Model and Objective Separation with Conditional Lower Bounds: Disjunction is Harder than Conjunction

Krishnendu Chatterjee[1], Wolfgang Dvořák[2], Monika Henzinger[2], and Veronika Loitzenbauer[2]

[1]IST Austria
[2]University of Vienna, Faculty of Computer Science, Vienna, Austria

May 9, 2016

Abstract

Given a model of a system and an objective, the model-checking question asks whether the model satisfies the objective. We study polynomial-time problems in two classical models, graphs and Markov Decision Processes (MDPs), with respect to several fundamental ω-regular objectives, e.g., Rabin and Streett objectives. For many of these problems the best-known upper bounds are quadratic or cubic, yet no super-linear lower bounds are known. In this work our contributions are two-fold: First, we present several improved algorithms, and second, we present the first conditional super-linear lower bounds based on widely believed assumptions about the complexity of CNF-SAT and combinatorial Boolean matrix multiplication. A separation result for two models with respect to an objective means a conditional lower bound for one model that is strictly higher than the existing upper bound for the other model, and similarly for two objectives with respect to a model. Our results establish the following separation results: (1) A separation of models (graphs and MDPs) for disjunctive queries of reachability and Büchi objectives. (2) Two kinds of separations of objectives, both for graphs and MDPs, namely, (2a) the separation of dual objectives such as reachability/safety (for disjunctive questions) and Streett/Rabin objectives, and (2b) the separation of conjunction and disjunction of multiple objectives of the same type such as safety, Büchi, and coBüchi. In summary, our results establish the first model and objective separation results for graphs and MDPs for various classical ω-regular objectives. Quite strikingly, we establish conditional lower bounds for the disjunction of objectives that are strictly higher than the existing upper bounds for the conjunction of the same objectives.

1 Introduction

The fundamental problem in formal verification is the *model-checking* question that given a model of a system and a property asks whether the model satisfies the property. The model can be, for example, a standard graph, or a probabilistic extension of graphs, and the

property describes the desired behaviors (or infinite paths) of the model. For several basic model-checking questions, though polynomial-time algorithms are known, the best-known existing upper bounds are quadratic or cubic, yet no super-linear lower bounds are known. In graph algorithmic problems unconditional super-linear lower bounds are very rare when polynomial-time solutions exist. However, recently there have been many interesting results that establish *conditional lower bounds* [3, 5, 1]. These are lower bounds based on the assumption that for some well-studied problem such as 3-SUM [24] or All-Pairs Shortest Paths [40, 36] no (polynomially[1]) faster algorithm exists (compared to the best known algorithm). The lower bounds in this work assume (A1) there is no combinatorial[2] algorithm with running time of $O(n^{3-\varepsilon})$ for any $\varepsilon > 0$ to multiply two $n \times n$ Boolean matrices; or (A2) for all $\varepsilon > 0$ there exists a k such that there is no algorithm for the k-CNF-SAT problem that runs in $2^{(1-\varepsilon)\cdot n} \cdot \text{poly}(m)$ time, where n is the number of variables and m the number of clauses. These two assumptions have been used to establish lower bounds for several well-studied problems, such as dynamic graph algorithms [3, 5], measuring the similarity of strings [4, 10, 11, 7, 2], context-free grammar parsing [34, 1], and verifying first-order graph properties [35, 43]. No relation between conjectures (A1) and (A2) is known. In this work we present conditional lower bounds that are super-linear for fundamental model-checking problems.

Models. The two most classical models in formal verification are *standard graphs* and *Markov decision processes* (*MDPs*). MDPs are probabilistic extensions of graphs, and an MDP consists of a finite directed graph (V, E) with a partition of the vertex set V into player 1 vertices V_1 and random vertices V_R and a probabilistic transition function that specifies for vertices in V_R a probability distribution over their successor vertices. Let $n = |V|$ and $m = |E|$. An infinite path in an MDP is obtained by the following process. A token is placed on an initial vertex and the token is moved indefinitely as follows: At a vertex $v \in V_1$ a choice is made to move the token along one of the outedges of v, and at a vertex $v \in V_R$ the token is moved according to the probabilistic transition function. Note that if $V_R = \emptyset$, then we have a standard graph, and if $V_1 = \emptyset$, then we have a Markov chain. Thus MDPs generalize standard graphs and Markov chains.

Objectives. Objectives (or properties) are subsets of infinite paths that specify the desired set of paths. The most basic objective is *reachability* where, given a set $T \subseteq V$ of *target* vertices, an infinite path satisfies the objective if the path visits a vertex of T *at least once*. The dual objective to reachability is *safety* where, given a set $T \subseteq V$ of *target* vertices, an infinite path satisfies the objective if the path does *not* visit any vertex of T. The next extension of a reachability objective is the *Büchi* objective that requires the set of target vertices to be reached *infinitely often*. Its dual, the *coBüchi* objective, requires the set of target vertices to be reached only *finitely often*. A natural extension of single objectives are *conjunctive* and *disjunctive objectives* [23, 44, 18]. For two objectives ψ_1 and ψ_2 their conjunctive objective is equal to $\psi_1 \cap \psi_2$ and their disjunctive objective is equal to $\psi_1 \cup \psi_2$. The conjunction of reachability (resp. Büchi) objectives is known as *generalized reachability* (*resp. Büchi*) [23, 44]. A very central and canonical class of objectives in formal verification are *Streett* (strong fairness) objectives

[1] In particular improvements by polylogarithmic factors are not excluded.

[2] Combinatorial here means avoiding fast matrix multiplication [33], see also the discussion in [27].

and their dual *Rabin* objectives [39]. A *one-pair Streett* objective for two sets of vertices L and U specifies that if the Büchi objective for target set L is satisfied, then also the Büchi objective for target set U has to be satisfied; in other words, a one-pair Streett objective is the disjunction of a coBüchi objective (with target set L) and a Büchi objective (with target set U). The dual *one-pair Rabin* objective for two vertex sets L and U is the conjunction of a Büchi objective with target set L and a coBüchi objective with target set U. A Streett objective is the conjunction of k one-pair Streett objectives and its dual Rabin objective is the disjunction of k one-pair Rabin objectives.

Algorithmic questions. The algorithmic question given a model and an objective is as follows: (a) for standard graphs, the model-checking question asks whether there is a path that satisfies the objective; and (b) for MDPs, the basic model-checking question asks whether there is a *policy* (or a *strategy* that resolves the non-deterministic choices of outgoing edges) for player 1 to ensure that the objective is satisfied with probability 1. Observe that if we consider the model-checking question for MDPs with $V_R = \emptyset$, then it exactly corresponds to the model-checking question for standard graphs. Given k objectives, the *conjunctive query* question asks whether there is a policy for player 1 to ensure that *all* the objectives are satisfied with probability 1, and the *disjunctive query* question asks whether there is a policy for player 1 to ensure that *one* of the objectives is satisfied with probability 1. Conjunctive queries coincide with conjunctive objectives on graphs and MDPs, while disjunctive queries coincide with disjunctive objectives on graphs but not on MDPs (see Observations 2.1 and 2.2).

Significance of model and objectives. Standard graphs are the model for non-deterministic systems, and provide the framework to model hardware and software systems [29, 20], as well as many basic logic-related questions such as automata emptiness. MDPs model systems with both non-deterministic and probabilistic behavior; and provide the framework for a wide range of applications from randomized communication and security protocols, to stochastic distributed systems, to biological systems [32, 8]. In verification, reachability objectives are the most basic objectives for safety-critical systems. In general all properties that arise in verification (such as liveness, fairness) are ω-regular languages (ω-regular languages extend regular languages to infinite words), and every ω-regular language can be expressed as a Streett objective (or a Rabin objective). Important special cases of Streett (resp. Rabin) objectives are Büchi and coBüchi objectives [16]. Thus the algorithmic questions we consider are the most basic questions in formal verification.

Model separation and objective separation questions. In this work our results (upper and conditional lower bounds) aim to establish the following two fundamental separations:

- *Model separation.* Consider an objective where the algorithmic question for both graphs and MDPs can be solved in polynomial time, and establish a conditional lower bound for MDPs that is strictly higher than the best-known upper bound for graphs. In other words, the conditional lower bound would separate the model of graphs and MDPs for problems (i.e., w.r.t. the objective) that can be solved in polynomial time.

- *Objective separation.* Consider a model (either graphs or MDPs) with two different objectives and show that, though the algorithmic question for both objectives can be

solved in polynomial time, there is a conditional lower bound for one objective that is strictly higher than the best-known upper bound for the other objective. In other words, the conditional lower bound would separate the two objectives w.r.t. the model though they both can be solved in polynomial time.

To the best of our knowledge, there is no previous work that establish any model separation or objective separation result in the literature.

Our results. In this work we present improved algorithms as well as the first conditional lower bounds that are super-linear for algorithmic problems in model checking that can be solved in polynomial time, and together they establish both model separation and objective separation results. An overview of the results for the different objectives is given in Table 1, where our results are highlighted in boldface. We use MEC to refer to the time to compute the maximal end-component decomposition of an MDP. An end-component is a (non-trivial) strongly connected sub-MDP for which random vertices have no edges out of the component. We have $\text{MEC} = O(\min(n^2, m^{1.5}))$ [16] and assume $\text{MEC} = \Omega(m)$ and $m \geq n$. Moreover, we use k to denote the number of combined objectives in the case of conjunction or disjunction of multiple objectives and b to denote the total number of elements in all the target sets that specify the objectives. We first describe Table 1 and our main results and then discuss the significance of our results for model and objective separation.

1. *Conjunctive and Disjunctive Reachability (and Büchi) Problems.* First, we consider conjunctive and disjunctive reachability objectives and queries. Recall that conjunctive objectives and queries in general and disjunctive objectives and queries on graphs coincide. For reachability further the disjunctive objective can be reduced to a single objective (see Observation 2.3). The following results are known: the algorithmic question for conjunctive reachability objectives is NP-complete for graphs [13], and PSPACE-complete for MDPs [23]; and the disjunctive objective can be solved in linear time for graphs and in $O(\min(n^2, m^{1.5}) + b)$ time in MDPs [19, 16]. We present three results for disjunctive reachability queries in MDPs: (i) We present an $O(km + \text{MEC})$-time algorithm[3]. (ii) We show that under assumption (A1) there does not exist a combinatorial $O(k \cdot n^{2-\varepsilon})$ algorithm for any $\varepsilon > 0$. (iii) We show that for $k = \Omega(m)$ there does not exist an $O(m^{2-\varepsilon})$ time algorithm for any $\varepsilon > 0$ under assumption (A2). Hence we establish an upper bound and matching conditional lower bounds based on (A1) and (A2).

   Disjunctive Büchi objectives (on graphs and MDPs) can be reduced in linear time to disjunctive reachability objectives and vice versa, therefore the same results apply to disjunctive Büchi problems (see Observation 2.6). The basic algorithm for conjunctive Büchi objectives runs in time $O(m + b)$ on graphs and in time $O(\text{MEC} + b)$ on MDPs.

2. *Conjunctive and Disjunctive Safety Problems.* Second, we consider conjunctive and disjunctive safety objectives and queries. The following results are known: the conjunctive problem can be reduced to a single objective and can be solved in linear time, both in

[3]This implies an $O(\text{MEC} + b)$-time algorithm for disjunctive objective but does not improve the running time for this case.

Table 1: Upper and lower bounds. Our results are boldface and respective results are referred.

		Graphs		MDPs	
		upper bound	lower bound*	upper bound	lower bound*
Reach	Conj.	NP-c [13]		PSPACE-c [23]	
	Disj. Obj.	$\Theta(m+b)$		$O(\text{MEC}+b)$ [19, 16]	
	Disj. Qu.	$\Theta(m+b)$		$\mathbf{O(k \cdot m + MEC)}$	$\mathbf{k \cdot n^{2-o(1)}}$, $\mathbf{m^{2-o(1)}}$
Safety	Conj.	$\Theta(m+b)$		$\Theta(m+b)$	
	Disj. Obj.	$O(k \cdot m)$	$\mathbf{k \cdot n^{2-o(1)}}$	PSPACE-c [23]	
	Disj. Qu.	$O(k \cdot m)$	$\mathbf{k \cdot n^{2-o(1)}}$	$O(k \cdot m)$	$\mathbf{k \cdot n^{2-o(1)}}$, $\mathbf{m^{2-o(1)}}$
Büchi	Conj.	$\Theta(m+b)$		$O(\text{MEC}+b)$	
	Disj. Obj.	$\Theta(m+b)$		$O(\text{MEC}+b)$ [19, 16]	
	Disj. Qu.	$\Theta(m+b)$		$\mathbf{O(k \cdot m + MEC)}$	$\mathbf{k \cdot n^{2-o(1)}}$, $\mathbf{m^{2-o(1)}}$
coBüchi	Conj.	$\Theta(m+b)$		$O(\text{MEC}+b)$	
	Disj. Obj.	$O(k \cdot m)$	$\mathbf{k \cdot n^{2-o(1)}}$	$\mathbf{O(k \cdot m + MEC)}$	$\mathbf{k \cdot n^{2-o(1)}}$, $\mathbf{m^{2-o(1)}}$
	Disj. Qu.	$O(k \cdot m)$	$\mathbf{k \cdot n^{2-o(1)}}$	$\mathbf{O(k \cdot m + MEC)}$	$\mathbf{k \cdot n^{2-o(1)}}$, $\mathbf{m^{2-o(1)}}$
Singleton	Disj. Obj.	$\mathbf{\Theta(m)}$			$\mathbf{m^{2-o(1)}}$
	Disj. Qu.	$\mathbf{\Theta(m)}$			$\mathbf{m^{2-o(1)}}$
Streett		$O(\min(n^2, m\sqrt{m \log n}, km) + b \log n)$ [28, 17]		$\mathbf{O(\min(n^2, m\sqrt{m \log n}) + b \log n)}$	
Rabin		$O(k \cdot m)$	$\mathbf{k \cdot n^{2-o(1)}}$	$O(k \cdot \text{MEC})$	$\mathbf{k \cdot n^{2-o(1)}}$, $\mathbf{m^{2-o(1)}}$

* $\mathbf{k \cdot n^{2-o(1)}}$ lower bounds are based on the BMM Conjecture / Strong Triangle Conjecture (A1)
$\mathbf{m^{2-o(1)}}$ lower bounds are based on the Orthogonal Vectors Conjecture / Strong ETH (A2)

graphs and MDPs (see e.g. [14]); disjunctive queries for MDPs can be solved in $O(k \cdot m)$ time; and disjunctive objectives for MDPs are PSPACE-complete [23]. We present two results: (i) We show that for the disjunctive problem in graphs under assumption (A1) there does not exist a combinatorial $O(k \cdot n^{2-\varepsilon})$ algorithm for any $\varepsilon > 0$. This implies the same conditional lower bound for disjunctive queries and objectives in MDPs and matches the upper bound for graphs and disjunctive queries in MDPs. (ii) We present, for $k = \Omega(m)$, an $\Omega(m^{2-o(1)})$ lower bound for disjunctive objectives and queries in MDPs under assumption (A2). Again this lower bound matches the upper bound of $O(k \cdot m)$ for disjunctive queries.

3. *Conjunctive and Disjunctive coBüchi Problems.* For coBüchi, a conjunctive objective can be reduced to a single objective. For single objectives the basic algorithm runs in time $O(\text{MEC}+b)$ on MDPs and in time $O(m+b)$ on graphs. Since the conditional lower bounds for disjunctive safety objectives and queries actually already apply for the non-emptiness of the winning set, the reductions also hold for coBüchi (see Observation 2.5). Here the running times and the conditional lower bounds are matching for both disjunctive queries and disjunctive objectives. For the conditional lower bound based on assumption (A2)

Table 2: Model Separation.

	upper bound Graphs	lower bounds MDPs
Reach/Büchi Disj. Qu.	$m + nk$	$\mathbf{k \cdot n^{2-o(1)}, m^{2-o(1)}}$
coBüchi Singleton Disj. Obj./Qu.	$\mathbf{m}$	$\mathbf{m^{2-o(1)}}$

Table 3: Dual Objective Separation for Graphs.

	upper bound	lower bound	
Reach Disj.	$m + nk$	$\mathbf{k \cdot n^{2-o(1)}}$	Safety Disj.
Büchi Disj.	$m + nk$	$\mathbf{k \cdot n^{2-o(1)}}$	coBüchi Disj.
Büchi Conj.	$m + nk$	$\mathbf{k \cdot n^{2-o(1)}}$	coBüchi Disj.
Streett	$n^2 + nk \log n$	$\mathbf{k \cdot n^{2-o(1)}}$	Rabin

Table 4: Dual Objective Separation for MDPs.

	upper bound	lower bound	
Büchi Disj. Obj.	$\min(n^2, m^{1.5}) + nk$	$\mathbf{k \cdot n^{2-o(1)}, m^{2-o(1)}}$	coBüchi Disj. Obj.
Büchi Conj.	$\min(n^2, m^{1.5}) + nk$	$\mathbf{k \cdot n^{2-o(1)}, m^{2-o(1)}}$	coBüchi Disj. Obj.
Streett	$\mathbf{\min(n^2, m\sqrt{m \log n}) + nk \log n}$	$\mathbf{k \cdot n^{2-o(1)}, m^{2-o(1)}}$	Rabin

only *singleton coBüchi* objectives, i.e., coBüchi objectives with target sets of cardinality one, are needed, therefore the bound already holds for this case. We additionally present two results: (i) We present $O(km + \text{MEC})$-time algorithms for disjunctive queries and objectives in MDPs. (ii) We present a linear time algorithm for disjunctive singleton coBüchi objectives in graphs.

4. *Rabin and Streett objectives.* Finally, we consider Rabin and Streett objectives. The basic algorithm for Rabin objectives runs in time $O(k \cdot m)$ on graphs and in time $O(k \cdot \text{MEC})$ on MDPs. As disjunctive coBüchi objectives are a special case of Rabin objectives, the conditional lower bounds for coBüchi objectives of $\Omega(k \cdot n^{2-o(1)})$ on graphs and additionally $\Omega(m^{2-o(1)})$ on MDPs extend to Rabin objectives. The conditional lower bound for graphs is matching (for combinatorial algorithms). Furthermore, we extend the results of [28, 17] from graphs to MDPs to show that MDPs with Streett objectives can be solved in $O(\min(m\sqrt{m \log n}, n^2) + b \log n)$ time.

Significance of our results. We now describe the model and objective separation results that are obtained from the results we established.

1. *Model Separation.* Table 2 shows our results that separate graphs and MDPs regarding their complexity for certain objectives and queries under assumptions (A1) and (A2). First, for reachability and Büchi objectives disjunction in graphs is in linear time while

in MDPs we have $\Omega(kn^{2-o(1)})$ and $\Omega(m^{2-o(1)})$ conditional lower bounds for disjunctive queries. Second, for coBüchi we have a separation when restricted to the class where each target set is a singleton. For these objectives disjunction in graphs is in linear time while we establish an $\Omega(m^{2-o(1)})$ conditional lower bound for MDPs for both disjunctive objectives and queries.

2. *Objective Separation.* Further we identify complexity separations between different objectives. Here we consider two aspects, separations between dual objectives like Büchi and coBüchi (Tables 3 and 4), and separations between conjunction and disjunction of objectives (Table 5). We compare dual objectives in two ways: (i) we show that single objectives that are dual to each other behave differently when we consider disjunction for each of them and (ii) we compare conjunctive objectives and their dual disjunctive objectives. For (ii) we have that conjunctive Büchi objectives are dual to disjunctive coBüchi objectives, and Streett objectives, the conjunction of 1-pair Streett objectives, are dual to Rabin objectives, the disjunction of 1-pair Rabin objectives.

   (a) *Separating Dual Objectives in Graphs.* First, we consider reachability and safety objectives. In graphs we have that for reachability objectives disjunction is in linear time while for disjunctive safety objectives we establish an $\Omega(kn^{2-o(1)})$ lower bound under assumption (A1). Analogously, we have disjunctive Büchi objectives are in linear time on graphs while we establish an $\Omega(kn^{2-o(1)})$ conditional lower bound for disjunction of coBüchi objectives. Further, conjunctive Büchi objectives are in linear time and thus can be separated from their dual objective, the disjunctive coBüchi objectives. Finally, for Streett objectives in graphs with $b = O(n^2/\log n)$ we have an $O(n^2)$ algorithm while we establish an $\Omega(n^{3-o(1)})$ lower bound for Rabin objectives when $k = \Theta(n)$.

