen Jacobs, Guillermo A Perez, Remco Abraham, Veronique Bruyere, Michael Cadilhac, Maximilien Colange, Charly Delfosse, Tom van Dijk, Alexandre Duret-Lutz, Peter Faymonville, et al. The reactive synthesis competition (syntcomp): 2018-2021. *arXiv preprint arXiv:2206.00251*, 2022.

[JR11a] Rémi Jaubert and Pierre-Alain Reynier. Quantitative robustness analysis of flat timed automata. In *Proceedings of the 14th international conference on Foundations of software science and computational structures: part of the joint European conferences on theory and practice of software*, volume 6604 of *Lecture Notes in Computer Science*, pages 229–244. Springer, 2011.

[JR11b] Rémi Jaubert and Pierre-Alain Reynier. Quantitative robustness analysis of flat timed automata. In *Proceedings of the 14th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'11)*, volume 6604 of *Lecture Notes in Computer Science*, pages 229–244. Springer, 2011.

[KA06] Branislav Kusy and Sherif Abdelwahed. Ftsp protocol verification using spin. May 2006.

[Kal11] Lodewijk Kallenberg. *Markov Decision Processes*. Lecture Notes. University of Leiden, 2011.

[KJN11] Roland Kindermann, Tommi Junttila, and Ilkka Niemela. Modeling for symbolic analysis of safety instrumented systems with clocks. In *Proc. 11th International Conference on Application of Concurrency to System Design (ACSD'11)*, pages 185–194. IEEE Computer Society Press, 2011.

[KLMP14] Piotr Kordy, Rom Langerak, Sjouke Mauw, and Jan Willem Polderman. A symbolic algorithm for the analysis of robust timed automata. In Cliff Jones, Pekka Pihlajasaari, and Jun Sun, editors, *FM 2014: Formal Methods*, volume 8442 of *Lecture Notes in Computer Science*, pages 351–366. Springer, 2014.

[KM18] Jan Kretínský and Tobias Meggendorfer. Conditional value-at-risk for reachability and mean payoff in Markov decision processes. In *33rd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 609–618. ACM, 2018.

[KS05] Antonín Kučera and Oldřich Stražovský. On the controller synthesis for finite-state markov decision processes. In *FSTTCS 2005*, volume 3821 of *LNCS*, pages 541–552. Springer, 2005.

[KSBD15] Daniel Krähmann, Jana Schubert, Christel Baier, and Clemens Dubslaff. Ratio and weight quantiles. In *40th International Conference on Mathematical Foundations of Computer Science 2015 Symposium (MFCS), Part I*, volume 9234 of *Lecture Notes in Computer Science*, pages 344–356. Springer, 2015.

[KU02] Igor O Kozine and Lev V Utkin. Interval-valued finite markov chains. *Reliable computing*, 8(2):97–113, 2002.

[Kur87] Masami Kurano. Markov decision processes with a minimum-variance criterion. *Journal of mathematical analysis and applications*, 123(2):572–583, 1987.

[LAL+14] Shang-Wei Lin, Étienne André, Yang Liu, Jun Sun, and Jin Song Dong. Learning assumptions for compositional verification of timed systems. *Transactions on Software Engineering*, 40(2):137–153, mar 2014.

[Lam87] Leslie Lamport. A fast mutual exclusion algorithm. *ACM Trans. Comput. Syst.*, 5(1):1–11, jan 1987.

[Lam05] Leslie Lamport. Real-time model checking is really simple. In *Advanced Research Working Conference on Correct Hardware Design and Verification Methods*, pages 162–175. Springer, 2005.

[LLTW11] Kim G. Larsen, Axel Legay, Louis-Marie Traonouez, and Andrzej Wasowski. Robust specification of real time components. In Uli Fahrenberg and Stavros Tripakis, editors, *Formal Modeling and Analysis of Timed Systems*, volume 6919 of *Lecture Notes in Computer Science*, pages 129–144. Springer, 2011.

[LLTW14] Kim G. Larsen, Axel Legay, Louis-Marie Traonouez, and Andrzej Wasowski. Robust synthesis for real-time systems. *Theor. Comput. Sci.*, 515:96–122, 2014.

[LOD+13] Alfons Laarman, Mads Chr. Olesen, Andreas Engelbredt Dalsgaard, Kim Guldstrand Larsen, and Jaco van de Pol. Multi-core emptiness checking of timed büchi automata using inclusion abstraction. In *CAV 2013*, volume 8044 of *Lecture Notes in Computer Science*, pages 968–983. Springer, 2013.

[LvBP94] Gang Luo, G. von Bochmann, and A. Petrenko. Test selection based on communicating nondeterministic finite-state machines using a generalized wp-method. *IEEE Transactions on Software Engineering*, 20(2):149–162, 1994.

[Man71] Petr Mandl. On the variance in controlled Markov chains. *Kybernetika*, 7(1):1–12, 1971.