   (b) *Separating Dual Objectives in MDPs.* First, consider Büchi and coBüchi objectives in MDPs. On MDPs disjunctive Büchi objectives are in time $O(\text{MEC} + b)$, which is in $O(\min(n^2, m^{1.5}) + nk)$, while for coBüchi objectives we show $\Omega(kn^{2-o(1)})$ and $\Omega(m^{2-o(1)})$ conditional lower bounds for both disjunctive queries and disjunctive objectives. This separates the two objectives for both sparse and dense graphs. Further conjunctive Büchi objectives can be solved in $O(\text{MEC} + b)$ time and thus there is also a separation between disjunctive coBüchi objectives and their dual. Finally, for Streett objectives in MDPs with $b = O(\min(n^2, m^{1.5})/\log n)$ we show both an $O(n^2)$-time and an $O(m^{1.5})$-time algorithm while we establish $\Omega(n^{3-o(1)})$ and $\Omega(m^{2-o(1)})$ conditional lower bounds for Rabin objectives when $k = \Theta(n)$.

   (c) *Separating Conjunction and Disjunction in Graphs and MDPs.* Except for reachability, i.e., in particular for all considered polynomial-time problems, we observe that the disjunction of objectives is computationally harder than the conjunction of these objectives (under assumptions (A1), (A2)). First, for safety objectives conjunction is in linear time even for MDPs while for disjunctive queries (disjunctive objectives are PSPACE-complete) we present $\Omega(kn^{2-o(1)})$ and $\Omega(m^{2-o(1)})$ conditional lower bounds, where the first bound also holds for graphs. Second, for Büchi and coBüchi

Table 5: Separating Conjunction and Disjunction.

		Conjunction	Disjunction
Safety	Graphs	$m + nk$	$\mathbf{k \cdot n^{2-o(1)}}$
	MDP Qu.	$m + nk$	$\mathbf{k \cdot n^{2-o(1)}, m^{2-o(1)}}$
Büchi	MDPs Qu.	$\min(n^2, m^{1.5}) + nk$	$\mathbf{k \cdot n^{2-o(1)}, m^{2-o(1)}}$
coBüchi	Graphs	$m + nk$	$\mathbf{k \cdot n^{2-o(1)}}$
	MDPs Obj./Qu.	$\min(n^2, m^{1.5}) + nk$	$\mathbf{k \cdot n^{2-o(1)}, m^{2-o(1)}}$
1-pair Streett	Graphs	$n^2 + nk \log n$	$\mathbf{k \cdot n^{2-o(1)}}$
	MDPs Obj./Qu.	$\mathbf{\min(n^2, m\sqrt{m \log n}) + nk \log n}$	$\mathbf{k \cdot n^{2-o(1)}, m^{2-o(1)}}$
1-pair Rabin	Graphs	$m + nk$	$\mathbf{k \cdot n^{2-o(1)}}$
	MDPs Obj./Qu.	$\min(n^2, m^{1.5}) + nk$	$\mathbf{k \cdot n^{2-o(1)}, m^{2-o(1)}}$

* $\log n$ factor omitted

objectives conjunction is in $O(\text{MEC}+b)$ on MDPs (and $O(m+b)$ on graphs) while we show $\Omega(kn^{2-o(1)})$ and $\Omega(m^{2-o(1)})$ conditional lower bounds for disjunctive coBüchi objectives and disjunctive Büchi / coBüchi queries on MDPs. The $\Omega(m^{2-o(1)})$ bound even holds for the disjunction of singleton coBüchi objectives. Further, for coBüchi objectives our $\Omega(kn^{2-o(1)})$ bound also holds on graphs, which separates conjunction and disjunction also in this setting. Third, we can also see the results for Streett and Rabin objectives as a separation between conjunction and disjunction. Recall that Streett objectives are the conjunction of one-pair Streett objectives and Rabin objectives are the disjunction of one-pair Rabin objectives. Further, both Büchi and coBüchi objectives are special cases of each of one-pair Streett and one-pair Rabin objectives. In particular the following separations are easy observations or corollaries of our results: For the disjunction of one-pair Streett objectives the same conditional lower bounds (and the same upper bound, see Observation 6.10) as for the disjunction of coBüchi objectives apply. Thus the disjunction of one-pair Streett objectives is harder than the conjunction of one-pair Streett objectives (under assumptions (A1)/(A2)). The conjunction of one-pair Rabin objectives can be solved in the same time as conjunctive Büchi objectives. Thus also the disjunction of one-pair Rabin objectives is harder than their conjunction.

Remark about Streett and Rabin objective separation. One remarkable aspect of our objective separation result is that we achieve it for Rabin and Streett objectives (both in graphs and MDPs), which are dual. In more general models such as games on graphs, Rabin objectives are NP-complete and Streett objectives are coNP-complete [22]. In graphs and MDPs, both Rabin and Streett objectives can be solved in polynomial time. Since Rabin and Streett objectives are dual, and they belong to the complementary complexity classes (either both in P, or one is NP-complete, other coNP-complete), they were considered to be equivalent for algorithmic purposes for graphs and MDPs. Quite surprisingly we show that under some widely believed

assumptions, both for MDPs and graphs, Rabin objectives are algorithmically harder than Streett objectives.

Remark about separating conjunction and disjunction. In logic disjunction and conjunction are dual and for polynomial-time problems to the best of our knowledge there have not been any results which show that one is harder than the other. For reachability objectives the conjunctive problems are harder (NP-complete for graphs, PSPACE-complete for MDPs) compared to the disjunctive problems, which are in polynomial time. In terms of strategy complexity, again conjunctive objectives are harder than disjunctive objectives: While for disjunctive Büchi objectives memoryless strategies suffice, for conjunctive Büchi objectives strategies require memory; and while for Rabin objectives memoryless strategies suffice, for Streett objectives strategies require memory (even exponential memory in games). Given the existing results, there was no evidence to expect that when polynomial-time algorithms exist that disjunction is harder than conjunction. On the contrary, existing results show that some aspects of conjunctive objectives are harder than the disjunctive counterpart. Surprisingly, our results indicate that from an algorithmic point of view several polynomial-time problems are harder for the disjunctive problems than for their conjunctive counterparts.

Technical contributions.

Algorithms. (1) We show that given the MEC-decomposition of an MDP, the *almost-sure reachability problem* can be solved in linear time on the MDP where each MEC is contracted to a player 1 vertex. This yields to the improved algorithms for disjunctive queries of reachability and Büchi objectives on MDPs. (2) For *MDPs* with *disjunctive coBüchi* objectives and disjunctive queries of coBüchi objectives we use the MEC-decomposition in a different way; namely, we show that it is sufficient to do a linear-time computation in each MEC per coBüchi objective to solve both disjunctive questions. (3) Further we show that for *graphs* with a *disjunctive coBüchi* objective for which the target set of each of the single coBüchi objectives has *cardinality one* the problem can be solved with a breadth-first search like algorithm in linear time. (4) Finally, we provide faster algorithms for *MDPs with Streett objectives*. The straight-forward algorithm repeatedly computes MEC-decompositions in a black-box manner; we show that one can open this black-box and combine the current best algorithms for MEC-decomposition [16] and graphs with Streett objectives [28, 17] to achieve almost the same running time for MDPs with Streett objectives as for graphs.

Conditional Lower Bounds. (a) Conjecture (A1) is equivalent to the conjecture that there is no combinatorial $O(n^{3-\varepsilon})$ time algorithm to detect whether an n-vertex graph contains a triangle [40]. We show that triangle-detection in graphs can be linear-time reduced to *disjunctive queries of almost-sure reachability* in MDPs and thus that the latter is hard assuming (A1). (b) For the hardness under (A2) we consider the intermediate problem Orthogonal Vectors, which is known to be hard under (A2) [41], and linear-time reduce it to *disjunctive queries of almost-sure reachability* in MDPs. (c) For *disjunctive safety problems* we give a linear-time reduction from triangle-detection that only requires player 1 vertices and thus hardness also holds in graphs when assuming (A1). (d) However, the reduction we give from Orthogonal Vectors to *disjunctive safety problems* requires random vertices and thus hardness under (A2) only holds on MDPs. (e) Based on the hardness results for almost-sure reachability and safety,

we then exploit reductions between the different types of objectives to obtain the hardness results for Büchi, coBüchi, and Rabin.

Outline. In Section 2 we provide formal definitions, describe the connections between different objectives, and state the conjectures on which the conditional lower bounds are based. Section 3 is about disjunctive reachability queries on MDPs; we first present the improved algorithm and then the conditional lower bounds. In Section 4 we describe the conditional lower bounds for disjunctive safety problems on graphs and MDPs. In Section 5 we provide the improved algorithms for MDPs with Streett objectives. In Section 6 we show how the conditional lower bounds extend from reachability and safety to Büchi, coBüchi, and Rabin and present algorithms for MDPs with Rabin objectives and for MDPs with disjunctive objectives and queries of Büchi and coBüchi objectives. In Section 7 we describe the linear time algorithm for disjunctive coBüchi objectives on graphs for the special case when all target sets are singletons. We conclude in Section 8.

2 Preliminaries

Markov Decision Processes (MDPs) and Graphs. An MDP $P = ((V, E), (V_1, V_R), \delta)$ consists of a finite directed graph with vertices V and edges E with a partition of the vertices into *player 1 vertices* V_1 and *random vertices* V_R and a probabilistic transition function δ. We call an edge (u, v) with $u \in V_1$ *player 1 edge* and an edge (v, w) with $v \in V_R$ a *random edge*. The probabilistic transition function is a function from V_R to $\mathcal{D}(V)$, where $\mathcal{D}(V)$ is the set of probability distributions over V and a random edge $(v, w) \in E$ if and only if $\delta(v)[w] > 0$. For the purpose of this paper we assume for simplicity that, for each random vertex v, $\delta(v)[w]$ is the uniform distribution over all $w \in V$ with $(v, w) \in E$; this is w.l.o.g. as we are only ask whether a probability is zero or one (qualitative analysis) or zero or larger than zero. Graphs are a special case of MDPs with $V_R = \emptyset$.

Sub-MDPs and Maximal End-Components. A sub-MDP of an MDP P induced by a vertex set $X \subseteq V$ is defined as $P[X] = ((X, E \cap (X \times X), (V_1 \cap X, V_R \cap X), \delta')$, where $\delta' : X \to \mathcal{D}(X)$ is for each $v \in V_R \cap X$ the uniform distribution over all $w \in X$ with $(v, w) \in E$. An *end-component* (EC) of an MDP P is a set of vertices $X \subseteq V$ such that (a) the induced sub-MDP $P[X]$ is strongly connected, (b) all outgoing edges in E of vertices in $X \cap V_R$ are contained in $P[X]$, and (c) $P[X]$ contains at least one edge. An end-component is a *maximal end-component* (MEC) if it is maximal under set inclusion. An end-component is *trivial* if it consists of a single vertex (with a self-loop), otherwise it is *non-trivial*. The *MEC-decomposition* of an MDP consists of all MECs of the MDP and the set of vertices that do not belong to any MEC.

Plays and Strategies. A *play* or infinite path in P is an infinite sequence $\omega = \langle v_0, v_1, v_2, \ldots \rangle$ such that $(v_i, v_{i+1}) \in E$ for all $i \in \mathbb{N}$; we denote by Ω the set of all plays. A player 1 *strategy* $\sigma : V^* \cdot V_1 \to V$ is a function that assigns to every finite prefix $\omega \in V^* \cdot V_1$ of a play that ends in a player 1 vertex v a successor vertex $\sigma(\omega) \in V$ such that there exists an edge $(v, \sigma(\omega)) \in E$; we denote by Σ the set of all player 1 strategies. A strategy is *memoryless* if we have $\sigma(\omega) = \sigma(\omega')$ for any $\omega, \omega' \in V^* \cdot V_1$ that end in the same vertex $v \in V_1$.

Objectives and Almost-Sure Winning Sets. An *objective* ψ is a subset of Ω said to be winning for player 1. We say that a play $\omega \in \Omega$ *satisfies the objective* if $\omega \in \psi$. For any measurable set of plays $A \subseteq \Omega$ we denote by $\Pr_v^\sigma(A)$ the probability that a play starting at $v \in V$ belongs to A when player 1 plays strategy σ. A strategy σ is *almost-sure* (*a.s.*) *winning* from a vertex $v \in V$ for an objective ψ if $\Pr_v^\sigma(\psi) = 1$. In graphs the existence of an almost-sure winning strategy corresponds to the existence of a play in the objective. The *almost-sure winning set* $\langle\langle 1 \rangle\rangle_{as}(P, \psi)$ of player 1 is the set of vertices for which player 1 has an almost-sure winning strategy. Computing the almost-sure winning set for some objective is also called *qualitative analysis* of MDPs. Below we define the objectives used in this work. Let $\mathrm{Inf}(\omega)$ for $\omega \in \Omega$ denote the set of vertices that occurs infinitely often in ω.

Reachability For a vertex set $T \subseteq V$ the reachability objective is the set of infinite paths that contain a vertex of T, i.e., $\mathrm{Reach}(T) = \{\langle v_0, v_1, v_2, \ldots\rangle \in \Omega \mid \exists j \geq 0 : v_j \in T\}$.

Safety For a vertex set $T \subseteq V$ the safety objective is the set of infinite paths that *do not contain* any vertex of T, i.e., $\mathrm{Safety}(T) = \{\langle v_0, v_1, v_2, \ldots\rangle \in \Omega \mid \forall j \geq 0 : v_j \notin T\}$.

Büchi For a vertex set $T \subseteq V$ the Büchi objective is the set of infinite paths in which a vertex of T occurs *infinitely often*, i.e., $\mathrm{Büchi}(T) = \{\omega \in \Omega \mid \mathrm{Inf}(\omega) \cap T \neq \emptyset\}$.

coBüchi For a vertex set $T \subseteq V$ the coBüchi objective is the set of infinite paths for which *no* vertex of T occurs *infinitely often*, i.e., $\mathrm{coBüchi}(T) = \{\omega \in \Omega \mid \mathrm{Inf}(\omega) \cap T = \emptyset\}$.

Streett Given a set SP of k pairs (L_i, U_i) of vertex sets $L_i, U_i \subseteq V$ with $1 \leq i \leq k$, the Streett objective is the set of infinite paths for which it holds *for each* $1 \leq i \leq k$ that whenever a vertex of L_i occurs infinitely often, then a vertex of U_i occurs infinitely often, i.e., $\mathrm{Streett}(\mathrm{SP}) = \{\omega \in \Omega \mid L_i \cap \mathrm{Inf}(\omega) = \emptyset \text{ or } U_i \cap \mathrm{Inf}(\omega) \neq \emptyset \text{ for all } 1 \leq i \leq k\}$.

Rabin Given a set RP of k pairs (L_i, U_i) of vertex sets $L_i, U_i \subseteq V$ with $1 \leq i \leq k$, the Rabin objective is the set of infinite paths for which there *exists* an i, $1 \leq i \leq k$, such that a vertex of L_i occurs infinitely often but no vertex of U_i occurs infinitely often, i.e., $\mathrm{Rabin}(\mathrm{RP}) = \{\omega \in \Omega \mid L_i \cap \mathrm{Inf}(\omega) \neq \emptyset \text{ and } U_i \cap \mathrm{Inf}(\omega) = \emptyset \text{ for some } 1 \leq i \leq k\}$.

Given c objectives $\psi_1, \ldots, \psi_c$, the *conjunctive objective* $\psi = \psi_1 \cap \ldots \cap \psi_c$ is given by the intersection of the c objectives, and the *disjunctive objective* $\psi = \psi_1 \cup \ldots \cup \psi_c = \bigvee_{i=1}^{c} \psi_i$ is given by the union of the c objectives. For the *conjunctive query* of c objectives $\psi_1, \ldots, \psi_c$ we define the (almost-sure) winning set to be the set of vertices that have one strategy that is (almost-sure) winning for *each of* the objectives $\psi_1, \ldots, \psi_c$. Analogously, a vertex is in the (almost-sure) winning set $\bigvee_{i=1}^{c} \langle\langle 1 \rangle\rangle_{as}(P, \psi_i)$ for the *disjunctive query* of the c objectives if it is in a (almost-sure) winning set for *at least one* of the c objectives (i.e. we take the union of the winning sets).

Below we present several observations that interlink different types of objectives.

Observation 2.1. *The almost-sure winning set for a* conjunctive objective *is the same as for the corresponding* conjunctive query*.*

Proof. We have for any $v \in V$ and $\sigma \in \Sigma$ and any two objectives ψ_1, ψ_2 that $\Pr_v^\sigma(\psi_1 \wedge \psi_2) = 1$ iff $\Pr_v^\sigma(\psi_1) = 1$ and $\Pr_v^\sigma(\psi_2) = 1$. □

Observation 2.2. *On* graphs (*i.e.* $V_R = \emptyset$) *the winning set for a* disjunctive objective *is the same as for the corresponding* disjunctive query*.*

Proof. For any two objectives ψ_1, ψ_2 we have for each $\omega \in \Omega$ that $\omega \in (\psi_1 \cup \psi_2)$ iff $\omega \in \psi_1$ or $\omega \in \psi_2$. □

Observation 2.3. *The* disjunctive objective *of* Büchi (*resp. reachability*) *objectives is the same as the Büchi* (*resp. reachability*) *objective of the union of the target sets.*

Proof. We show the claim for Büchi, the proof for reachability is analogous. For two target sets $T_1, T_2 \subseteq V$ we have $\{\omega \in \Omega \mid \mathrm{Inf}(\omega) \cap T_1 \neq \emptyset\} \cup \{\omega \in \Omega \mid \mathrm{Inf}(\omega) \cap T_2 \neq \emptyset\} = \{\omega \in \Omega \mid \mathrm{Inf}(\omega) \cap (T_1 \cup T_2) \neq \emptyset\}$. □

Observation 2.4. *The* conjunctive objective *of* coBüchi (*resp. safety*) *objectives is the same as the coBüchi* (*resp. safety*) *objective of the union of the target sets.*

Proof. We show the claim for coBüchi, the proof for safety is analogous. For two target sets $T_1, T_2 \subseteq V$ we have $\{\omega \in \Omega \mid \mathrm{Inf}(\omega) \cap T_1 = \emptyset\} \cap \{\omega \in \Omega \mid \mathrm{Inf}(\omega) \cap T_2 = \emptyset\} = \{\omega \in \Omega \mid \mathrm{Inf}(\omega) \cap (T_1 \cup T_2) = \emptyset\}$. □

By definition each path winning for a safety objective is also winning for the corresponding coBüchi objective while the converse is not always true. However, when it comes to the non-emptiness of winning sets these two objectives become equivalent.

Observation 2.5. *For a fixed MDP P the winning set for Safety(T) is non-empty iff the winning set for coBüchi(T) is non-empty. This equivalence extends also to conjunctions and disjunctions of safety and coBüchi objectives.*

Proof. By [21, p. 891] (see also Section 5.1) the winning set for Safety (T) resp. coBüchi (T) is non-empty if and only if there exists an end-component X with $X \cap T = \emptyset$. □

Observation 2.6. *Disjunctive* (*Obj./Qu.*) *Reachability in MDPs can be linear time reduced to disjunctive* (*Obj./Qu.*) *Büchi-Objectives in MDPs and vice versa.*

Proof. Reachability ⇒ Büchi: For each target set T replace each $t \in T$ with two vertices: $t_{\mathrm{in}} \in V_1$ and t_{out}, where t_{out} belongs to the same player as t. Assign all incoming edges of t to t_{in} and all outgoing edges of t to t_{out}, and add the edge $(t_{\mathrm{in}}, t_{\mathrm{out}})$ and the self-loop $(t_{\mathrm{in}}, t_{\mathrm{in}})$. Let the corresponding target set for Büchi be the union of t_{in} for all $t \in T$. A vertex t_{in} in the modified MDP can be visited infinitely often almost surely iff in the original MDP the vertex t can be reached almost surely.

Büchi ⇒ Reachability: For each target set T replace each $t \in T$ with three vertices: $t_{\mathrm{in}} \in V_R$, $t_r \in V_1$, and t_{out}, where t_{out} belongs to the same player as t. Assign all incoming edges of t to t_{in} and all outgoing edges of t to t_{out}, and add the edges $(t_{\mathrm{in}}, t_{\mathrm{out}})$, (t_{in}, t_r), and (t_r, t_{out}). Let the corresponding target set for Reachability be the union of t_r for all $t \in T$. A vertex t_r

in the modified MDP can be reached almost surely iff in the original MDP the vertex t can almost surely be visited infinitely often. □

Observation 2.7. *Conjunctive Büchi (resp. coBüchi) objectives are special instances of Streett objectives.*

Proof. For Büchi let $L_i = V$ and $U_i = T_i$, for coBüchi let $L_i = T_i$ and $U_i = \emptyset$. □

Observation 2.8. *Disjunctive Büchi (resp. coBüchi) objectives are special instances of Rabin objectives.*

Proof. For Büchi let $L_i = T_i$ and $U_i = \emptyset$, for coBüchi let $L_i = V$ and $U_i = T_i$. □

2.1 Conjectured Lower Bounds

While classical complexity results are based on standard complexity-theoretical assumptions, e.g., $\mathsf{P} \neq \mathsf{NP}$, polynomial lower bounds are often based on widely believed, conjectured lower bounds about well studied algorithmic problems. Our lower bounds will be conditioned on the popular conjectures discussed below.

First, we consider conjectures on Boolean matrix multiplication [40, 3] and triangle detection [3] in graphs, which build the basis for our lower bounds on dense graphs. A triangle in a graph is a triple x, y, z of vertices such that $(x, y), (y, z), (z, x) \in E$.

Conjecture 2.9 (Combinatorial Boolean Matrix Multiplication Conjecture (BMM))**.** *There is no $O(n^{3-\varepsilon})$ time combinatorial algorithm for computing the boolean product of two $n \times n$ matrices for any $\varepsilon > 0$.*

Conjecture 2.10 (Strong Triangle Conjecture (STC))**.** *There is no $O(\min\{n^{\omega-\varepsilon}, m^{2\omega/(\omega+1)-\varepsilon}\})$ expected time algorithm and no $O(n^{3-\varepsilon})$ time combinatorial algorithm that can detect whether a graph contains a triangle for any $\varepsilon > 0$, where $\omega < 2.373$ is the matrix multiplication exponent.*

By a result of Vassilevska Williams and Ryan Williams [40], we have that BMM is equivalent to the combinatorial part of STC. Moreover, if we do not restrict ourselves to combinatorial algorithms, STC still gives a super-linear lower bound.

Second, we consider the Strong Exponential Time Hypothesis [31, 12] and the Orthogonal Vectors Conjecture [6], the former dealing with satisfiability in propositional logic and the latter with the *Orthogonal Vectors Problem*.

The Orthogonal Vectors Problem (OV). Given two sets S_1, S_2 of d-bit vectors with $|S_i| \leq N$, $d \in \Theta(\log N)$, are there $u \in S_1$ and $v \in S_2$ such that $\sum_{i=1}^{d} u_i \cdot v_i = 0$?

Conjecture 2.11 (Strong Exponential Time Hypothesis (SETH))**.** *For each $\varepsilon > 0$ there is a k such that k-CNF-SAT on n variables and m clauses cannot be solved in $O(2^{(1-\varepsilon)n}\operatorname{poly}(m))$ time.*

Conjecture 2.12 (Orthogonal Vectors Conjecture (OVC))**.** *There is no $O(N^{2-\varepsilon})$ time algorithm for the Orthogonal Vectors Problem for any $\varepsilon > 0$.*

By a result of Williams [41] we know that SETH implies OVC, i.e., whenever a problem is hard assuming OVC, it is also hard when assuming SETH. Hence, it is preferable to use OVC for proving lower bounds. Finally, to the best of our knowledge, no relations between the former two conjectures and the latter two conjectures are known.

Remark 2.13. *The conjectures that no* polynomial *improvements over the best known running times are possible do not exclude improvements by sub-polynomial factors such as polylogarithmic factors or factors of, e.g.,* $2^{\sqrt{\log n}}$ *as in [42].*

3 Reachability in MDPs

First let us briefly discuss reachability on Graphs. The winning set for disjunctive reachability can simply be computed by union all target sets and then starting a breadth-first search which is in $O(m)$. On the other hand, the problem becomes NP-complete when considering conjunctive reachability [23], as with conjunction one can require a path to contain several vertices and in particular one can embed the well-known NP-hard problem of Hamiltonian path.

Turning to MDPs, notice that in MDPs based on acyclic graphs almost-sure reachability is equivalent to computing the winning set for a player with reachability objectives in a 2-player graph-game where all the random vertices are owned by the opponent (as random will play the optimal strategy for the opponent with non-zero probability). As computing the winning set for conjunctive reachability in the 2-player graph-game is PSPACE-hard [23] even for acyclic graphs, we have that conjunctive almost-sure reachability in MDPs is PSPACE-hard as well. Moreover, as we will show later, compared to graphs, also disjunctive reachability becomes harder, i.e., we will provide polynomial lower-bound based on popular conjectures.

In the first part of this section we present an improved algorithm for disjunctive reachability queries in MDPs. As disjunctive reachability objectives can be easily reduced to a single reachability objective by taking the union of all target sets, the algorithm mentioned above is also an algorithm for disjunctive reachability objectives (by setting $k = 1$). In the second part we present two lower bounds for disjunctive reachability queries, an $\Omega(n^{3-o(1)})$ lower bound based on STC and an $\Omega(m^{2-o(1)})$ lower bound based on OVC (resp. SETH).

3.1 Algorithm for Disjunctive Reachability Queries in MDPs

In this section we present an algorithm to compute the almost-sure winning set for disjunctive reachability queries in MDPs. In particular we show the following theorem:

Theorem 3.1. *For an MDP P and target sets $T_i \subseteq V$ for $1 \leq i \leq k$ the almost-sure winning set for disjunctive reachability queries can be computed in $O(km + \mathrm{MEC})$ time, where MEC is the time needed to compute a MEC-decomposition.*

A vertex v is in the almost-sure winning set if player 1 has a strategy to reach one of the k target sets T_i with probability 1 starting from v. Note that the sets T_i are not absorbing in contrast to what is often assumed for the reachability objective in MDPs. The trivial algorithm would be to invoke an algorithm for almost-sure reachability in MDPs k times (for one target

set T_i at a time, temporarily making the set T_i absorbing if necessary). The crucial observation to improve upon this is that given an MDP without non-trivial end-components, almost-sure reachability in MDPs can be solved in linear time.

We further observe that, for each target set, either all vertices of an end-component are winning (almost-surely) or none. Thus if we know the MEC-decomposition of an MDP, we can contract the MECs to single vertices with self-loops and solve almost-sure reachability on the derived MDP. This derived MDP does not have non-trivial end-components, therefore given the MEC decomposition, the problem can be solved in linear time per target set. Our algorithm implies that almost-sure reachability (i.e. $k = 1$) can be solved in the same asymptotic time needed to determine the MEC-decomposition of an MDP.

Definition 3.2 (Contraction of MECs)**.** *Contracting a MEC X in an MDP P creates a modified MDP P' from P where the vertices of X are replaced by a single vertex u that belongs to player 1 and the edges to or from a vertex in X are replaced with edges to or from, respectively, the vertex u; parallel edges are omitted from P', for parallel random edges the probabilities are added up.*

Observation 3.3 ([15])**.** *The MDP P' that is constructed from the MDP P by contracting all MECs of P does not contain any non-trivial end-components.*

Proof. Assume by contradiction that the MDP P' contains an end-component X' with at least two vertices. Let X be the set of vertices corresponding to the vertices of X' in the original MDP P. Then X is an end-component in P, a contradiction to the definition of P'. □

In the derived MDP we basically apply, for each target set, one iteration of the classical almost-sure reachability algorithm but with a slightly modified random attractor computation defined below. The classical algorithm repeatedly executes the following two steps: 1) Compute the vertices S from which player 1 can reach the target set T. 2a) If $S = V$, output S as the (almost-sure) winning set of player 1. 2b) If $S \subsetneq V$, remove the random attractor of $V \setminus S$ from the graph (and from V) and repeat. Intuitively, a *random attractor* of a set of vertices W contains the vertices from which there is a positive probability to reach W for every strategy of player 1. The *extended random attractor*, formally defined below and used implicitly in [16], additionally includes player 1 vertices for which the only player 1 strategy to avoid a positive probability to reach W is using a self-loop of a vertex not in the target set. Additionally, we explicitly avoid adding vertices in the considered target set to the attractor. In the classical algorithm this was achieved by making the target set absorbing, which would not work for the extended random attractor.

Definition 3.4 (Extended Random Attractor)**.** *Let $E(v)$ denotes the set of vertices $u \in V$ for which $(v, u) \in E$. In an MDP $P = ((V, E), (V_1, V_R), \delta)$ the* extended random attractor *$Attr^+(P, W, T)$ for sets of vertices $W, T \subseteq V$ is defined as $Attr^+(P, W, T) = \bigcup_{j \geq 0} Z_j$ where $Z_0 = W \setminus T$ and Z_j for $j > 0$ is defined recursively as $Z_{j+1} = Z_j \cup \{v \in V_R \mid E(v) \cap Z_j \neq \emptyset\} \cup \{v \in V_1 \mid E(v) \subseteq Z_j \cup \{v\}\} \setminus T$. In contrast to a random attractor (a) a set of vertices T can be specified that is never included in $Attr^+(P, W, T)$ and (b) a player 1 vertex is also included in*

Z_{j+1} if all its outgoing edges apart from its self-loop are contained in Z_j. The extended random attractor $A = Attr^+(P, W, T)$ can be computed in $O(\sum_{v \in A} Indeg(v) + |V_1 \setminus T|)$ time [9, 30].

Putting the pieces together, our algorithm looks as follows: First, the MEC-decomposition of the input MDP P is computed. Then all MECs of P are contracted to construct the derived MDP P', which does not contain any non-trivial MECs. For each target set we execute one iteration of the classical algorithm, replacing the usual random attractor with the extended random attractor. The union of the winning sets determined for each target set then gives the winning set of player 1 for disjunctive reachability.

Algorithm DisjReachMDP: Disjunctive Query Reachability in MDPs

Input : an MDP $P = ((V, E), (V_1, V_R), \delta)$ and target sets $T_i \subseteq V$ for $1 \leq i \leq k$
Output: $\bigvee_{1 \leq i \leq k} \langle\!\langle 1 \rangle\!\rangle_{as} (P, \text{Reach}(T_i))$

```
compute MEC decomposition of P
let P' be P with all MECs contracted
let T'_i for 1 ≤ i ≤ k be the set of vertices of P' that represent some vertex of T_i
W' ← ∅
for i ← 1 to k do
    S' ← GraphReach(P', T'_i)
    A' ← Attr+(P', V' \ S', T'_i)
    W' ← W' ∪ V' \ A'
let W be the vertices in W' after undoing contraction
return W
```

Proposition 3.5 (Runtime)**.** *Algorithm* DisjReachMDP *runs in time $O(km + \text{MEC})$.*

Proof. Contracting all MECs can be done in time $O(m)$ as we have to consider each edge (and vertex) at most twice. The for-loop is executed k times. Within the for-loop both the vertices S that can reach T_i and the extended random attractor $A = Attr^+(P', V \setminus S, T_i)$ can be found in linear time, that is, in $O(km)$ time over all iterations of the for-loop. Undoing the contraction takes again at most $O(m)$ time. □

Proposition 3.6 (Correctness)**.** *For an MDP P and target sets $T_i \subseteq V$ for $1 \leq i \leq k$ Algorithm* DisjReachMDP *returns the set $\bigvee_{1 \leq i \leq k} \langle\!\langle 1 \rangle\!\rangle_{as} (P, Reach(T_i))$.*

Proof. We assume that in the MDP P each vertex has at least one outgoing edge and each random vertex has at least one outgoing edge that is not a self-loop. This is w.l.o.g. because $\bigvee_{1 \leq i \leq k} \langle\!\langle 1 \rangle\!\rangle_{as} (P, \text{Reach}(T_i))$ does not change if we replace each vertex without outgoing edges by a vertex with a self-loop and treat a random vertex whose only outgoing edge is a self-loop as a player 1 vertex.

First note that by definition a vertex is in $\bigvee_{1 \leq i \leq k} \langle\!\langle 1 \rangle\!\rangle_{as} (P, \text{Reach}(T_i))$ if and only if it is in $\langle\!\langle 1 \rangle\!\rangle_{as} (P, \text{Reach}(T_i))$ for some $1 \leq i \leq k$. Hence we can consider the k target sets separately

by showing that in the i-th iteration of the for-loop of Algorithm DisjReachMDP the set $\langle\!\langle 1 \rangle\!\rangle_{as}(P, \text{Reach}(T_i))$ is identified.

Let P' be the MDP derived from the MDP P by contracting all MECs of P and let T'_i be the set of contracted vertices that represent some vertex of T_i as in Algorithm DisjReachMDP. We use the superscript $'$ to denote sets related to the MDP P' and omit the superscript for sets related to the original MDP P. Note that since only strongly connected subgraphs are contracted in P', it clearly holds that a vertex $v \in V$ can reach another vertex $u \in V$ if and only if the vertex $v' \in V'$ corresponding to v can reach the vertex $u' \in V'$ corresponding to u.

Fix some iteration i and let $S' = \texttt{GraphReach}(P', T'_i)$, let $A' = Attr^+(P', V' \setminus S', T'_i)$, and let $W'_i = V' \setminus A'$, that is, W'_i is the set added to W' in the i-th iteration of the for-loop of Algorithm DisjReachMDP. Let the same letters without superscript denote the corresponding sets of vertices after reverting the contraction of the MECs of P. We prove the lemma by first showing $\langle\!\langle 1 \rangle\!\rangle_{as}(P, \text{Reach}(T_i)) \subseteq W_i$ and then $W_i \subseteq \langle\!\langle 1 \rangle\!\rangle_{as}(P, \text{Reach}(T_i))$.

We prove $\langle\!\langle 1 \rangle\!\rangle_{as}(P, \text{Reach}(T_i)) \subseteq W_i$ by showing $A \subseteq V \setminus \langle\!\langle 1 \rangle\!\rangle_{as}(P, \text{Reach}(T_i))$ by induction on the recursive definition of $A' = Attr^+(P', V' \setminus S', T'_i) = \cup_{j \geq 0} Z'_j$, where the sets Z'_j are defined as in Definition 3.4 and the sets Z_j are the corresponding sets after reverting the contraction of the MECs of P. Since the attractor computation is done on P', each set Z_j either contains all vertices of a MEC of P or none. Clearly $A \cap T_i = \emptyset$ as vertices in T'_i are explicitly excluded from A'. Player 1 cannot reach T_i almost surely from the vertices in $Z_0 = V \setminus S$ because these vertices cannot reach any vertex in T_i. Assume the claim holds for Z_j, i.e., for all vertices $z \in Z_j$ and any strategy σ of player 1 we have $\Pr^\sigma_z(P, \text{Reach}(T_i)) < 1$. By the definition of Z'_{j+1}, for a random vertex v' in $Z'_{j+1} \setminus Z'_j$ there is a positive probability to reach a vertex in Z'_j; thus, $\Pr^{\sigma'}_{v'}(P', \text{Reach}(T'_i)) < 1$ for any strategy σ' of player 1. Random vertices in P' were not contracted, thus the same argument holds for Z_{j+1} and P. A player 1 vertex x' in $Z'_{j+1} \setminus Z'_j$ corresponds to either a player 1 vertex x or a MEC X in $Z_{j+1} \setminus Z_j$. In both cases all the edges from x resp. X lead to vertices in Z_j or to x resp. X itself. Hence since $x \notin T_i$ resp. $X \cap T_i = \emptyset$, we also have $\Pr^\sigma_x(\text{Reach}(T_i)) < 1$ for any strategy σ of player 1 and x resp. all $x \in X$.

We next show $W_i \subseteq \langle\!\langle 1 \rangle\!\rangle_{as}(P, \text{Reach}(T_i))$. Let $G[W_i] = (W_i, E \cap (W_i \times W_i))$ be the subgraph induced by the vertices in W_i. We establish two properties: (1) all outgoing edges of random vertices $V_R \cap W_i$ lead to vertices in W_i, and (2) all vertices in $W_i \setminus T_i$ can reach T_i in $G[W_i]$. The claim follows from these two properties using the same proof as for the classical algorithm for almost-sure reachability in MDPs (see below).

(1) For vertices in V_R we distinguish whether they are contained in a MEC of P or not. In the first case property (1) follows from the fact that a MEC has no outgoing random edges and every MEC is either completely contained in W_i or completely contained in $V \setminus W_i$. In the second case property (1) follows from the definition of an extended random extractor because a vertex in $V_R \cap W'_i$ with an edge to a vertex in A would have been included in A.

(2) To show property (2) we will use that by Observation 3.3 the MDP P' does not contain any non-trivial MEC. Assume by contradiction that some vertices in $W_i \setminus T_i$ cannot reach T_i in $G[W_i]$. Then there exists a bottom SCC C (i.e. an SCC without outgoing edges,

possibly a single vertex) in $G[W_i]$ with $C \cap T_i = \emptyset$. Note that every MEC in $G[W_i]$ is completely contained in one of the SCCs of $G[W_i]$. By property (1) C has no outgoing random edges in P; by this and the fact that C is strongly connected, the corresponding set C' of vertices in P' would be a non-trivial MEC in P' if it contained more than one vertex. Thus C' can contain only one vertex c' and this vertex has either no outgoing edge or only a self-loop in $G'_{W'_i}$. If c' was a player 1 vertex, then all its outgoing edges would go to vertices in A' or be a self-loop, hence c' would have been included in the attractor A'. If c' was a random vertex, then by the assumption that in P, and thus in P', every random vertex has an outgoing edge that is not a self-loop we would get a contradiction to property (1). Thus no such bottom SCC C can exist, that is, every bottom SCC of $G[W_i]$ contains a vertex of T_i and thus property (2) holds.

To see that the two established properties imply $W_i \subseteq \langle\!\langle 1 \rangle\!\rangle_{as}(P, \mathrm{Reach}(T_i))$, let for a vertex $u \in W_i$ be $d(u)$ the shortest path distance to a vertex in T_i. Consider the following strategy σ of player 1: For a player 1 vertex u, choose an edge to a vertex v such that $d(v) < d(u)$. For a random vertex u, there is always an edge to a vertex v such that $d(v) < d(u)$. Let $\ell = |W_i|$ and let α be the minimum positive transition probability in the MDP P. For all vertices $v \in W_i$ the probability that T_i is reached within ℓ steps is at least α^ℓ, that is, the probability that T_i is not reached within $b \cdot \ell$ steps is at most $(1 - \alpha^\ell)^b$, which goes to 0 as b goes to ∞. Thus for all $v \in W_i$ strategy σ ensures that T_i is reached with probability 1. □

3.2 Conditional Lower Bounds for Disjunctive Reachability in MDPs

Here we complement the above algorithm by conditional lower bounds for disjunctive reachability queries in MDPs. These lower bound will be based on the conjectures STC, SETH, and OVC introduced in Section 2.1.

We first present our lower bound for dense MDPs based on STC.

Theorem 3.7. *There is no combinatorial $O(n^{3-\epsilon})$ or $O((k \cdot n^2)^{1-\epsilon})$ algorithm (for any $\epsilon > 0$) for disjunctive reachability queries in MDPs under Conjecture 2.10 (i.e., unless STC and BMM fail). In particular, there is no such algorithm deciding whether the winning set is non-empty or deciding whether a specific vertex is in the winning set. The bounds hold for dense MDPs with $m = \Theta(n^2)$.*

The above theorem is by the following reduction from the triangle detection problem.

Reduction 3.8. *Given an instance of triangle detection, i.e., a graph $G = (V, E)$, we build the following MDP P.*

- *The vertices V' of P are given by four copies V^1, V^2, V^3, V^4 of V, a start vertex s, and absorbing vertices $F = \{g_v \mid v \in V\}$. The edges E' of P are defined as follows: There is an edge from s to the first copy $v^1 \in V^1$ of every $v \in V$ and the last copy $v^4 \in V^4$ of every $v \in V$ is connected to its first copy v^1 and its corresponding absorbing vertex $g_v \in F$; further for $1 \leq i \leq 3$ there is an edge from v^i to u^{i+1} iff $(v, u) \in E$.*

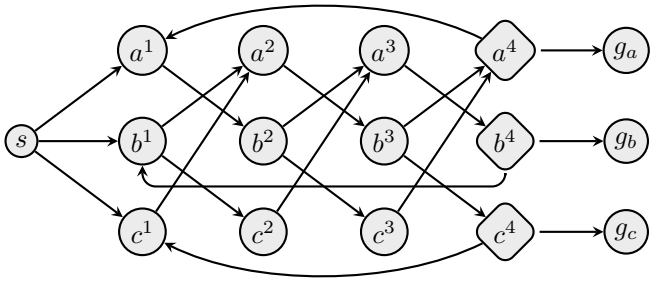

Figure 1: Illustration of Reduction 3.8, with $G = (\{a, b, c\}, \{(a, b), (b, a), (b, c), (c, a)\})$. Vertices drawn as cycle are owned by player 1, vertices drawn as diamond are random vertices.

- *The set of vertices V' is partitioned into player 1 vertices $V'_1 = \{s\} \cup V^1 \cup V^2 \cup V^3 \cup F$ and random vertices $V'_R = V^4$. Moreover, the probabilistic transition function for each vertex $v \in V'_R$ chooses among v's successors with equal probability $1/2$ each.*

The reduction is illustrated in Figure 1. As all random choices are uniformly at random we omit the exact probabilities in the figures.

Next we prove that Reduction 3.8 is indeed a valid reduction from triangle detection to disjunctive reachability queries in MDPs.