[Mar52] Harry Markowitz. Portfolio selection. *The Journal of Finance*, 7(1):77–91, 1952.

[McI09] A. I. McInnes. Model-checking the flooding time synchronization protocol. In *Control and Automation, 2009. ICCA 2009. IEEE International Conference on*, pages 422–429, Dec 2009.

[MKSL04] Miklós Maróti, Branislav Kusy, Gyula Simon, and Ákos Lédeczi. The flooding time synchronization protocol. In *Proceedings of the 2Nd International Conference on Embedded Networked Sensor Systems*, SenSys '04, pages 39–49, New York, NY, USA, 2004. ACM.

[MM14] Oded Maler and Irini-Eleftheria Mens. Learning regular languages over large alphabets. In Erika Ábrahám and Klaus Havelund, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 485–499, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[MM17] Oded Maler and Irini-Eleftheria Mens. A generic algorithm for learning symbolic automata from membership queries. In Luca Aceto, Giorgio Bacci, Giovanni Bacci, Anna Ingólfsdóttir, Axel Legay, and Radu Mardare, editors, *Models, Algorithms, Logics and Tools: Essays Dedicated to Kim Guldstrand Larsen on the Occasion of His 60th Birthday*, pages 146–169, Cham, 2017. Springer International Publishing.

[MM21] Richard Mayr and Eric Munday. Strategy Complexity of Mean Payoff, Total Payoff and Point Payoff Objectives in Countable MDPs. In Serge Haddad and Daniele Varacca, editors, *32nd International Conference on Concurrency Theory (CONCUR 2021)*, volume 203 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 12:1–12:15, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[MPS95] Oded Maler, Amir Pnueli, and Joseph Sifakis. On the synthesis of discrete controllers for timed systems (an extended abstract). In *STACS*, pages 229–242, 1995.

[MSTW17] Richard Mayr, Sven Schewe, Patrick Totzke, and Dominik Wojtczak. MDPs with energy-parity objectives. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, IEEE Computer Society, pages 1–12, 2017.

[MT04] Shie Mannor and John N. Tsitsiklis. The sample complexity of exploration in the multi-armed bandit problem. *J. Mach. Learn. Res.*, 5:623–648, December 2004.

[MT11] Shie Mannor and John N. Tsitsiklis. Mean-variance optimization in Markov decision processes. In *Proceedings of the 28th International Conference on Machine Learning*, ICML'11, page 177–184, Madison, WI, USA, 2011. Omnipress.

[NEG05] Arnab Nilim and Laurent El Ghaoui. Robust control of markov decision processes with uncertain transition matrices. *Operations Research*, 53(5):780–798, 2005.

[NSL+12a] Truong Khanh Nguyen, Jun Sun, Yang Liu, Jin Song Dong, and Yan Liu. Improved BDD-based discrete analysis of timed systems. In *Proc. 20th International Symposium on Formal Methods (FM'12)*, volume 7436, pages 326–340. Springer, 2012.

[NSL+12b] Truong Khanh Nguyen, Jun Sun, Yang Liu, Jin Song Dong, and Yan Liu. Improved BDD-based discrete analysis of timed systems. In Dimitra Giannakopoulou and Dominique Méry, editors, *Proceedings of the 18th International Symposium on Formal Methods (FM'12)*, volume 7436 of *Lecture Notes in Computer Science*, pages 326–340. Springer-Verlag, August 2012.

[Oht04] Yoshio Ohtsubo. Optimal threshold probability in undiscounted Markov decision processes with a target set. *Applied Mathematics and Computation*, 149(2):519 – 532, 2004.

[PB19] Jakob Piribauer and Christel Baier. Partial and conditional expectations in Markov decision processes with integer weights. In Mikolaj Bojanczyk and Alex Simpson, editors,

22nd International Conference on Foundations of Software Science and Computation Structures (FoSSaCS), volume 11425 of *Lecture Notes in Computer Science*, pages 436–452. Springer, 2019.

[PB20] Jakob Piribauer and Christel Baier. On Skolem-hardness and saturation points in Markov decision processes. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 168 of *LIPIcs*, pages 138:1–138:17. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020.

[PEM11] Hans-Jörg Peter, Rüdiger Ehlers, and Robert Mattmüller. Synthia: Verification and synthesis for timed automata. In *International Conference on Computer Aided Verification*, pages 649–655. Springer, 2011.

[PGB+08] Corina S Păsăreanu, Dimitra Giannakopoulou, Mihaela Gheorghiu Bobaru, Jamieson M Cobleigh, and Howard Barringer. Learning to divide and conquer: applying the L* algorithm to automate assume-guarantee reasoning. *Formal Methods in System Design*, 32(3):175–205, 2008.

[Pir21] Jakob Piribauer. *On Non-Classical Stochastic Shortest Path Problems*. PhD thesis, Technische Universität Dresden, Germany, 2021.

[Pnu77] Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pages 46–57. ieee, 1977.