Lemma 3.9. *A graph G has a triangle iff s is contained in $\bigvee_{v \in V} \langle\langle 1 \rangle\rangle_{as} (P, \mathit{Reach}(T_v))$, where P is the MDP given by Reduction 3.8 and $T_v = \{g_v\}$ for $v \in V$.*

Proof. For the only if part assume that G has a triangle with vertices a, b, c and let a^i,b^i,c^i be the copies of a, b, c in V^i. Now a strategy for player 1 in the MDP P to reach g_a with probability 1 is as follows: When in s, go to a^1; when in a^1, go to b^2; when in b^2, go to c^3; when in c^3, go to a^4. As a, b, c form a triangle, all the edges required by the above strategy exist. When player 1 starts in s and follows the above strategy the only random vertex he encounters is a^4. The random choice sends him to the target vertex g_a and to vertex a^1 with probability $1/2$ each. In the former case he is done, in the latter case he continues playing his strategy and will reach a^4 again after three steps. The probability that player 1 has reached g_a after $3q + 1$ steps is $1 - (1/2)^q$ which converges to 1 with q going to infinity. Thus we have found a strategy to reach g_a with probability 1.

For the if part assume that $s \in \bigvee_{v \in V} \langle\langle 1 \rangle\rangle_{as} (P, \mathrm{Reach}(T_v))$. That is, there is an $a \in V$ such that $s \in \langle\langle 1 \rangle\rangle_{as} (P, \mathrm{Reach}(T_a))$. Let us consider a corresponding strategy for reaching $T_a = \{g_a\}$. First, assume that the strategy would visit a vertex v^4 for $v \in V \setminus \{a\}$. Then with probability $1/2$ player 1 would end up in the vertex g_v which has no path to g_a, a contradiction to $s \in \langle\langle 1 \rangle\rangle_{as} (P, \mathrm{Reach}(T_a))$. Thus the strategy has to avoid visiting vertices v^4 for $v \in V \setminus \{a\}$. Second, as the only way to reach g_a is a^4, the strategy has to choose a^4. But then with probability $1/2$ it will be send to a^1 and there must be a path from a^1 to g_a that doesn't not cross $V^4 \setminus \{a^4\}$. By the latter this path must be of the form a^1, b^2, c^3, a^4, g_a for some $b, c \in V$. Now by the construction of G' in the MDP P the vertices a, b, c form a triangle in the original graph G. □

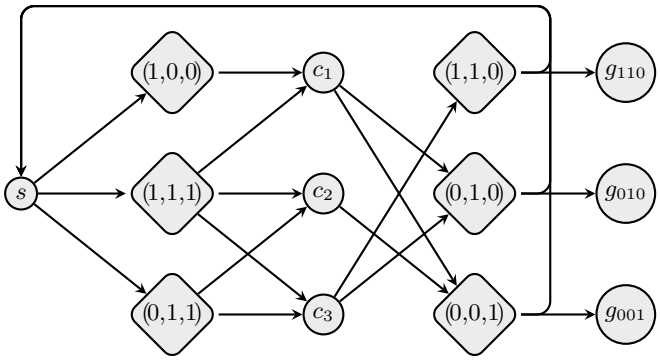

Figure 2: Illustration of Reduction 3.11 for $S_1 = \{(1,0,0),(1,1,1),(0,1,1)\}$ and $S_2 = \{(1,1,0),(0,1,0),(0,0,1)\}$.

The size and the construction time of the MDP P, constructed by Reduction 3.8, is linear in the size of the original graph G and we have $k = \Theta(n)$ target sets. Thus if we would have a combinatorial $O(n^{3-\epsilon})$ or $O((k \cdot n^2)^{1-\epsilon})$ algorithm for disjunctive queries of reachability objectives in MDPs for any $\epsilon > 0$, we would immediately get a combinatorial $O(n^{3-\epsilon})$ algorithm for triangle detection, which contradicts STC and BMM.

Next we present a lower bound for sparse MDPs based on OVC and SETH.

Theorem 3.10. *There is no $O(m^{2-\epsilon})$ or $O((k\cdot m)^{1-\epsilon})$ algorithm (for any $\epsilon > 0$) for disjunctive reachability queries in MDPs under Conjecture 2.12 (i.e., unless OVC and SETH fail). In particular, there is no such algorithm deciding whether the winning set is non-empty or deciding whether a specific vertex is in the winning set.*

To prove the above we give a reduction from OVC to disjunctive reachability queries in MDPs.

Reduction 3.11. *Given two sets S_1, S_2 of d-dimensional vectors, we build the following MDP P.*

- *The vertices V of the MDP P are given by a start vertex s, vertices S_1 and S_2 representing the sets of vectors, vertices $\mathcal{C} = \{c_i \mid 1 \le i \le d\}$ representing the coordinates, and absorbing vertices $F = \{g_v \mid v \in S_2\}$. The edges E of P are defined as follows: the start vertex s has an edge to every vertex of S_1 and every vertex $v \in S_2$ has an edge to s and to its corresponding absorbing vertex $g_v \in F$; further for each $x \in S_1$ there is an edge* to $c_i \in \mathcal{C}$ *iff $x_i = 1$ and for each $y \in S_2$ there is an edge* from $c_i \in \mathcal{C}$ *iff $y_i = 0$.*
- *The set of vertices V is partitioned into player 1 vertices $V_1 = \{s\} \cup \mathcal{C} \cup F$ and random vertices $V_R = S_1 \cup S_2$. The probabilistic transition function for each vertex $v \in V_R$ chooses among v's successors uniformly at random.*

The reduction is illustrated on an example in Figure 2.

Lemma 3.12. *There exist orthogonal vectors $x \in S_1$, $y \in S_2$ iff $s \in \bigvee_{v \in V} \langle\!\langle 1 \rangle\!\rangle_{as}(P, \mathit{Reach}(T_v))$ where P is the MDP given by Reduction 3.11 and $T_v = \{g_v\}$ for $v \in V$.*

Proof. If one of the sets S_1 and S_2 contains the all-zero vector, an orthogonal pair of vectors exists trivially and this can be detected in linear time; thus we assume w.l.o.g. that none of the sets contains the all-zero vector.

For the only if part assume that there are orthogonal vectors $x \in S_1$, $y \in S_2$. Now a strategy for player 1 in the MDP P to reach g_y with probability 1 is as follows: When in s, go to x; when in some $c \in \mathcal{C}$, go to y. As x and y are orthogonal, each $c_i \in \mathcal{C}$ reachable from x has an edge to y, i.e., for $x_i = 1$ it must be that $y_i = 0$. When player 1 starts in s and follows the above strategy, he reaches y after three steps. There the random choice sends him to the target vertex g_y and back to vertex y with probability 1/2 each. In the former case he is done, in the latter case he continues playing his strategy and will reach y again after three steps. The probability that player 1 has reached g_y after $3q$ steps is $1 - (1/2)^q$, which converges to 1 with q going to infinity. Thus we have found a strategy to reach g_y with probability 1.

For the if part assume that $s \in \bigvee_{v \in V} \langle\!\langle 1 \rangle\!\rangle_{as}(P, \mathrm{Reach}(T_v))$. That is, there is an $y \in S_2$ such that $s \in \langle\!\langle 1 \rangle\!\rangle_{as}(P, \mathrm{Reach}(T_y))$. Let us consider a corresponding strategy for reaching $T_y = \{g_y\}$. First, assume that the strategy would visit a vertex $y' \in S_2$ for $y' \neq y$. Then with probability 1/2 the player would end up in the vertex $g_{y'}$ which has no path to g_y, a contradiction to $s \in \langle\!\langle 1 \rangle\!\rangle_{as}(P, \mathrm{Reach}(T_y))$. Thus the strategy has to avoid visiting vertices $S_2 \setminus \{y\}$. Second, as the only way to reach g_y is y, the strategy has to choose y. But then with probability 1/2 it will be send to s and thus there must be a strategy to reach g_y from s with probability 1 that does not cross $S_2 \setminus \{y\}$. As y is the only predecessor of g_y, there must also be such a strategy to reach y. In other words, there must be an $x \in S_1$ such that for each successor $c_i \in \mathcal{C}$ there is an edge to y. By the construction of the MDP P this is equivalent to the existence of an $x \in S_1$ such that whenever $x_i = 1$ then $y_i = 0$, and thus x and y are orthogonal vectors. □

The number of vertices in P, constructed by Reduction 3.11, is $O(N)$ and the construction can be performed in $O(N \log N)$ time (recall that $d \in O(\log N)$). The number of edges m is $O(N \log N)$ (thus we consider P to be a sparse MDP) and the number of target sets $k \in \Theta(N) = \theta(m/\log N)$. Finally, if we would have an $O(m^{2-\epsilon})$ or $O((k \cdot m)^{1-\epsilon})$ algorithm for disjunctive reachability queries in MDPs for any $\epsilon > 0$, we would immediately get an $O(N^{2-\epsilon})$ algorithm for OV, which contradicts OVC (and thus SETH).

4 Safety Objectives

It is well-known that computing the a.s. winning set for a single safety objective in an MDP is equivalent to computing the winning set of player 1 for safety objectives in the 2-player graph-game where all the random vertices are owned by the opponent, called player 2 (see e.g. [14]). A 2-player graph-game is defined as a graph with a partition of the vertices into player 1 vertices V_1 and player 2 vertices V_2. A player 2 strategy is defined analogous to a player 1 strategy (replacing the vertices V_1 with the vertices V_2 in the definition). The objective of player 2 is the dual of the objective of player 1.

Safety objectives in 2-player graph-games can be computed in $O(m)$ time by computing a player 2 attractor (the definition of a player 1 or player 2 attractor is analogous to the definition of a random attractor in Definition 5.12). Thus in MDPs the a.s. winning set for a single safety objective can be computed in $O(m)$ time by computing a random attractor, and the a.s. winning set for a disjunctive query can be determined in $O(k \cdot m)$ time by computing k random attractors and union the winning sets. Conjunctive safety can be reduced to a single safety objective in $O(b)$ time by taking the union of all the sets T_i.

Turning to disjunctive safety objectives, we have the same equivalence to 2-player graph-games as for single objectives (Observation 4.1). In this 2-player game the disjunctive safety objective is the complementary objective to the conjunctive reachability objective with the same sets and, as the game is determined [23][4], the PSPACE-hardness shown in [23] also applies to disjunctive safety objectives.

Observation 4.1. *Computing the a.s. winning set for a disjunctive safety objective in an MDP with player 1 vertices V_1 and random vertices V_R is equivalent to computing the same disjunctive safety objective in the 2-player graph-game with the same edges and the same player 1 vertices and player 2 vertices $V_2 = V_R$.*

Proof. We show that a vertex s is almost sure winning in the MDP if and only if it is winning for player 1 in the game graph.
$\Leftarrow$: Assume s is not winning for player 1 in the graph-game. Then s is winning for player 2 and thus player 2 has a strategy to visit all target sets from s. As there are only finitely many target sets, all these target sets are visited after a finite number of steps, lets say after l steps. Now consider the corresponding MDP; with some constant probability the random choices in the MDP will follow exactly the strategy of player 2 in the graph-game for the first l steps and in that case player 1 cannot win almost surely from s. Hence, s is not in the a.s. winning set.
$\Rightarrow$: Assume player 1 has a winning strategy for the graph-game starting in s. By definition this strategy is also winning for the MDP (if it is winning for each possible choice of player 2 then it also winning for a random choice). □

4.1 Conditional Lower Bounds for Safety Objectives

We first present a lower bound for disjunctive safety based on STC that even holds on graphs.

Theorem 4.2. *There is no combinatorial $O(n^{3-\epsilon})$ or $O((k \cdot n^2)^{1-\epsilon})$ algorithm (for any $\epsilon > 0$) for disjunctive safety (objectives or queries) in graphs under Conjecture 2.10 (i.e., unless STC and BMM fail). In particular, there is no such algorithm deciding whether the winning set is non-empty or deciding whether a specific vertex is in the winning set.*

The above is by the linear time reduction from triangle detection to disjunctive safety in graphs provided below.

Reduction 4.3. *Given a graph $G = (V, E)$ (for triangle detection), we build a graph $G' = (V', E')$ (for disjunctive safety) as follows. As vertices V' we have four copies V^1, V^2, V^3, V^4 of*

[4] A graph-game is determined if the winning set of player 1 is the complement of the winning set of player 2.

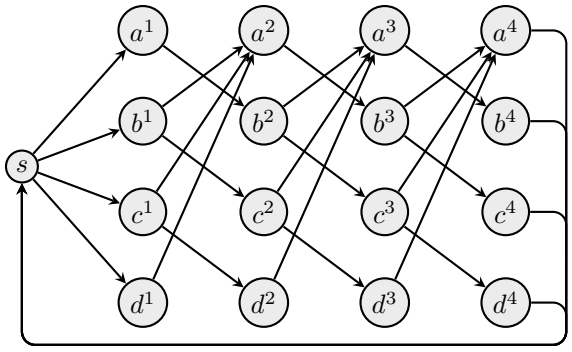

Figure 3: Illustration of Reduction 4.3, with $G = (\{a, b, c, d\}, \{(a, b), (b, a), (b, c), (c, a), (c, d), (d, a)\})$. The target sets for disjunctive safety are $T_a = \{b^1, c^1, d^1, b^4, c^4, d^4\}$, $T_b = \{a^1, c^1, d^1, a^4, c^4, d^4\}$, $T_c = \{a^1, b^1, d^1, a^4, b^4, d^4\}$, and $T_d = \{a^1, b^1, c^1, a^4, b^4, c^4\}$.

V and a vertex s. A vertex $v^i \in V^i$ has an edge to a vertex $u^{i+1} \in V^{i+1}$ iff $(v, u) \in E$. Finally, s has an edge to all vertices in V^1 and all vertices in V^4 have an edge to s.

Reduction 4.3 is illustrated in Figure 3.

Lemma 4.4. *Let G' be the graph given by Reduction 4.3 for a graph G and let $T_v = (V^1 \setminus \{v^1\}) \cup (V^4 \setminus \{v^4\})$. Then the following statements are equivalent.*

1. *G has a triangle.*
2. *s is in the winning set of $(G', \bigvee_{v \in V} \mathit{Safety}(T_v))$.*
3. *The winning set of $(G', \bigvee_{v \in V} \mathit{Safety}(T_v))$ is non-empty.*

Proof. (1)⇒(2): Assume that G has a triangle with vertices a, b, c and let a^i,b^i,c^i be the copies of a, b, c in V^i. Now a strategy for player 1 in G' to satisfy Safety (T_a) is as follows: When in s, go to a^1; when in a^1, go to b^2; when in b^2, go to c^3; when in c^3, go to a^4; and when in a^4, go to s. As a, b, c form a triangle, all the edges required by the above strategy exist. When player 1 starts in s and follows the above strategy, then he plays an infinite path that only uses vertices s, a^1, b^2, c^3, a^4 and thus satisfies Safety (T_a).

(2)⇒(1): Assume that there is a winning play starting in s and satisfying Safety (T_a). Starting from s, this play has to first go to a^1, as all other successors of s would violate the safety constraint. Then the play continues on some vertex $b^2 \in V^2$ and $c^3 \in V^3$ and then, again by the safety constraint, has to enter a^4. Now by construction of G' we know that there must be edges $(a, b), (b, c), (c, a)$ in the original graph G, i.e. there is a triangle in G.

(2)⇔(3): Notice that when removing s from G' we get an acyclic graph and thus each infinite path has to contain s infinitely often. Thus, if the winning set is non-empty, there is a cycle winning for some vertex and then this cycle is also winning for s. For the converse direction we have that if s is in the winning set, then the winning set is non-empty. □

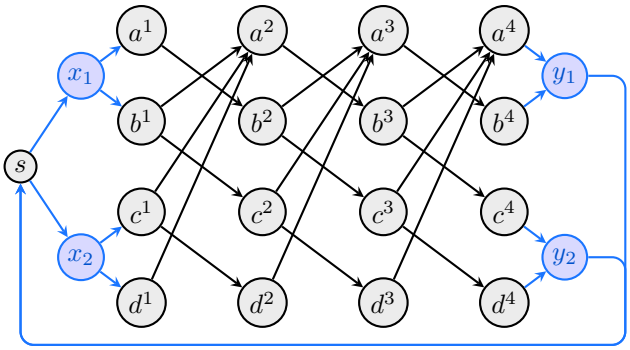

Figure 4: Illustration of how to reduce the number of entries in the target sets in Reduction 4.3 with two complete binary trees. Here $G = (\{a, b, c, d\}, \{(a, b), (b, a), (b, c), (c, a), (c, d), (d, a)\})$ and the target sets for disjunctive safety are $T_a = \{b^1, x_2, b^4, y_2\}$, $T_b = \{a^1, x_2, a^4, y_2\}$, $T_c = \{d^1, x_1, d^4, y_1\}$, and $T_d = \{c^1, x_1, c^4, y_1\}$.

The size and the construction time of the graph G', constructed by Reduction 4.3, is linear in the size of the original graph G and we have $k = \Theta(n)$ target sets. Thus if we would have a combinatorial $O(n^{3-\epsilon})$ or $O((k \cdot n^2)^{1-\epsilon})$ algorithm for disjunctive safety objectives or queries in graphs, we would immediately get a combinatorial $O(n^{3-\epsilon})$ algorithm for triangle detection, which contradicts STC (and thus BMM).

The above reduction uses a linear number of safety constraints which are all of linear size. Thus, a natural question is whether smaller safety sets would make the problem any easier. Next we argue that our result even holds for safety sets that are of logarithmic size. To this end we modify Reduction 4.3 as follows. We remove all edges incident to s and replace them by two complete binary trees. The first tree with s as root and the vertices V^1 as leaves is directed towards the leaves, the second tree with root s and leaves V^4 is directed towards s. Now for each pair v^1, v^4 one can select one vertex of each level of the trees (except for the root levels) for the set T_v such that the only safe path starting in s has to use v^1 and each safe path to s must pass v^4. As the depth of the trees is logarithmic in the number of leaf vertices, we get sets of logarithmic size. The construction with the binary trees is illustrated in Figure 4.

Next we present an $\Omega(m^{2-o(1)})$ lower bound for disjunctive objective/query safety in sparse MDPs.

Theorem 4.5. *There is no $O(m^{2-\epsilon})$ or $O((k \cdot m)^{1-\epsilon})$ algorithm (for any $\epsilon > 0$) for disjunctive safety objectives/queries in MDPs under Conjecture 2.12 (i.e., unless OVC and SETH fail). In particular, there is no such algorithm for deciding whether the winning set is non-empty or deciding whether a specific vertex is in the winning set.*

To prove the above, we give a linear time reduction from OV to disjunctive safety objectives/queries.

Reduction 4.6. *Given two sets S_1, S_2 of d-dimensional vectors, we build the following MDP P.*

- *The vertices V of the MDP P are given by a start vertex s, vertices S_1 and S_2 representing the sets of vectors, and vertices $\mathcal{C} = \{c_i \mid 1 \leq i \leq d\}$ representing the coordinates. The edges E of P are defined as follows: the start vertex s has an edge to every vertex of S_1 and every vertex $v \in S_2$ has an edge to s; further for each $x \in S_1$ there is an edge* to *$c_i \in \mathcal{C}$ iff $x_i = 1$ and for each $y \in S_2$ there is an edge* from *$c_i \in \mathcal{C}$ iff $y_i = 1$.*
- *The set of vertices V is partitioned into player 1 vertices $V_1 = \{s\} \cup S_2$ and random vertices $V_R = S_1 \cup \mathcal{C}$. Moreover, the probabilistic transition function for each vertex $v \in V_R$ chooses among v's successors uniformly at random.*

The reduction is illustrated on an example in Figure 5.

Lemma 4.7. *Given two sets S_1, S_2 of d-dimensional vectors, the corresponding MDP P given by Reduction 4.6 and $T_v = \{v\}$ for $v \in S_2$ the following statements are equivalent*

1. *There exist orthogonal vectors $x \in S_1$, $y \in S_2$.*
2. *$s \in \bigvee_{v \in S_2} \langle\!\langle 1 \rangle\!\rangle_{as} (P, \mathit{Safety}(T_v))$*
3. *$s \in \langle\!\langle 1 \rangle\!\rangle_{as} (P, \bigvee_{v \in S_2} \mathit{Safety}(T_v))$*
4. *The winning set $\bigvee_{v \in S_2} \langle\!\langle 1 \rangle\!\rangle_{as} (P, \mathit{Safety}(T_v))$ is non-empty.*
5. *The winning set $\langle\!\langle 1 \rangle\!\rangle_{as} (P, \bigvee_{v \in S_2} \mathit{Safety}(T_v))$ is non-empty.*

Proof. W.l.o.g. we assume that the 1-vector, i.e., the vector with all coordinates being 1, is contained in S_2 (adding the 1-vector does not change the result of the OV instance). Then a play in the MDP P proceeds as follows. Starting from s, player 1 chooses a vertex $x \in S_1$; then a vertex $c \in \mathcal{C}$ and then a vertex $y \in S_2$ are picked randomly; then the play goes back to s, starting another cycle of the play.

(1)⇒(2): Assume there are orthogonal vectors $x \in S_1$, $y \in S_2$. Now player 1 can satisfy $\mathrm{Safety}(T_y)$ in the MDP P by simply going to x whenever the play is in s. The random player will then send it to some adjacent $c \in \mathcal{C}$ and then to some adjacent vertex in S_2, but as x and y are orthogonal, this c is not connected to y. Thus the play will never visit y.

(2)⇒(3): Assume $s \in \bigvee_{v \in S_2} \langle\!\langle 1 \rangle\!\rangle_{as} (P, \mathrm{Safety}(T_v))$. Then there is a vertex $y \in S_2$ such that $s \in \langle\!\langle 1 \rangle\!\rangle_{as} (P, \mathrm{Safety}(T_y))$. Now we can enlarge the objective to $\bigvee_{v \in S_2} \mathrm{Safety}(T_v)$ and obtain $s \in \langle\!\langle 1 \rangle\!\rangle_{as} (P, \bigvee_{v \in S_2} \mathrm{Safety}(T_v))$.

(3)⇒(1): Assume $s \in \langle\!\langle 1 \rangle\!\rangle_{as} (P, \bigvee_{v \in S_2} \mathrm{Safety}(T_v))$ and consider a corresponding strategy σ. W.l.o.g. we can assume that this strategy is memoryless [38]. Thus whenever the play is in s, it picks a fixed $x \in S_1$ as the next vertex. Assume towards contradiction that there is no orthogonal vector $y \in S_2$ for x. Then for each $y \in S_2$ we have that there is a $c \in \mathcal{C}$ connecting x to y. In each cycle of the play one goes from s to x and then by random choice to some vertex in S_2. By the above, each of the vertices in S_2 has a non-zero probability to be reached in this cycle, which can, for each fixed n, be lower bounded by a constant p. Thus after n cycles in the play with probability at least $p^{|S_2|}$ all vertices in S_2 have been visited and thus none of the

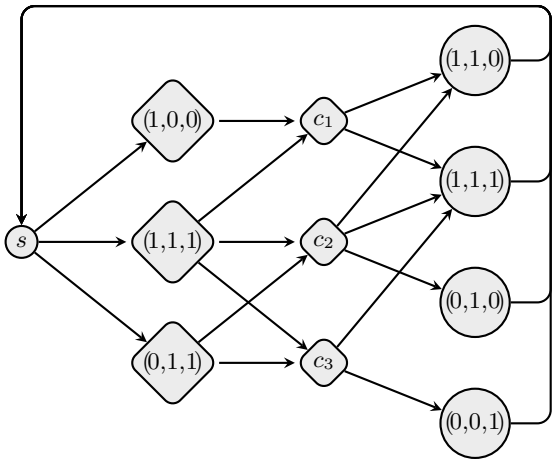

Figure 5: Illustration of Reduction 4.6, for $S_1 = \{(1,0,0),(1,1,1),(0,1,1)\}$ and $S_2 = \{(1,1,0),(1,1,1),(0,1,0),(0,0,1)\}$.

safety objectives is satisfied, a contradiction to the assumption that with probability 1 at least one safety objective is satisfied. Thus there must exist a vector $y \in S_2$ orthogonal to x.

(2)⇔(4) & (3)⇔(5): Notice that when removing s from P we get an acyclic MDP and thus each infinite path has to contain s infinitely often. Certainly if s is in the a.s. winning set, this set is non-empty. Thus let us assume there is a vertex v different from s with a winning strategy σ. All (winning) paths starting in v cross s after at most 3 steps and thus σ must be also winning when starting in s. □

The number of vertices in the MDP P, constructed by Reduction 4.6, is $O(N)$, the number of edges m is $O(N \log N)$ (recall that $d \in O(\log N)$), we have $k \in \Theta(N)$ target sets, and the construction can be performed in $O(N \log N)$ time. Thus, if we would have an $O(m^{2-\epsilon})$ or $O((k \cdot m)^{1-\epsilon})$ algorithm for disjunctive queries or disjunctive objectives of safety objectives for any $\epsilon > 0$, we would immediately get an $O(N^{2-\epsilon})$ algorithm for OV, which contradicts OVC (and thus SETH).

5 Algorithms for MDPs with Streett objectives

In this section we extend algorithms for graphs with Streett objectives to MDPs. In particular we prove the following theorem.

Theorem 5.1. *For an MDP P with Streett objectives defined by Streett pairs* $\mathrm{SP} = \{(L_i, U_i) \mid 1 \leq i \leq k\}$ *with* $b = \sum_{i=1}^{k}(|L_i| + |U_i|)$ *the almost-sure winning set can be computed in* $O(\min(n^2, m\sqrt{m \log n}) + b \log n)$ *time.*[5]

[5]It can also be computed in $O((\mathrm{MEC} + b) \cdot k)$ time, which is faster for some combinations of parameters with $k = O(\log n)$.

We first describe the basic algorithm for MDPs with Streett objectives, which uses an algorithm for MEC-decomposition as a black box. We then develop a new algorithm that opens up this black box and after an initial computation of the MEC-decomposition only uses strongly connected components and random attractor computations (Section 5.3). This algorithm reveals strong similarities to the known algorithms for graphs with Streett objectives. We then extend the two approaches that lead to the best asymptotic running times on graphs, one for dense graphs (Section 5.4) and one for sparse (Section 5.4) graphs, to MDPs. The algorithms for graphs are based on finding "good" strongly connected subgraphs and then determining which vertices can reach these "good components". For MDPs we find *good end-components* and then compute almost-sure reachability with the union of all good end-components as target set to determine the almost-sure winning set. We first show that this approach is correct (Section 5.1, see also [8, Chap. 10.6.3]) and then provide algorithms that identify all good end-components.

5.1 Good End-Components

Good end-components are also useful for other objectives such as Rabin objectives. The results of this subsection are valid for all objectives for which whether an infinite path ω belongs to the objective depends only on the vertices $\mathrm{Inf}(\omega)$ that occur infinitely often in ω. For such objectives we show that determining the winning set is equivalent to computing almost-sure reachability of the union of all good end-components. We define a good end-component as an end-component for which the objective is satisfied if exactly the vertices of the end-component are visited infinitely often.

Definition 5.2 (Good End-Component)**.** *Given an MDP P and an objective ψ, an end-component X of P such that each path $\omega \in \Omega$ with $\mathrm{Inf}(\omega) = X$ is in ψ is called a* good ψ end-component*.*

For a Streett objective the following is an equivalent definition.

Definition 5.3 (Good Streett End-Component)**.** *Given an MDP P and a set $\mathrm{SP} = \{(L_i, U_i) \mid 1 \leq i \leq k\}$ of Streett pairs, a* good Streett end-component *is an end-component X of P such that for each $1 \leq i \leq k$ either $L_i \cap X = \emptyset$ or $U_i \cap X \neq \emptyset$.*

The importance of end-components lies in the fact that player 1 can keep the play in an end-component forever and can visit each vertex in the end-component almost surely and also almost surely infinitely often (Lemma 5.4). This implies that in a good end-component player 1 has an almost-sure winning strategy (Lemma 5.5) and thus player 1 has an almost-sure winning strategy from every vertex that can almost-surely reach a good end-component (Lemma 5.6 and Corollary 5.7). This shows the soundness of the approach of determining the almost-sure winning set for an objective determined by $\mathrm{Inf}(\omega)$ by computing almost-sure reachability of the union of all good end-components.

Lemma 5.4. *Given an MDP P and an end-component X, player 1 has a strategy from each vertex of X such that all vertices of X are almost-surely reached infinitely often and only vertices of X are visited.*

Proof. We define a strategy σ as follows: Choose some arbitrary numbering of the vertices in X. The (not memoryless) strategy of player 1 is to first follow a shortest path within the end-component (with, say, lexicographic tie breaking) to the first vertex from the current position of the play until this vertex is reached, then a shortest path within the end-component to the second vertex and so on, until he starts with the first vertex again. This is possible because an end-component is a strongly connected subgraph. Since an end-component has no outgoing random edges, the play does not leave the end-component when player 1 plays this strategy. Let $\ell = |X|$ and let α be the smallest positive transition probability in the MDP. Then the probability that the first chosen shortest path is followed with the above strategy is at least α^ℓ and the probability that a sequence of ℓ shortest paths within X are followed and thus all vertices of X are visited is at least α^{ℓ^2}. Thus the probability that not all vertices in X were visited after $q \cdot \ell^2$ steps is at most $(1 - \alpha^{\ell^2})^q$, which goes to 0 when q goes to infinity. Hence player 1 has a strategy such that all vertices in X are visited with probability 1. By the same argument all vertices in X are visited infinitely often with probability 1 because the probability that some vertex is not visited after some finite prefix of length $t \cdot \ell^2$ can be bounded by $(1 - \alpha^{\ell^2})^{(q-t)}$. □

Lemma 5.5. *Player 1 has a strategy σ from each vertex in a good ψ end-component X to satisfy ψ almost-surely.*

Proof. By Lemma 5.4 player 1 has a strategy that almost-surely visits all nodes in X infinitely often. By the definition of good ψ end-component, all paths visiting all nodes in X infinitely often are in ψ. Hence, the strategy given by Lemma 5.4 is also almost-sure winning for ψ. □

Lemma 5.6. *Given an MDP P, an objective ψ that is determined by* $\mathrm{Inf}(\omega)$*, and a set S of almost-sure winning nodes we have that if $v \in \langle\!\langle 1 \rangle\!\rangle_{as}(P, \mathit{Reach}(S))$, then also $v \in \langle\!\langle 1 \rangle\!\rangle_{as}(P, \psi)$.*

Proof. Assume $v \in \langle\!\langle 1 \rangle\!\rangle_{as}(P, \mathrm{Reach}(S))$ and consider the following strategy. Start with the strategy for reaching S and as soon as one vertex s of S is reached switch to the almost-sure winning strategy of s. As S is (almost-surely) reached within a finite number of steps, the vertices visited by the strategy for reaching S does not affect the objective ψ. □

Corollary 5.7 (Soundness of Good End-Components)**.** *For a set of good end-components $\mathcal{X}$ and an objective ψ that is determined by* $\mathrm{Inf}(\omega)$ *we have that $\langle\!\langle 1 \rangle\!\rangle_{as}(P, \mathit{Reach}(\bigcup_{X \in \mathcal{X}} X))$ is contained in $\langle\!\langle 1 \rangle\!\rangle_{as}(P, \psi)$.*

Another conclusion we can draw from the above lemmata is that if a MEC contains a good end-component, then player 1 has an almost-sure winning strategy for the whole MEC because he can reach the good end-component almost-surely from every vertex of the MEC. We exploit this observation in the improved algorithm for coBüchi objectives in Section 6.4.

Corollary 5.8 (of Lemmata 5.4 and 5.6)**.** *Given an MDP P and an objective ψ that is determined by* $\mathrm{Inf}(\omega)$*, if a MEC X contains an almost-sure winning vertex (e.g. a good end-component X), then all vertices in X are almost-sure winning for player 1.*

To show the completeness of the approach of computing good end-components, we have to argue that every vertex from which player 1 can satisfy the objective almost-surely has also a strategy to reach a good end-component almost-surely. For this we need two rather technical lemmata. The intuition behind Lemma 5.9 is that if a random vertex occurs infinitely often on a path, then almost-surely also each of its successors appears infinitely often on that path. Thus we can argue that vertex sets that are reached infinitely often with positive probability are closed under random edges and hence SCCs within such sets of vertices are end-components (Lemma 5.10). To show completeness (Proposition 5.11) we then use a set of paths in the objective that are reached with positive probability to show that the vertices that these paths use infinitely often form good end-components. A similar proof is given for Büchi objectives in [21].

Lemma 5.9. *Given an MDP P, a strategy σ of player 1, the set Ω_σ of infinite paths starting at a vertex v that are compatible with the strategy σ, and a vertex $a \in V_R$ with $\Pr_\sigma(\{\omega \in \Omega_\sigma \mid a \in \mathrm{Inf}(\omega)\}) = p$, for each successor b of a we have $\Pr_\sigma(\{\omega \in \Omega_\sigma \mid a \in \mathrm{Inf}(\omega), b \in \mathrm{Inf}(\omega)\}) = p$ and $\Pr_\sigma(\{\omega \in \Omega_\sigma \mid a \in \mathrm{Inf}(\omega), b \notin \mathrm{Inf}(\omega)\}) = 0$.*

Proof. Whenever the strategy visits node a, with some constant probability q the play continues in b. Thus the probability that b was visited less than ℓ times after a was visited n times is upper bounded by $(1-q^\ell)^{n/\ell}$ which goes to 0 with increasing n. Thus, we have $\Pr_\sigma(\{\omega \in \Omega_\sigma \mid a \in \mathrm{Inf}(\omega), b \notin \mathrm{Inf}(\omega)\}) = 0$ and hence for the complement set $\Pr_\sigma(\{\omega \in \Omega_\sigma \mid a \in \mathrm{Inf}(\omega), b \in \mathrm{Inf}(\omega)\}) = p$. □

Lemma 5.10. *Given an MDP P, a strategy σ of player 1, the set Ω_σ of infinite paths starting at a vertex v that are compatible with the strategy σ, a set $\Omega' \subseteq \Omega_\sigma$, and the set of vertices $S = \{a \mid \Pr_\sigma(\{\omega \mid a \in \mathrm{Inf}(\omega), \omega \in \Omega'\}) > 0\}$, then for each SCC C of S and each vertex $a \in C \cap V_R$ all successors of a are contained in C, i.e., C is an end-component of P.*

Proof. Consider an SCC C, a vertex $a \in C \cap V_R$, and a successor b. Then by definition $\Pr_\sigma(\{\omega \mid a \in \mathrm{Inf}(\omega), \omega \in \Omega'\}) = p$ for a $p > 0$ and by Lemma 5.9 we get $\Pr_\sigma(\{\omega \mid a \in \mathrm{Inf}(\omega), b \notin \mathrm{Inf}(\omega), \omega \in \Omega_\sigma\}) = 0$ and thus $\Pr_\sigma(\{\omega \mid a \in \mathrm{Inf}(\omega), b \in \mathrm{Inf}(\omega),\ \omega \in \Omega'\}) = p$, i.e., $b \in S$. For each of the paths ω in the latter set we have a path from b to a consisting solely of nodes in $\mathrm{Inf}(\omega)$. As in P there are just finitely many paths from b to a at least one must have non-zero probability and thus is also contained in S. Hence, b belongs to the SCC C. □

Proposition 5.11 (Completeness of Good End-Components)**.** *Given an MDP P with an objective ψ determined by $\mathrm{Inf}(\omega)$ and let $\mathcal{X}$ be the set of all good ψ end-components, then $\langle\!\langle 1 \rangle\!\rangle_{as}(P, \psi)$ is contained in $\langle\!\langle 1 \rangle\!\rangle_{as}(P, \mathit{Reach}(\cup_{X \in \mathcal{X}} X))$.*

Proof. For a vertex $v \in \langle\!\langle 1 \rangle\!\rangle_{as}(P, \psi)$, fix a strategy σ of player 1 such that the objective is satisfied almost-surely. Let P_σ be the sub-MDP of P that consists of the vertices that are visited infinitely often with non-zero probability when player 1 follows strategy σ. Note that by Lemma 5.10 each SCC of P_σ is an end-component of P. Moreover, σ is a strategy for almost-surely reaching P_σ (each infinite path has to visit at least one vertex infinitely often).

It remains to show that each vertex of P_σ can almost-surely reach a good end component. We will actually show that each vertex of P_σ is already contained in a good end component. To this end let Ω_σ be the set of infinite paths starting at v that are compatible with the strategy σ and satisfy the objective. For an arbitrary node u of P_σ we consider all paths $\omega \in \Omega_\sigma$ with $u \in \mathrm{Inf}(\omega)$ and group them by $\mathrm{Inf}(\omega)$. At least one of these groups has non-zero probability, as there are only finitely many possible sets $\mathrm{Inf}(\omega)$ and $u \in \mathrm{Inf}(\omega)$ has non-zero probability. Let us consider one of the groups of paths Ω_σ^S with non-zero probability and the corresponding set $S = \mathrm{Inf}(\omega)$ for $\omega \in \Omega_\sigma^S$. By Lemma 5.10 the set S is closed under random edges. Moreover, as in each path $\omega \in \Omega_\sigma$ the vertices $\mathrm{Inf}(\omega)$ are strongly connected, the set S is also strongly connected and thus an end-component. Finally, as the paths $\omega \in \Omega_\sigma^S$ satisfy the objective and the objective ψ is determined by $\mathrm{Inf}(\omega) = S$, the set S forms a good end component. Hence, we have shown that each vertex of P_σ is contained in a good ψ end-component, which completes the proof. □

5.2 Algorithm Preliminaries

We introduce some additional notation for the algorithms for MDPs with Streett and Rabin objectives. For a set $\mathrm{RP} = \{(L_i, U_i) \mid 1 \leq i \leq k\}$ of Rabin pairs or a set $\mathrm{SP} = \{(L_i, U_i) \mid 1 \leq i \leq k\}$, let $b = \sum_{i=1}^{k}(|L_i| + |U_i|)$. A *strongly connected component* (SCC) is a *maximal* strongly connected subgraph. A single vertex is considered strongly connected. An SCC without outgoing edges is a *bottom SCC*, one without incoming edges a *top SCC*. The *reverse graph* *RevG* is constructed by reversing the direction of all edges of the graph G. In a graph $G = (V, E)$ the set of vertices $E(v)$ for some vertex v denotes the set of vertices $w \in V$ for which $(v, w) \in E$. The out-degree of $v \in V$ in G is denoted with $\mathit{Outdeg}_G(v)$, its in-degree with $\mathit{Indeg}_G(v)$. Let MEC denote the runtime to compute the maximal end-component decomposition of an MDP; we assume $\mathrm{MEC} = \Omega(m)$. Further we assume that each vertex in the input MDP has at least one outgoing edge, and thus we have $m = \Omega(n)$.

Definition 5.12 (Random Attractor)**.** *In an MDP $P = ((V, E), (V_1, V_R), \delta)$ the* random attractor *$\mathit{Attr}(P, W)$ of a set of vertices $W \subseteq V$ is defined as $\mathit{Attr}(P, W) = \bigcup_{j \geq 0} Z_j$ where $Z_0 = W$ and Z_j for $j > 0$ is defined recursively as $Z_{j+1} = Z_j \cup \{v \in V_R \mid E(v) \cap Z_j \neq \emptyset\} \cup \{v \in V_1 \mid E(v) \subseteq Z_j\}$. The random attractor $\mathit{Attr}(P, W)$ can be computed in $O(\sum_{v \in \mathit{Attr}(P,W)} \mathit{Indeg}(v))$ time [9, 30].*

All the algorithms for Streett objectives maintain vertex sets that are candidates for good end-components. For such a vertex set S we (a) refine the maintained sets according to the SCC decomposition of $P[S]$ and (b) for a set of vertices W for which we know that it cannot be contained in a good end-component, we remove its random attractor from S. The following lemma shows the correctness of these operations.

Lemma 5.13. *Given an MDP $P = ((V, E), (V_1, V_R), \delta)$, let X be an end-component with $X \subseteq S$ for some $S \subseteq V$. We have*

(a) $X \subseteq C$ for one SCC C of $P[S]$ and

(b) $X \subseteq S \setminus \mathit{Attr}(P', W) = \emptyset$ *for each* $W \subseteq V \setminus X$ *and each sub-MDP* P' *containing* X.

Proof. Property (a) holds since every end-component induces a strongly connected sub-MDP. We prove Property (b) by showing that $\mathit{Attr}(P', W)$ does not contain a vertex of X by induction over the recursive definition of a random attractor. Let the sets Z_j be as in Definition 5.12 and let $E'(v)$ be the vertices to which v has an edge in P'. We have $Z_0 = W$ and thus $Z_0 \cap X = \emptyset$. Assume we have $Z_j \cap X = \emptyset$ for some $j \geq 0$. No vertex of $V_R \cap X$ has an outgoing edge to $V \setminus X$ and thus the set $X \cap \{v \in V_R \mid E'(v) \cap Z_j \neq \emptyset\}$ is empty. Further every vertex in $V_1 \cap X$ has an outgoing edge to a vertex in X. Hence also $X \cap \{v \in V_1 \mid E'(v) \subseteq Z_j\}$ is empty and we have that $Z_{j+1} \cap X = \emptyset$. □

Let X be a good Streett end-component. Then $X \cap U_i = \emptyset$ implies $X \cap L_i = \emptyset$. Thus if $S \cap U_i = \emptyset$ for some vertex set S and some index i, then we have $U_i \subseteq V \setminus X$ for each end-component $X \subseteq S$. Hence we obtain the following corollary.