[Pra64] John W. Pratt. Risk aversion in the small and in the large. *Econometrica*, 32(1/2):122–136, 1964.

[Pur00] Anuj Puri. Dynamical properties of timed automata. *Discrete Event Dynamic Systems*, 10(1-2):87–113, 2000.

[Put94] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1994.

[RAB+95] Rajeev K Ranjan, Adnan Aziz, Robert K Brayton, Bernard Plessier, and Carl Pixley. Efficient bdd algorithms for fsm synthesis and verification. *IWLS95, Lake Tahoe, CA*, 253:254, 1995.

[RS93] R.L. Rivest and R.E. Schapire. Inference of finite automata using homing sequences. *Information and Computation*, 103(2):299–347, 1993.

[SB03] Sanjit A. Seshia and Randal E. Bryant. Unbounded, fully symbolic model checking of timed automata using boolean methods. In Warren A. Hunt and Fabio Somenzi, editors, *Computer Aided Verification*, pages 154–166, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[SB18] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[SF07] M. Swaminathan and M. Franzle. A symbolic decision procedure for robust safety of timed systems. In *Temporal Representation and Reasoning, 14th International Symposium on*, pages 192–192, June 2007.

[SFK08] Mani Swaminathan, Martin Fränzle, and Joost-Pieter Katoen. The surprising robustness of (closed) timed automata against clock-drift. In Giorgio Ausiello, Juhani Karhumäki, Giancarlo Mauri, and Luke Ong, editors, *Fifth Ifip International Conference On Theoretical Computer Science – Tcs 2008*, volume 273, pages 537–553. Springer, 2008.

[SO13] Masahiko Sakaguchi and Yoshio Ohtsubo. Markov decision processes associated with two threshold probability criteria. *Journal of Control Theory and Applications*, 11(4):548–557, 2013.

[Sob82] Matthew J. Sobel. The variance of discounted markov decision processes. *Journal of Applied Probability*, 19(4):794–802, 1982.

[Sob94] Matthew J. Sobel. Mean-variance tradeoffs in an undiscounted mdp. *Operations Research*, 42(1):175–183, 1994.

[TM15] Yann Thierry-Mieg. Symbolic model-checking using ITS-tools. In *Proc. 21st International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'15)*, pages 231–237. Springer, 2015.

[TM17] Tamás Tóth and István Majzik. Lazy reachability checking for timed automata using interpolants. In *International Conference on Formal Modeling and Analysis of Timed Systems*, pages 264–280. Springer, 2017.

[Tra16] Thanh-Tung Tran. *Verification of timed automata : reachability, liveness and modelling. (Vérification d'automates temporisés : sûreté, vivacité et modélisation)*. PhD thesis, University of Bordeaux, France, 2016.

[UB13] Michael Ummels and Christel Baier. Computing quantiles in Markov reward models. In *Proc. of the 16th International Conference on Foundations of Software Science and Computation Structures (FOSSACS)*, volume 7794 of *Lecture Notes in Computer Science*, pages 353–368. Springer, 2013.

[vdVJJ23] Marck van der Vegt, Nils Jansen, and Sebastian Junges. Robust almost-sure reachability in multi-environment mdps, 2023.

[Ver04] Tom Verhoeff. Reward variance in Markov chains: A calculational approach. In *Proceedings of Eindhoven FASTAR Days*. Citeseer, 2004.

[Wan01] Farn Wang. Symbolic verification of complex real-time systems with clock-restriction diagram. In *Proc. 21st International Conference on Formal Techniques for Networked and Distributed Systems (FORTE'01)*, volume 197 of *IFIP Conference Proceedings*, pages 235–250. Kluwer, 2001.

[WG15] Xiao Wu and Xianping Guo. First passage optimality and variance minimisation of Markov decision processes with varying discount factors. *Journal of Applied Probability*, 52(2):441–456, 2015.

[Whi93] Douglas J. White. Minimizing a threshold probability in discounted Markov decision processes. *Journal of Mathematical Analysis and Applications*, 173(2):634 – 646, 1993.

[WL99] Congbin Wu and Yuanlie Lin. Minimizing risk models in Markov decision processes with policies depending on target values. *Journal of Mathematical Analysis and Applications*, 231(1):47–67, 1999.

[Xia16] Li Xia. Optimization of Markov decision processes under the variance criterion. *Automatica*, 73:269–278, 2016.

[Xia18a] Li Xia. Mean–variance optimization of discrete time discounted Markov decision processes. *Automatica*, 88:76–82, 2018.

[Xia18b] Li Xia. Variance minimization of parameterized Markov decision processes. *Discrete Event Dynamic Systems*, 28(1):63–81, 2018.

[Xia20] Li Xia. Risk-sensitive Markov decision processes with combined metrics of mean and variance. *Production and Operations Management*, 29(12):2808–2827, 2020.

[XM11] Huan Xu and Shie Mannor. Probabilistic goal Markov decision processes. In *IJCAI*, pages 2046–2052, 2011.