Corollary 5.14. *Given an MDP* P*, let* X *be a* good *Streett end-component with* $X \subseteq S$ *for some* $S \subseteq V$*. For each* i *with* $S \cap U_i = \emptyset$ *it holds that* $X \subseteq S \setminus \mathit{Attr}(P[S], L_i \cap S)$.

5.3 Improving Upon the Basic Algorithm

In Algorithm StreettMDPBasic, the basic algorithm for MDPs with Streett objectives, we maintain a set of already identified (maximal) good end-components goodEC, which is initially empty, and a set of candidate end-components $\mathcal{X}$, which is initialized with the MECs of the input MDP P. In each iteration of the while-loop we remove an end-component X from $\mathcal{X}$ and check whether it is a good end-component. For this check we find sets U_i for $1 \leq i \leq k$ that do not intersect with X and identify vertices in $X \cap L_i$ for such an i as "bad vertices" B. If there are no bad vertices, then X is a good end-component and added to goodEC. Otherwise the bad vertices and their random attractor within X are removed from X. On the sub-MDP induced by the remaining vertices of X we compute the MEC-decomposition, which identifies all remaining candidate end-components among the vertices of X. The new candidates are then added to $\mathcal{X}$. If the algorithm finds good end-components, it returns the almost-sure winning set for the reachability of the union of them.

Proposition 5.15 (Runtime of Algorithm StreettMDPBasic)**.** *Algorithm* StreettMDPBasic *can be implemented to run in* $O((\mathrm{MEC} + b)\min(n, k))$ *time.*

Proof. The initialization of $\mathcal{X}$ with all MECs of the input MDP P can clearly be done in $O(\mathrm{MEC})$ time. Further by Theorem 3.1 the almost-sure reachability computation after the while-loop can be done in $O(\mathrm{MEC})$ time.

Let X_v denote the end-component of $\mathcal{X}$ currently containing an arbitrary, fixed vertex $v \in V$ during Algorithm StreettMDPBasic. In each iteration of the while-loop in which X_v is considered either (a) $B = \emptyset$ and X_v will not be considered further or (b) the number of vertices in X_v is reduced by at least one and we have for some $1 \leq i \leq k$ that $X_v \cap L_i \neq \emptyset$ before the iteration of the while-loop and $X_v \cap L_i = \emptyset$ after the while-loop. Thus each vertex and each edge of the MDP P is considered in at most $O(\min(n, k))$ iterations of the while-loop.

Algorithm StreettMDPBasic: Basic Algorithm for MDPs with Streett Objectives

Input : an MDP $P = ((V, E), (V_1, V_R), \delta)$ and Streett pairs $\mathrm{SP} = \{(L_i, U_i) \mid 1 \leq i \leq k\}$
Output: $\langle\!\langle 1 \rangle\!\rangle_{as}(P, \mathrm{Streett}(\mathrm{SP}))$

```
1  goodEC ← ∅
2  𝒳 ← allMECs(P)
3  while 𝒳 ≠ ∅ do
4      remove some X ∈ 𝒳 from 𝒳
5      B ← {x ∈ X | ∃i s.t. x ∈ L_i and X ∩ U_i = ∅}
6      if B ≠ ∅ then
7          X ← X \ Attr(P[X], B)
8          𝒳 ← 𝒳 ∪ allMECs(P[X])
9      else
10         goodEC ← goodEC ∪ {X}
11 return ⟨⟨1⟩⟩_as (P, Reach(⋃_{X∈goodEC} X))
```

Consider the jth iteration of the while-loop; let X_j denote the set removed from $\mathcal{X}$ in this iteration and let $\mathit{bits}(X_j) = \sum_{i=1}^{k}(|L_i \cap X_j| + |U_i \cap X_j|)$. Assume that each vertex has a list of the sets L_i and U_i for $1 \leq i \leq k$ it belongs to. (We can generate these lists from the lists of the Streett pairs in $O(b)$ time at the beginning of the algorithm.) Then we can determine B by going through all lists of the vertices in X_j in $O(|X_j| + b_j)$ time, which amounts to $O((n + b)\min(n, k))$ total time over all iterations of the while-loop. The random attractor computed in Line 7 is removed and not considered further, thus its computation takes $O(m)$ time over the whole algorithm (see Definition 5.12). The computation of all MECs in $P[X_j]$ takes total time $O(\mathrm{MEC} \cdot \min(n, k))$ over all iterations of the while loop. Thus the whole algorithm can be implemented in $O((\mathrm{MEC} + b)\min(n, k))$ total time. □

Proposition 5.16 (Soundness of Algorithm StreettMDPBasic)**.** *Let W be the set returned by Algorithm* StreettMDPBasic*. We have $W \subseteq \langle\!\langle 1 \rangle\!\rangle_{as}(P, \mathit{Streett}(\mathrm{SP}))$.*

Proof. By Corollary 5.7 it is sufficient to show that every set $X \in \mathsf{goodEC}$ is a good end-component. The algorithm explicitly checks immediately before X is added to goodEC that we have for each $1 \leq i \leq k$ either $L_i \cap X = \emptyset$ or $U_i \cap X \neq \emptyset$. Thus it only remains to show that X is an end-component when it is added to goodEC. Before a set is added to goodEC, the same set is contained in the set $\mathcal{X}$. We show that all sets in $\mathcal{X}$ are end-components at any point in the algorithm by induction over the iterations of the while-loop in the algorithm. Before the first iteration of the while-loop the sets $X \in \mathcal{X}$ are the maximal end-components of P. Now consider an iteration in which a set X is removed from $\mathcal{X}$ and new sets are added to $\mathcal{X}$. First, some vertices and their random attractor in the sub-MDP $P[X]$ induced by X are removed from X. Let X' be the remaining set of vertices. By the definition of a random attractor there are no random edges from X' to the removed random attractor. Further, by the induction

hypothesis there are no random edges from X to $V \setminus X$. Thus there are no random edges from X' to $V \setminus X'$. Then the algorithm adds the MECs of the sub-MDP $P[X']$ to $\mathcal{X}$. Let $\hat{X}$ be one such MEC. Since $\hat{X}$ is a MEC in $P[X']$, it is a MEC in P if and only if it has no random edges from $\hat{X}$ to $V \setminus X'$. This holds by $\hat{X} \subseteq X'$ and the properties of X' established above. □

Proposition 5.17 (Completeness of Algorithm StreettMDPBasic)**.** *Let W be the set returned by Algorithm* StreettMDPBasic*. We have $\langle\!\langle 1 \rangle\!\rangle_{as}(P, \mathit{Streett}(\mathrm{SP})) \subseteq W$.*

Proof. By Proposition 5.11 it is sufficient to show that at the end of Algorithm StreettMDPBasic the union of the sets in goodEC contains all good end-components of the MDP P. We show by induction that every good end-component is a subset of either goodEC or $\mathcal{X}$ before and after each iteration of the while-loop in Algorithm StreettMDPBasic; as $\mathcal{X}$ is empty at the end of the algorithm, this implies the claim.

Before the first iteration of the while-loop, the set $\mathcal{X}$ is initialized with the MECs of P, thus the induction base holds. Let X be the set of vertices removed from $\mathcal{X}$ in an iteration of the while-loop and let X^* be the union of the good end-components contained in X. Either X is added to goodEC or we have that for some indices i the set X contains vertices of L_i but not of U_i; then for these indices the sets L_i and their random attractor are removed from X. Let $\hat{X}$ be this the updated set, i.e., $\hat{X} = X \setminus \mathit{Attr}(P[X], B)$. By Corollary 5.14 we still have $X^* \subseteq \hat{X}$ after this step. Then all MECs of $P[\hat{X}]$ are added to $\mathcal{X}$. Every good end-component contained in $\hat{X}$ is completely contained in one MEC of $P[\hat{X}]$, thus the claim continues to hold after the iteration of the while-loop. □

The essential observation towards faster algorithms for MDPs with Streett objectives is the following. Consider a set X in an iteration of the basic algorithm after some vertices in $\mathit{Attr}(P[X], B)$ were removed. We have that there are no random edges from X to the remaining vertices in the graph and further we have for each $1 \leq i \leq k$ either $L_i \cap X = \emptyset$ or $U_i \cap X \neq \emptyset$. Thus if $P[X]$ is still strongly connected, then X is a good end-component and is added to goodEC in one of the subsequent iterations of the algorithm. If, however, the sub-MDP $P[X]$ consists of multiple SCCs, then we have that the bottom SCCs of $P[X]$ are end-components in P but the remaining SCCs of $P[X]$ might have outgoing random edges within $P[X]$. Note, however, that we have for any good end-component $\hat{X}$ in $P[X]$ and any SCC C of $P[X]$ that either $\hat{X} \subseteq C$ or $\hat{X} \cap C = \emptyset$, simply by the fact that every good end-component is strongly connected (Lemma 5.13 (a)). Let $\hat{X} \subseteq C$ and let R be the random vertices of C with edges to vertices not in C. Then the vertices in R cannot intersect with $\hat{X}$ because an end-component has no outgoing random edges. Further, also the random attractor of R cannot intersect with $\hat{X}$ (Lemma 5.13 (b)). Thus we can remove $\mathit{Attr}(P[X], R)$ from $P[X]$ and all good end-components that were contained in $P[X]$ are still contained in the remaining sub-MDP. However, now the set of vertices in $C \setminus \mathit{Attr}(P[X], R)$ has no outgoing random edges. Thus if it is still strongly connected, then it is an end-component. With this observation we can avoid computing a MEC decomposition in the while-loop of the basic algorithm and instead only compute strongly connected components and random attractors, which both can be done in linear time. Note that in the improved algorithm we do not have the property that every maintained set of

vertices is an end-component (as in the basic algorithm) but still none of the maintained sets has outgoing random edges.

In this formulation the algorithm for MDPs with Streett objectives has a very similar structure to the algorithm for graphs with Streett objectives: We repeatedly remove "bad vertices" and recompute strongly connected components. The main difference is that we additionally compute random attractors. Based on this, we can indeed show that for Streett objectives the same techniques as for graphs also apply to MDPs and by this improve the runtime to the runtime for graphs plus the time to compute one MEC decomposition. This can be seen as opening up the "black-box" use of a MEC-decomposition algorithm and combining the fastest algorithms for MEC-decomposition [15, 16] and graphs with Streett objectives [28, 17]. In contrast to graphs with Streett objectives, no $O((m+b)k)$ algorithm can be achieved for small values of k. Intuitively, this is because it could be that only in a few iterations bad vertices are removed while the majority of the iterations is actually used to recompute MECs. We present the new algorithmic ideas for MDPs with Streett objectives in Algorithm StreettMDPImpr (which is only faster for large enough k) and then apply the known techniques for sparse and dense graphs in Algorithms StreettMDPSparse and StreettMDPDense, respectively, to beat the basic algorithm for all parameters except very small values of k; the basic algorithm is faster for e.g. $k = O(1)$ or $k = O(\sqrt{\log n})$ and $m = O(n^{4/3})$.

In our improved algorithms we use the data structure $D(X)$ from [28] to quickly identify and remove vertices in $X \cap L_i$ for which $X \cap U_i = \emptyset$ from a set of vertices X.

Lemma 5.18 ([28]). *After a one-time preprocessing time of $O(k)$, there is a data structure $D(X)$ for a given set X that can be initialized with the operation* `Construct`(X) *in time $O(bits(X)+|X|)$, where $bits(X) = \sum_{i=1}^{k} (|X \cap L_i| + |X \cap U_i|)$. Further it supports the operation* `Remove`$(X, D(X), B)$ *that removes a set $B \subseteq V$ from X and updates $D(X)$ accordingly in time $O(bits(B) + |B|)$ and the operation* `Bad`$(D(X))$ *that returns a pointer to the set $\{x \in X \mid \exists i$ s.t. $x \in L_i$ and $X \cap U_i = \emptyset\}$ in constant time.*

In Algorithm StreettMDPImpr we maintain a list Q of data structures of disjoint vertex sets that are candidates for good end-components. For every set S with $D(S)$ in Q we maintain that there are no random edges from S to $V \setminus S$. The list Q is initialized with the data structures of all MECs of the input MDP P. In each iteration of the outer while-loop the data structure of one vertex set S is pulled from Q. In the inner while-loop the set of "bad vertices" $\{x \in S \mid \exists i$ s.t. $x \in L_i$ and $S \cap U_i = \emptyset\}$ is identified and its random attractor is removed from S and $D(S)$. Through removing the random attractor we maintain the property that there are no random edges from S to $V \setminus S$ at this step. Thus we have that if $P[S]$ is (still) strongly connected, then $P[S]$ is a good end-component, which we identify in Line 11. If $P[S]$ does not contain an edge, we do not have to consider it further. If it contains an edge but is not strongly connected, the SCCs of $P[S]$ are identified. For each SCC C we identify its random vertices that have edges to vertices of $S \setminus C$ and remove their random attractor from C. After this step the data structure of the remaining vertices of C is added to Q. At this point we distinguish between the largest SCC and the other SCCs of $P[S]$. We construct a new data structure for all but the largest SCC and reuse the data structure of S for the largest SCC. This improves the runtime because we only spend time proportional to the smaller SCCs and a

vertex can be in a smaller SCC at most $O(\log n)$ times. Note that at this point of the algorithm the sub-MDP $P[C]$ is not necessarily strongly connected since vertices were removed after the SCC computation but we maintain the property that there are no random edges from a vertex set for which the data structure is in Q to other vertices. When the list Q becomes empty, the algorithm terminates. If good end-components were identified, the almost-sure winning set for the reachability objective of the union of the good end-components is output.

Algorithm StreettMDPImpr: New Algorithm for MDPs with Streett Objectives

Input : an MDP $P = ((V, E), (V_1, V_R), \delta)$ and Streett pairs $\mathrm{SP} = \{(L_i, U_i) \mid 1 \leq i \leq k\}$
Output: $\langle\!\langle 1 \rangle\!\rangle_{as}(P, \mathrm{Streett}(\mathrm{SP}))$

```
1  goodEC ← ∅; Q ← ∅
2  𝒳 ← allMECs(P)
3  for X ∈ 𝒳 do Q ← Q ∪ {Construct(X)}
4  while Q ≠ ∅ do
5      remove some D(S) from Q
6      while Bad(D(S)) ≠ ∅ do
7          A ← Attr(P[S], Bad(D(S)))
8          (S, D(S)) ← Remove(S, D(S), A)
9      if P[S] contains at least one edge then
10         if P[S] is strongly connected then
11             goodEC ← goodEC ∪ {S}
12         else
13             𝒞 ← SCCs(P[S]); S' ← S
14             for C ∈ 𝒞 do
15                 R ← {v ∈ V_R ∩ C | ∃w ∈ S' \ C s.t. (v, w) ∈ E}
16                 A ← Attr(P[C], R)
17                 if C is largest SCC in 𝒞 then
18                     (S, D(S)) ← Remove(S, D(S), A)
19                 else
20                     (S, D(S)) ← Remove(S, D(S), C)
21                     C ← C \ A
22                     Q ← Q ∪ {Construct(C)}
23             Q ← Q ∪ {D(S)}
24 return ⟨⟨1⟩⟩_as (P, Reach(⋃_{X ∈ goodEC} X))
```

Proposition 5.19 (Runtime Algorithm StreettMDPImpr)**.** *Algorithm* StreettMDPImpr *terminates in* $O(mn + b \log n)$ *time.*

Proof. Using the data structure of Lemma 5.18 ([28]), the initialization phase of Algo-

rithm StreettMDPImpr takes $O(k + \text{MEC} + b + n)$ time, which is in $O(mn + b)$. Further by Theorem 3.1 the almost-sure reachability computation after the outer while-loop can be done in $O(\text{MEC})$ time.

Whenever bad vertices and their random attractor are identified in lines 6–7, they are removed in Line 8 and not considered further. Thus finding bad vertices takes total time $O(n)$, identifying the random attractor of bad vertices takes total time $O(m)$ (see Definition 5.12), and removing the bad vertices and their attractor takes total time $O(m + b)$ by Lemma 5.18.

After the initialization of Q with the MECs of P, all vertex sets for which a data structure is stored in Q induce a strongly connected sub-MDP. Consider the set S when Line 13 is reached and its smallest superset $S' \supseteq S$ that was identified as strongly connected in the algorithm (i.e. S' is either a MEC of P or an SCC computed in Line 13 in a previous iteration of the algorithm). We have that S is a proper subset of S', i.e., either bad vertices were removed from S' in Line 8 or a non-empty set of random vertices was identified in Line 15. Hence any part of P is considered in at most n iterations of the outer while-loop. This implies that we can bound the total time spent in lines 13–16 with $O(mn)$.

By the same argument as for the removal of bad vertices and their attractor, the calls to `Remove` in Line 18 take total time $O(n + b)$. It remains to bound the time for the calls to `Remove` and `Construct` in lines 20–22. Note that we avoid to make these calls for the largest of the SCCs of the sub-MDP induced by S, which are computed in Line 13. Thus whenever we call `Remove` and `Construct` for an SCC C, we have $|C| \leq |S|/2$. Hence we can charge the time for `Remove` and `Construct` to the vertices of C and to $\mathit{bits}(C)$ such that every vertex v and every $\mathit{bits}(\{v\})$ is charged $O(\log n)$ times. Thus we can bound the time for lines 20–22 with $O((n + b) \log n)$. This proves the claimed runtime. □

Proposition 5.20 (Soundness of Algorithm StreettMDPImpr)**.** *Let W be the set returned by Algorithm* StreettMDPImpr*. We have $W \subseteq \langle\langle 1 \rangle\rangle_{as}(P, \mathit{Streett}(\text{SP}))$.*

Proof. By Corollary 5.7 it is sufficient to show that every set $X \in \mathsf{goodEC}$ is a good end-component. The algorithm explicitly checks immediately before X is added to goodEC in Line 11 that X contains at least one edge and is strongly connected. Further we have by the termination condition of the inner while-loop that for each $1 \leq i \leq k$ either $L_i \cap X = \emptyset$ or $U_i \cap X \neq \emptyset$. Thus it remains to show that there are no random edges from X to $V \setminus X$.

Let X' be the set of vertices for which the data structure $D(X')$ was removed from Q in the iteration of the outer while-loop in which X was added to goodEC. By the following invariant there are no random edges from X' to $V \setminus X'$.

Invariant 5.21. *For every set S for which the data structure $D(S)$ is in Q there are no random edges from S to $V \setminus S$.*

Assume the invariant holds. If X' is not equal to X, then some vertices and their random attractor within $P[X']$ were removed in the inner while-loop. By the definition of a random attractor there are no random edges from X to $X' \setminus X$ and thus to $V \setminus X$.

It remains to prove the invariant by induction over the iterations of the outer while-loop. Before the first iteration of the while-loop Q is initialized with the maximal end-components

of P and thus the invariant holds. Assume the invariant holds before the beginning of an iteration of the outer while-loop and let S be the set of vertices for which the data structure is removed from Q in this iteration. In the inner while-loop some vertices and their random attractor in $P[S]$ might be removed from S. Let S' be the remaining vertices. By the definition of a random attractor there are no random edges from S' to $S \setminus S'$ and thus by the induction hypothesis there are no random edges from S' to $V \setminus S'$.

If $P[S']$ is strongly connected, then no set is added to Q in this iteration of the while-loop. Otherwise the SCCs $\mathcal{C}$ of $P[S']$ are considered as candidates to be added to Q. For each set $C \in \mathcal{C}$ the random vertices R in C with edges to vertices in $S' \setminus C$ and their random attractor A in $P[C]$ are removed from C. Let C' be the remaining vertices. We have that there are no random edges from C' to $S' \setminus C$ by the definition of R and that there are no random edges from C' to $C \setminus C'$ by the definition of A. Thus there are no random edges from C' to $V \setminus C'$ for any set C' for which the data structure is added to Q, which shows the invariant. □

Proposition 5.22 (Completeness of Algorithm StreettMDPimpr)**.** *Let W be the set returned by Algorithm* StreettMDPimpr*. We have $\langle\!\langle 1 \rangle\!\rangle_{as}(P, \mathit{Streett}(\mathrm{SP})) \subseteq W$.*

Proof. By Proposition 5.11 it is sufficient to show that at the end of the algorithm the union of the sets in goodEC contains all good end-components of the MDP P. We show the following invariant by induction over the iterations of the outer while-loop; as Q is empty at the end of the algorithm, this implies the claim.

Invariant 5.23. *For each good end-component X of P and some set $Y \supseteq X$ either $Y \in$ goodEC or $D(Y) \in Q$ holds before and after each iteration of the outer while-loop.*

Before the first iteration of the outer while-loop, the set Q is initialized with the MECs of P, thus the induction base holds. Let S be the set of vertices for which the data structure is removed from Q in an iteration of the outer while-loop and let $\mathcal{X}_S$ be the set of good end-components contained in S. We have $X \subseteq S'$ for every $X \in \mathcal{X}_S$ after the inner while-loop by Corollary 5.14.

Since every end-component contains an edge, $P[S']$ contains at least one edge if $\mathcal{X}_S$ is not empty. Then either S' and thus all $X \in \mathcal{X}_S$ are added to goodEC or the SCCs $\mathcal{C}$ of $P[S']$ are computed. For each $X \in \mathcal{X}_S$ there exists $C \in \mathcal{C}$ such that $X \subseteq C$ by Lemma 5.13 (a); let X and C be such that $X \subseteq C$. Since X has not outgoing random edges, we have $R \cap X = \emptyset$ (Line 15) and thus also $X \subseteq C \setminus \mathit{Attr}(P[C], R)$ by Lemma 5.13 (b). The data structure of $C \setminus A$ is added to Q in lines 22 or 23, hence the claim holds after the outer while-loop. □

5.4 Algorithm for Dense MDPs with Streett Objectives

Algorithm StreettMDPdense combines Algorithm StreettMDPimpr with the ideas of the MEC-algorithm for dense MDPs of [16] and the algorithm for graphs with Streett objectives of [17]. The difference to Algorithm StreettMDPimpr lies in the search for strongly connected components. To detect a good end-component, it is essential to detect when a sub-MDP $P[S]$ remains strongly connected after some vertices and their random attractor were removed from the vertex set S for which the data structure $D(S)$ is maintained in Q.

Algorithm StreettMDPdense: Algorithm for dense MDPs with Streett Objectives

Input : an MDP $P = ((V, E), (V_1, V_R), \delta)$ and Streett pairs $\mathrm{SP} = \{(L_i, U_i) \mid 1 \le i \le k\}$
Output: $\langle\!\langle 1 \rangle\!\rangle_{as}(P, \mathrm{Streett}(\mathrm{SP}))$

```
1  goodEC ← ∅; Q ← ∅
2  𝒳 ← allMECs(P)
3  for X ∈ 𝒳 do Q ← Q ∪ {Construct(X)}
4  while Q ≠ ∅ do
5      remove some D(S) from Q
6      while Bad(D(S)) ≠ ∅ do
7          A ← Attr(P[X], Bad(D(S)))
8          D(S) ← Remove(S, D(S), A)
9      if P[S] contains at least one edge then
10         for j ← 1 to ⌈log(|S|)⌉ do
11             foreach H ∈ {G, RevG} do
12                 construct H_j[S]
13                 Bl_j ← {v ∈ S | Outdeg_H(v) > 2^j}
14                 Z ← S \ GraphReach(H_j[S], Bl_j)
15                 if Z ≠ ∅ then
16                     C ← SmallestBSCC(H_j[Z])
17                     if C = S then
18                         goodEC ← goodEC ∪ {C}
19                         continue with next iteration of while-loop
20                     if |C| ≤ |S|/2 then
21                         if H = RevG then                    /* top SCC */
22                             Q ← Q ∪ Remove(S, D(S), C)
23                             R ← {v ∈ V_R ∩ C | ∃u ∈ S \ C s.t. (v, u) ∈ E}
24                             C ← C \ Attr(P[C], R)
25                         else                                /* bottom SCC */
26                             Q ← Q ∪ Remove(S, D(S), Attr(P[S], C))
27                         Q ← Q ∪ Construct(C)
28                         continue with next iteration of while-loop
29 return ⟨⟨1⟩⟩_as (P, Reach(⋃_{X ∈ goodEC} X))
```

For this it is sufficient to identify one strongly connected component C of the sub-MDP $P[S]$: The sub-MDP is strongly connected if and only if the SCC spans the whole sub-MDP, i.e., $C = S$. As for Algorithm StreettMDPImpr, the correctness of the algorithm is based on maintaining the Invariants 5.21 and 5.23. For maintaining these invariants it makes no difference whether we compute all SCCs of $P[S]$ or just one. Whenever $P[S]$ is not strongly connected, there exists a top or bottom SCC that contains at most half of the vertices of S. In Algorithm StreettMDPDense we search for such a "small" top or bottom SCC of $P[S]$. The search for a top SCC is done by searching for a bottom SCC in the reverse graph. To search for a bottom SCC, a sparsification technique called *Hierarchical Graph Decomposition* is used. This technique was introduced by [25] for undirected graphs and extended to directed graphs and game graphs by [16]. In the level-j graph H_j of a graph H only the first 2^j outgoing edges of each vertex are considered, thus H_j has $O(n \cdot 2^j)$ edges. The main observation (Lemma 5.25) is that we can identify each bottom SCC with at most 2^j vertices by searching for bottom SCCs of H_j that only contain vertices for which all their outgoing edges in H are also in H_j. The search is started at level $j = 1$ and then j is doubled until such a bottom SCC is found in H_j. Note that $H_j = H$ for $j \geq \log n$. When a bottom SCC is identified at level j^* but not at $j^* - 1$, then this bottom SCC has $\Omega(2^{j^*})$ vertices by the above observation. Further, the number of edges in the graphs from level 1 to j^* form a geometric series. Thus the work spent in all the levels up to j^* can be bounded in terms of the number of edges in H_{j^*}, that is, the bottom SCC of size $\Omega(2^{j^*})$ is identified in $O(n \cdot 2^{j^*})$ time. By searching "in parallel" for top and bottom SCCs and charging the needed time to the identified SCC, the total runtime can be bounded by $O(n^2)$. To identify only bottom SCCs of H_j for which all the outgoing edges are present in H_j we determine the set of "blue" vertices Bl_j that have an out-degree higher than 2^j and remove vertices that can reach blue vertices before computing SCCs. In the following we provide formal definitions and proofs for Algorithm StreettMDPDense.

Definition 5.24 (Hierarchical Graph Decomposition)**.** *Let $H = (V, E)$ be a simple directed graph. We consider for $j \in \mathbb{N}$ the subgraphs $H_j = (V, E_j)$ of H where E_j contains for each vertex of V its first 2^j outgoing edges in E (for some arbitrary but fixed ordering of the outgoing edges of each vertex). Note that when $j \geq \log(\max_{v \in V} Outdeg_H(v))$, then $H_j = H$. Let the set Bl_j denote all vertices with out-degree more than 2^j in H.*

Lemma 5.25 (See e.g. [26])**.** *We use Definition 5.24.*

1. *A set $C \subseteq V \setminus Bl_j$ is a bottom SCC in H_j if and only if it is a bottom SCC in H.*
2. *If a set $C \subseteq V$ with $|C| \leq 2^j$ is a bottom SCC in H, then $C \subseteq V \setminus Bl_j$.*

Proof.
1. By $C \subseteq V \setminus Bl_j$ the outgoing edges of the vertices in C are the same in H_j and in H. Thus we have $H_j[C] = H[C]$ and C has no outgoing edges in H_j if and only if it has no outgoing edges in H.
2. In H all outgoing edges of each vertex of C have to go to other vertices of C. Thus each vertex of C has an out-degree of at most $|C| \leq 2^j$ in H. □

Proposition 5.26 (Runtime of Algorithm StreettMDPDense)**.** *Algorithm* StreettMDPDense *terminates in* $O(n^2 + b \log n)$ *time.*

Proof. Using the data structure of Lemma 5.18 ([28]), the initialization phase of Algorithm StreettMDPDense takes $O(\text{MEC} + b + n)$ time, which is in $O(n^2 + b)$ [16]. Further by Theorem 3.1 the almost-sure reachability computation after the outer while-loop can be done in $O(\text{MEC})$ time. Removing bad vertices takes total time $O(n + b)$ by Lemma 5.18. Whenever a random attractor is computed, its edges are not considered further; thus all attractor computations take $O(m)$ total time by Definition 5.12. Whenever `Remove` or `Construct` are called (after the initialization of Q), the vertices that are removed resp. added are either (1) vertices for which the size of the SCC containing them was at least halved or (2) vertices that are not considered further. For each vertex case (1) can happen at most $O(\log n)$ times and case (2) at most once, thus all calls to `Remove` or `Construct` take total time $O((n + b) \log n)$ by Lemma 5.18.

To efficiently construct the graphs H_j and compute Bl_j for $1 \leq j < \lceil \log(n) \rceil$ and $H \in \{G, RevG\}$, we maintain for all vertices a list of their incoming and outgoing edges, which we update whenever we encounter obsolete entries while constructing H_j. Each entry can be removed at most once, thus this can be done in $O(m)$ total time.

Let S be the set of vertices considered in an iteration of the outer while-loop and let $|S| = n'$. The jth iteration of the for-loop takes $O(n' \cdot 2^j)$ time because H_j contains $O(n' \cdot 2^j)$ edges and constructing H_j and Bl_j and computing reachability, SCCs, and R can all be done in time linear in the number of edges. The search in G and $RevG$ only increases the runtime by a factor of two. Further *all* iterations up to the jth iteration can be executed in time $O(n' \cdot 2^j)$ as their runtimes form a geometric series. Note that whenever a graph is not strongly connected, it contains a top SCC and a bottom SCC and one of them has at most half of the vertices. Thus in some iteration j^* a top or bottom SCC with either $C = S$ or $|C| \leq n'/2$ is found by Lemma 5.25. Since C was not found in iteration $j^* - 1$, we have $|C| = \Omega(2^{j^*})$ by Lemma 5.25.

In the case $C = S$ the vertices in S are not considered further by the algorithm. Thus we can bound the time for this iteration with $O(n' \cdot 2^{\log(n')}) = O(n'^2)$ and hence the total time for this case with $O(n^2)$.

It remains to bound the time for the case $|C| \leq n'/2$. Let $|C| = n_1$ and let c be some constant such that the time spent for the search of C is bounded by $c \cdot n_1 \cdot n'$. We denote this time for the set S over the whole algorithm with $f(n')$ and show $f(n') = 2cn'^2$ by induction as follows:

$$\begin{aligned} f(n') &\leq f(n_1) + f(n' - n_1) + cn'n_1\,, \\ &\leq 2cn_1^2 + 2c(n' - n_1)^2 + cn'n_1\,, \\ &= 2cn_1^2 + 2cn'^2 - 4cn'n_1 + 2cn_1^2 + cn'n_1\,, \\ &= 2cn'^2 + 4cn_1^2 - 3cn'n_1\,, \\ &\leq 2cn'^2\,, \end{aligned}$$

where the last inequality follows from $n_1 \leq n'/2$. □

Proposition 5.27 (Soundness of Algorithm StreettMDPdense)**.** *Let W be the set returned by Algorithm* StreettMDPdense*. We have $W \subseteq \langle\!\langle 1 \rangle\!\rangle_{as}(P, \mathit{Streett}(\mathrm{SP}))$.*

Proof. We follow the proof of Proposition 5.20. Let C be a set of vertices added to goodEC in Line 18. Since $P[C]$ is strongly connected by Lemma 5.25, we have that immediately before C is added to goodEC it was checked that $P[C]$ contains at least one edge, is strongly connected, and $\mathtt{Bad}(D(C))$ is empty. Thus it is sufficient to show that Invariant 5.21 holds in Algorithm StreettMDPdense.

Before the first iteration of the while-loop Q is initialized with the maximal end-components of P and thus the invariant holds. Assume the invariant holds before the beginning of an iteration of the outer while-loop and let S be the set of vertices for which the data structure is removed from Q in this iteration. In the inner while-loop some vertices and their random attractor in $P[S]$ might be removed from S. Let S' be the remaining vertices. By the definition of a random attractor there are no random edges from S' to $S \setminus S'$ and thus by the induction hypothesis there are no random edges from S' to $V \setminus S'$.

Then either $P[S']$ is strongly connected and no set is added to Q in this iteration of the while-loop or either a top or a bottom SCC C of $P[S']$ is identified by Lemma 5.25.

If C is a top SCC, then there are no edges from $S' \setminus C$ to C and thus $S' \setminus C$ has no outgoing random edges. Hence the invariant is maintained when $D(S' \setminus C)$ is added to Q. Then the random vertices of C with edges to vertices in $S' \setminus C$ and their random attractor are removed from C. Thus the remaining vertices of C have no random edges to $V \setminus C$ and the invariant is maintained when the data structure of this vertex set is added to Q.

If C is a bottom SCC, then there are no edges from C to $S' \setminus C$; thus the invariant is maintained when $D(C)$ is added to Q. The random attractor of C is removed from $S' \setminus C$ before the data structure of the remaining vertices is added to Q, hence the invariant is maintained in all cases. □

Proposition 5.28 (Completeness of Algorithm StreettMDPdense)**.** *Let W be the set returned by Algorithm* StreettMDPdense*. We have $\langle\!\langle 1 \rangle\!\rangle_{as}(P, \mathit{Streett}(\mathrm{SP})) \subseteq W$.*

Proof. Following the proof of Proposition 5.22, it is sufficient to show by induction over the iterations of the outer-while loop that Invariant 5.23 holds in Algorithm StreettMDPdense.

Before the first iteration of the outer while-loop, the set Q is initialized with the MECs of P, thus the induction base holds. Let S be the set of vertices for which the data structure is removed from Q in an iteration of the outer while-loop and let $\mathcal{X}_S$ be the set of good end-components contained in S. Let S' be the subset of S that is not removed in the inner while-loop. We have $X \subseteq S'$ for every $X \in \mathcal{X}_S$ by Corollary 5.14.

Since every end-component contains an edge, $P[S']$ contains at least one edge if $\mathcal{X}_S$ is not empty. Then either S' and thus all $X \in \mathcal{X}_S$ are added to goodEC (Line 18) or an SCC $C \subsetneq S'$ of $P[S']$ is identified in Line 16 by Lemma 5.25. By Lemma 5.13 (a) each $X \in \mathcal{X}_S$ is either a subset of C or of $S' \setminus C$. For $X \subseteq C$ we have $R \cap X = \emptyset$ (Line 23) since X has no outgoing random edges and thus $X \subseteq C \setminus \mathit{Attr}(P[C], R)$ by Lemma 5.13 (b). For $X \subseteq S' \setminus C$ we have $X \cap C = \emptyset$ and thus $X \subseteq S' \setminus \mathit{Attr}(P[S'], C)$ by Lemma 5.13 (b). The data structures of

$C \setminus Attr(P[C], R)$ and of $S' \setminus Attr(P[S'], C)$ are added to Q in lines 27 and either 22 or 26, hence the invariant holds after the outer while-loop. □

5.5 Algorithm for Sparse MDPs with Streett Objectives

Algorithm StreettMDPsparse combines Algorithm StreettMDPimpr with the ideas of the MEC-algorithm for sparse MDPs of [16] and the algorithm for graphs with Streett objectives of [28]. As for dense graphs, the difference to Algorithm StreettMDPimpr lies in the search for strongly connected components in the sub-MDP $P[S]$ induced by a vertex set S for which the data structure was maintained in Q and then some vertices (and their random attractor) might have been removed from it. The algorithm is based on the following observation: Whenever a strongly connected component C is not strongly connected after some vertices A were removed from it, then (a) there is a top and a bottom SCC in $P[C \setminus A]$ and (b) some vertex of the top SCC had an incoming edge from a vertex of A and some vertex of the bottom SCC had an outgoing edge to a vertex of A. We label vertices that lost an incoming edge since the last SCC computation with h (for head) and vertices that lost an outgoing edge with t (for tail). If more than $\sqrt{m/\log n}$ vertices are labeled, we remove all labels and compute SCCs as in Algorithm StreettMDPimpr; this can happen at most $\sqrt{m \log n}$ times. Otherwise we search for the smallest top or bottom SCC of $P[S]$ by searching in *lock-step* from all labeled vertices. Lock-step means that one step in each of the searches is executed before the next step of a search is started and all searches are stopped as soon as one search finishes. The search for top SCCs is done by searching for bottom SCCs in the reverse graph. Tarjan's depth-first search based SCC algorithm detects a bottom SCC in time proportional to the number of edges in the bottom SCC when the search is started from a vertex inside the bottom SCC. As there are at most $\sqrt{m/\log n}$ parallel searches, the time for all the lock-step searches is $O(\sqrt{m/\log n})$ times the number of edges in the smallest top or bottom SCC of $P[S]$. Since each edge can be in the smallest SCC at most $O(\log n)$ times, this leads to a total runtime of $O(m\sqrt{m \log n})$. Whenever an SCC is identified, the labels of its vertices are removed. The Invariants 5.21 and 5.23 are maintained as in Algorithm StreettMDPimpr.

Lemma 5.29 (Label Invariant)**.** *In Algorithm* StreettMDPsparse *the following invariant is maintained for every set S for which the data structure $D(S)$ is in Q: Either (1) no vertex of S is labeled and $P[S]$ is strongly connected or (2) in each top SCC of $P[S]$ at least one vertex is labeled with h and in each bottom SCC of $P[S]$ at least one vertex is labeled with t.*

Proof. The proof is by induction over the iterations of the outer while-loop. After the initialization of Q with the MECs of P no vertex is labeled and every set S with $D(S) \in Q$ is strongly connected. Let now S denote the set for which $D(S)$ is removed from Q at the beginning of an iteration of the outer while-loop and assume the invariant holds for S.

Observation. *We have for non-empty vertex sets W and $Z = W \setminus Y$ with $Y \subsetneq W$ that if C is a top (bottom) SCC in $P[Z]$ but had incoming (outgoing) edges in $P[W]$, then these incoming (outgoing) edges were from (to) vertices in Y. Thus when the invariant holds for W and we label each vertex of Z with an incoming edge from Y with h and each vertex of Z with an outgoing edge to Y with t, then the invariant holds for Z.*

Algorithm StreettMDPSparse: Algorithm for sparse MDPs with Streett Objectives

Input : an MDP $P = ((V, E), (V_1, V_R), \delta)$ and Streett pairs $\text{SP} = \{(L_i, U_i) \mid 1 \leq i \leq k\}$
Output: $\langle\!\langle 1 \rangle\!\rangle_{as}(P, \text{Streett}(\text{SP}))$

```
1  goodEC ← ∅; Q ← ∅; 𝒳 ← allMECs(P)
2  for X ∈ 𝒳 do Q ← Q ∪ {Construct(X)}
3  while Q ≠ ∅ do
4      remove some D(S) from Q
5      while Bad(D(S)) ≠ ∅ do
6          A ← Attr(P[X], Bad(D(S)))
7          (S, D(S)) ← Remove(S, D(S), A)
8          add label h (t) to vertices that just lost an incoming (outgoing) edge
9      H ← {v ∈ S | h ∈ label(v)}; T ← {v ∈ S | t ∈ label(v)}
10     if P[S] contains at least one edge then
11         if |H| + |T| = 0 then goodEC ← goodEC ∪ {S}
12         else if |H| + |T| ≥ √(m/log n) then
               /* like Algorithm StreettMDPImpr plus maintaining labels   */
13             remove all labels from S
14             𝒞 ← SCCs(P[S]); S' ← S
15             for C ∈ 𝒞 do
16                 A ← Attr(P[C], {v ∈ V_R ∩ C | ∃w ∈ S' \ C s.t. (v, w) ∈ E})
17                 add label h (t) to vertices with incoming (outgoing) edge from (to) A
18                 if C is largest SCC in 𝒞 then (S, D(S)) ← Remove(S, D(S), A)
19                 else
20                     (S, D(S)) ← Remove(S, D(S), C); C ← C \ A
21                     Q ← Q ∪ {Construct(C)}
22             Q ← Q ∪ D(S)
23         else
24             Search in lock-step from each v ∈ T in G[S] and from each v ∈ H in RevG[S],
               terminate when first search has found a bottom SCC C
               /* like Alg. StreettMDPDense plus maintaining labels       */
25             if C = S then goodEC ← goodEC ∪ {S}
26             else
27                 remove all labels from C
28                 if C is bottom SCC in RevG[S] then                  /* top SCC */
29                     Q ← Q ∪ Remove(S, D(S), C)
30                     C ← C \ Attr(P[C], {v ∈ V_R ∩ C | ∃u ∈ S \ C s.t. (v, u) ∈ E})
31                 else                                               /* bottom SCC */
32                     Q ← Q ∪ Remove(S, D(S), Attr(P[S], C))
33                 add label h (t) to vertices that just lost an incoming (outgoing) edge
34                 Q ← Q ∪ Construct(C)
35 return ⟨⟨1⟩⟩_as (P, Reach(⋃_{X ∈ goodEC} X))
```

By this observation the invariant remains to hold for S after the inner while-loop. In the case $|H| + |T| \geq \sqrt{m/\log n}$ all labels are removed from S and then each SCC C of $P[S]$ is considered separately. Note that for each C the invariant holds and thus the invariant remains to hold for the set C added to Q after the vertices in A were removed and the corresponding labels were added in Line 17. In the case $|H| + |T| < \sqrt{m/\log n}$ a bottom or top SCC C of $P[S]$ is identified and all labels of C are removed. The invariant holds for C and thus the invariant remains to hold for the set C added to Q after vertices were removed from C in Line 30 and the corresponding labels were added in Line 33. By the above observation with $W = S$ and $Y = \mathit{Attr}(P[S], C)$ the invariant also holds for the set $S \setminus \mathit{Attr}(P[S], C)$ for which the data structure is added to Q after the corresponding labels are added in Line 33. □

Proposition 5.30 (Runtime of Algorithm StreettMDPsparse)**.** *Algorithm* StreettMDPsparse *takes* $O(m\sqrt{m \log n} + b \log n)$ *time.*

Proof. Using the data structure of Lemma 5.18 ([28]), the initialization phase of Algorithm StreettMDPsparse takes $O(\text{MEC} + b + n)$ time, which is in $O(m\sqrt{m} + b)$ [16]. Further by Theorem 3.1 the almost-sure reachability computation after the outer while-loop can be done in $O(\text{MEC})$ time. Removing bad vertices takes total time $O(n+b)$ by Lemma 5.18. Since a label is added only when an edge is not considered further by the algorithm, the total time for adding and removing labels is $O(m)$. Whenever a random attractor is computed, its edges are not considered further; thus all attractor computations take $O(m)$ total time by Definition 5.12. Note that whenever a graph is not strongly connected, it contains a top SCC and a bottom SCC and one of them has at most half of the vertices. Thus whenever a top or bottom SCC C with $C \subsetneq S$ is identified in Line 24, then $|C| \leq |S|/2$. This implies by Lemma 5.29 that whenever `Remove` or `Construct` are called (after the initialization of Q), the vertices that are removed resp. added are either (1) vertices for which the size of the SCC containing them was at least halved or (2) vertices that are not considered further. Case (1) can happen at most $O(\log n)$ times, thus all calls to `Remove` or `Construct` take total time $O((n + b) \log n)$ by Lemma 5.18.

It remains to bound the time for identifying SCCs and determining the random boundary vertices $R = \{v \in V_R \cap C \mid \exists u \in S \setminus C \text{ s.t. } (v, u) \in E\}$) in Case 1, $|H| + |T| \geq \sqrt{m/\log n}$, and Case 2, $|H|+|T| < \sqrt{m/\log n}$. Since labels are added only when edges are not considered further and all labels of the considered vertices are deleted when Case 1 occurs, Case 1 can happen at most $\sqrt{m \log n}$ times. Thus the total time for Case 1 can be bounded by $O(m\sqrt{m \log n})$. In Case 2 we charge the time for the $O(\sqrt{m/\log n})$ lock-step searches to the edges in the identified SCC C. With Tarjan's SCC algorithm [37] a bottom SCC is identified in time proportional to the number of edges in the bottom SCC when the search is started at a vertex in the bottom SCC, which is in Algorithm StreettMDPsparse guaranteed by Lemma 5.29 for both top and bottom SCCs. Since always the smallest top or bottom SCC in $P[S]$ is identified, each edge is charged at most $O(\log n)$ times. Thus the total time for identifying SCCs in Case 2 is $O(m\sqrt{m \log n})$. Determining the random boundary vertices R in Case 2 can be charged to the edges in C and to the edges from C to $S \setminus C$, which are then not considered further by the algorithm. Thus the total runtime of the algorithm is $O(m\sqrt{m \log n})$. □

Proposition 5.31 (Correctness of Algorithm StreettMDPsparse)**.** *Let W be the set returned by Algorithm* StreettMDPsparse*. We have $W = \langle\langle 1 \rangle\rangle_{as}(P, \mathit{Streett}(\mathrm{SP}))$.*

Proof. Lemma 5.29 implies that whenever a vertex set is added to goodEC in Line 11, it induces a strongly connected sub-MDP. Thus we have that immediately before a set of vertices C is added to goodEC in Line 11 or Line 25, it is checked that $P[C]$ contains at least one edge, is strongly connected, and $\mathtt{Bad}(D(C))$ is empty. For the soundness and completeness of Algorithm StreettMDPsparse it remains to show the Invariants 5.21 and 5.23. We have for each iteration of the outer while-loop: The inner while-loop is the same as in Algorithms StreettMDPimpr and StreettMDPdense. In the case $|H| + |T| = 0$, the currently considered set of vertices is added to goodEC and no set is added to Q. If $|H| + |T| \geq \sqrt{m/\log n}$, the same operations as in Algorithm StreettMDPimpr are performed. If $|H| + |T| < \sqrt{m/\log n}$, like in Algorithm StreettMDPdense, either a top or a bottom SCC is identified and then the same operations as in Algorithm StreettMDPdense are applied to the identified SCC and the remaining vertices. As the operations in Algorithms StreettMDPimpr and StreettMDPdense preserve the invariants, this is also true for Algorithm StreettMDPsparse. □

6 MDPs with Rabin and Disjunctive Büchi and coBüchi Objectives

In the first part of this section we prove the following conditional lower bounds for Rabin, and disjunctive Büchi and coBüchi objectives.

Theorem 6.1. *Assuming STC, there is no combinatorial $O(n^{3-\epsilon})$ or $O((kn^2)^{1-\epsilon})$ algorithm for each of the following problems:*

1. *computing the a.s. winning set in an MDP with a disjunctive Büchi query;*
2. *computing the winning set in a graph with a disjunctive coBüchi objective and thus also computing the a.s. winning set in an MDP for disjunctive coBüchi objective or a disjunctive coBüchi query;*
3. *computing the a.s. winning set in an MDP with a Rabin objective.*

Theorem 6.2. *Assuming SETH or OVC, there is no $O(m^{2-\epsilon})$ or $O((k \cdot m)^{1-\epsilon})$ algorithm for each of the following problems:*

1. *computing the a.s. winning set in an MDP with a disjunctive Büchi query;*
2. *computing the a.s. winning set in an MDP with a disjunctive coBüchi objective or a disjunctive coBüchi query;*
3. *computing the a.s. winning set in an MDP with a disjunctive Singleton coBüchi objective or a disjunctive Singleton coBüchi query;*

4. *computing the a.s. winning set in an MDP with a Rabin objective.*

On the algorithmic side we prove the following theorem in the second part of this section. Note that a Rabin objective corresponds to a disjunctive objective over 1-pair Rabin objectives.

Theorem 6.3. *Given an MDP* $P = ((V, E), (V_1, V_R), \delta)$ *and a Rabin objective wit Rabin pairs* $\mathrm{RP} = \{(L_i, U_i) \mid 1 \leq i \leq k\}$*, let* $b = \sum_{i=1}^{k}(|L_i| + |U_i|)$*. Let* MEC *denote the time to compute a MEC-decomposition.*

1. *The almost-sure winning set* $\langle\!\langle 1 \rangle\!\rangle_{as}(P, \mathit{Rabin}(\mathrm{RP}))$ *can be computed in* $O(k \cdot \mathrm{MEC})$ *time.*
2. *If* $U_i = \emptyset$ *for all* $1 \leq i \leq k$ *(i.e. the Rabin pairs are Büchi objectives), then the almost-sure winning set for the disjunctive objective over the Rabin pairs can computed in* $O(\mathrm{MEC}+b)$ *time and the disjunctive query in* $O(k \cdot m + \mathrm{MEC})$ *time.*
3. *If* $L_i = V$ *for all* $1 \leq i \leq k$ *(i.e. the Rabin pairs are coBüchi objectives), then the almost-sure winning set for the disjunctive objective and the disjunctive query over the Rabin pairs can computed in* $O(k \cdot m + \mathrm{MEC})$ *time.*

6.1 Conditional Lower Bounds for Rabin, Büchi and coBüchi

The conditional lower bounds for Rabin, and disjunctive Büchi and coBüchi objectives are based on our results for reachability (see Section 3.2) and safety objects (see Section 4.1) and the Observations 2.5, 2.6 & 2.8 that interlink these objectives.

Proposition 6.4. *Assuming STC, there is no combinatorial* $O(n^{3-\epsilon})$ *or* $O((k \cdot n^2)^{1-\epsilon})$ *algorithm for*

1. *computing the winning set in an MDP with a disjunctive Büchi query,*
2. *computing the winning set in a graph with a disjunctive coBüchi objective, and*
3. *computing the winning set in an MDP with a Rabin objective.*

Moreover, there is no such algorithm deciding whether the winning set is non-empty or deciding whether a specific vertex is in the winning set.

Proof. 1) By Observation 2.6 in MDPs reachability can be reduced in linear time to Büchi. Thus the result follows from the corresponding hardness result for reachability (cf. Theorem 3.7).

2) By Observation 2.5 the winning set of disjunctive safety is non-empty iff the winning set of disjunctive coBüchi with the same target sets is non-empty. Thus the result follows from the corresponding hardness result for safety (cf. Theorem 4.2).
For the problem of deciding whether a specific vertex is in the winning set, recall that the graph G' constructed in Reduction 4.3 is such that vertex s appears in each infinite path and thus if there is a winning strategy starting in some vertex, then there is also one starting in s. That is, deciding on G' whether s is winning is equivalent to deciding whether the winning set is non-empty. Hence, the lower bound for the former follows.

3) The result follows from (2) and Observation 2.8, by which disjunctive coBüchi objectives are special instances of Rabin objectives. □

Proposition 6.5. *Assuming SETH or OVC, there is no $O(m^{2-\epsilon})$ or $O((k \cdot m)^{1-\epsilon})$ algorithm for*

1. *computing the winning set in an MDP with a disjunctive Büchi query,*
2. *computing the winning set in an MDP with a disjunctive coBüchi objective or a disjunctive coBüchi query,*
3. *computing the winning set in an MDP with a disjunctive Singleton coBüchi objective or a disjunctive Singleton coBüchi query, and*
4. *computing the winning set in an MDP with a Rabin objective.*

Moreover, there is no such algorithm for deciding whether the winning set is non-empty or deciding whether a specific vertex is in the winning set.

Proof. 1) By Observation 2.6 in MDPs reachability can be reduced in linear time to Büchi. Thus the result follows from the corresponding hardness result for reachability (cf. Theorem 3.10).

2) By Observation 2.5 the winning set of disjunctive safety is non-empty iff the winning set of disjunctive coBüchi with the same target sets is non-empty. Thus the result follows from the corresponding hardness result for safety (cf. Theorem 4.5).
For the problem of deciding whether a specific vertex is in the winning set, recall that the MDP P constructed in Reduction 4.6 is such that vertex s appears in each infinite path and thus if there is a winning strategy starting in some vertex, then there is also one starting in s. That is, deciding on P whether s is winning is equivalent to deciding whether the winning set is non-empty. Hence, the lower bound for the former follows.

3) This holds by (2) and the fact that all sets T_i in Lemma 4.7 are singletons.

3) The result follows from (2) and Observation 2.8, by which disjunctive coBüchi objectives are special instances of Rabin objectives. □

6.2 Algorithm for MDPs with Rabin Objectives

In this section we describe an algorithm for MDPs with Rabin objectives that considers each MEC of the input MDP separately. This formulation has the advantage that we can obtain a faster runtime than previously known for the special case of disjunctive coBüchi objectives, which we describe in Section 6.4. The special case of Büchi objectives is described in Section 6.3.

For Rabin objectives a good end-component could, equivalently to Definition 5.2, be defined as follows.

Definition 6.6 (Good Rabin End-Component)**.** *Given an MDP P and a set* $\mathrm{RP} = \{(L_i, U_i) \mid 1 \leq i \leq k\}$ *of Rabin pairs, a* good Rabin end-component *is an end-component X of P such that $L_i \cap X \neq \emptyset$ and $U_i \cap X = \emptyset$ for some $1 \leq i \leq k$.*

As for Streett objectives, we determine the almost-sure winning set for Rabin objectives by computing almost-sure reachability of the union of all good Rabin end-components. The correctness of this approach follows from Corollary 5.7 and Proposition 5.11. We use the

notation defined in Section 5.2. Our strategy to find all good Rabin end-components is as follows. First the MEC-decomposition of the input MDP P is determined. For each MEC X and separately for each $1 \leq i \leq k$ we first remove the set U_i and its random attractor and then compute the MEC-decomposition in the sub-MDP induced by the remaining vertices. Every newly computed MEC that contains a vertex of L_i is a good Rabin end-component. If the MEC X of P contains one such good end-component, then by Corollary 5.8 all vertices of X are in the almost-sure winning set for the Rabin objective. Thus we can immediately add X to the set of winning MECs in Line 8.[6]

Algorithm RabinMDP: Algorithm for MDPs with Rabin Objectives

Input : MDP $P = ((V, E), (V_1, V_R), \delta)$ and Rabin pairs $\mathrm{RP} = \{(L_i, U_i) \mid 1 \leq i \leq k\}$
Output: $\langle\!\langle 1 \rangle\!\rangle_{as}(P, \mathrm{Rabin}(\mathrm{RP}))$

```
1  X ← allMECs(P); winMEC ← ∅
2  foreach X ∈ X do
3      for 1 ≤ i ≤ k do
4          if L_i ∩ X ≠ ∅ then
5              Y ← allMECs(X \ Attr(P[X], U_i))
6              foreach Y ∈ Y do
7                  if L_i ∩ Y ≠ ∅ then
8                      winMEC ← winMEC ∪ {X}
9                      continue with next X ∈ X
10 return ⟨⟨1⟩⟩_as(P, Reach(⋃_{X∈winMEC} X))
```

Proposition 6.7 (Runtime of Algorithm RabinMDP)**.** *Algorithm* RabinMDP *can be implemented in* $O(k \cdot \mathrm{MEC})$ *time.*

Proof. The initialization of $\mathcal{X}$ with all MECs of the input MDP P can clearly be done in $O(\mathrm{MEC})$ time. Further by Theorem 3.1 the final almost-sure reachability computation can be done in $O(\mathrm{MEC})$ time[7]. Assume that each vertex has a list of the sets L_i and U_i for $1 \leq i \leq k$ it belongs to. (We can generate these lists from the lists of the Rabin pairs in $O(b) = O(nk)$ time at the beginning of the algorithm.) Consider an iteration of the outer for-each loop, let X denote the considered MEC, and fix one iteration i of the k iterations of the for loop. Line 4 requires $O(|X|)$ time. Let m_X be the number of edges in $P[X]$ and let MEC_X denote the time needed to compute a MEC-decomposition on $P[X]$. The Line 5

[6] We could alternatively add only the vertices in the good end-component because the winning MEC would be detected as winning in the final almost-sure reachability computation; the presented formulation shows the similarities to the coBüchi algorithm in Section 6.4. Additionally, this allows reusing the initial MEC-decomposition for the almost-sure reachability computation.

[7] Actually the almost-sure reachability computation can be done in $O(m)$ reusing the already computed MEC decomposition.

requires $O(m_X + \mathrm{MEC}_X) = O(\mathrm{MEC}_X)$ time. The inner for-each loop takes $O(|X|)$ time as in each iteration we need $O(|Y|)$ in Line 7 and constant time in Line 8. Thus in total we have $O(b + \mathrm{MEC} + \sum_{X \in \mathcal{X}} k \cdot (|X| + \mathrm{MEC}_X)) = O(k \cdot \mathrm{MEC})$. □

Proposition 6.8 (Correctness of Algorithm RabinMDP)**.** *Algorithm* RabinMDP *computes* $\langle\!\langle 1 \rangle\!\rangle_{as}(P, \mathit{Rabin}(\mathrm{RP}))$.

Proof. By the Corollaries 5.8 & 5.7 and Proposition 5.11 we know that it suffices to correctly classify each MEC as either *winning* or *not winning*; we say a MEC is *winning* iff it contains a good Rabin EC, that is, it contains an EC X such that $L_i \cap X \neq \emptyset$ and $U_i \cap X = \emptyset$ for some $1 \leq i \leq k$. The loops in Lines 2 & 3 iterate over all MECs X and all Rabin Pairs (L_i, U_i). What remains to show is that Lines 4–8 correctly classify whether a MEC contains a good EC satisfying the Rabin pair (L_i, U_i).

- Assume X contains a good EC X' that satisfies (L_i, U_i), i.e., $L_i \cap X' \neq \emptyset$ and $U_i \cap X' = \emptyset$. Then the if condition in Line 4 is true and the algorithm subtracts the random attractor of U_i. As X' is strongly connected, has no outgoing random edges, and $U_i \cap X' = \emptyset$, it does not intersect with $\mathit{Attr}(P[X], U_i)$ (see also Lemma 5.13). Thus there is a MEC $Y \in \mathcal{Y}$ that contains X' and thus $L_i \cap Y \neq \emptyset$. Hence, the algorithm correctly classifies the set X as winning MEC.
- Assume the algorithm classifies a MEC X as winning. Then for some i in Line 7 there is an end-component $Y \in \mathcal{Y}$ of $P[X \setminus \mathit{Attr}(P[X], U_i)]$ with $L_i \cap Y \neq \emptyset$ and $U_i \cap Y = \emptyset$, i.e., Y is a good end-component in $P[X \setminus \mathit{Attr}(P[X], U_i)]$. Moreover, there cannot be a random edge from $u \in Y$ to $\mathit{Attr}(P[X], U_i)$ as such an u would be included in the random attractor $\mathit{Attr}(P[X], U_i)$. Thus Y is also a good end-component of the full MDP P, i.e., it was classified correctly.

By the above we have that whenever the outer for-each loop terminates, the set winMEC consists of all winning MECs and then by Corollary 5.7 and Proposition 5.11 we can compute $\langle\!\langle 1 \rangle\!\rangle_{as}(P, \mathrm{Rabin}(\mathrm{RP}))$ by computing almost-sure reachability of the union of all winning MECs. □

6.3 Algorithms for MDPs with Büchi Objectives

As Büchi objectives can be encoded as Rabin pairs, Algorithm RabinMDP can also be used to compute the a.s. winning set for disjunctive Büchi objectives. However, Büchi objectives allow for some immediate simplifications that result in Algorithm DisjObjBüchiMDP. This simplifications are based on the observation that for Büchi all sets U_i are empty and therefore also the random attractors computed in Line 5 of Algorithm RabinMDP are empty. Hence, there is also no need to recompute the MECs and deciding whether a MEC is winning reduces to testing whether it intersects with one of the target sets.

Proposition 6.9 (Runtime of Algorithm DisjObjBüchiMDP)**.** *Algorithm* DisjObjBüchiMDP *can be implemented in* $O(\mathrm{MEC} + b)$ *time.*

Algorithm DisjObjBüchiMDP: Algorithm for MDPs with Disjunctive Büchi Objectives

Input : MDP $P = ((V, E), (V_1, V_R), \delta)$ and Büchi objectives T_i for $1 \leq i \leq k$
Output: $\langle\!\langle 1 \rangle\!\rangle_{as} \left(P, \bigvee_{1 \leq i \leq k} \text{Büchi}\,(T_i)\right)$

1 $\mathcal{X} \leftarrow \texttt{allMECs}(P)$; $\mathsf{winMEC} \leftarrow \emptyset$
2 **foreach** $X \in \mathcal{X}$ **do**
3 **if** $\bigcup_{1 \leq i \leq k} T_i \cap X \neq \emptyset$ **then**
4 $\mathsf{winMEC} \leftarrow \mathsf{winMEC} \cup \{X\}$
5 **return** $\langle\!\langle 1 \rangle\!\rangle_{as} (P, \mathit{Reach}\,(\bigcup_{X \in \mathsf{winMEC}} X))$

Proof. The initialization of $\mathcal{X}$ with all MECs of the input MDP P can clearly be done in $O(\text{MEC})$ time. Further by Theorem 3.1 the final almost-sure reachability computation can be done in $O(\text{MEC})$ time. Assume that each vertex has a flag indicating whether it is in one of the sets T_i or in none of them (We can generate these flags from lists of the sets T_i in $O(b)$ time at the beginning of the algorithm.). Consider an iteration of the for-each loop, let X denote the considered MEC and fix some iteration i of the for loop. One Iteration costs $O(|X|)$ as in each iteration we need $O(|X|)$ in Line 3 and constant time in Line 4. Thus in total the algorithm takes $O(\text{MEC} + n + b) = O(\text{MEC} + b)$ time. □

When it comes to disjunctive Büchi queries with k sets T_i, one basically solves k Büchi problems and then computes disjunctive almost-sure reachability queries of the winning sets of the Büchi problems. However, as the MEC-decomposition is independent of the sets T_i, is suffices to compute the MEC-decomposition once. This results in an $O(k \cdot m + \text{MEC} + b) = O(k \cdot m + \text{MEC})$ time algorithm (see Algorithm DisjQueryBüchiMDP).

Algorithm DisjQueryBüchiMDP: Algorithm for Disjunctive Büchi Queries on MDPs

Input : MDP $P = ((V, E), (V_1, V_R), \delta)$ and Büchi objectives T_i for $1 \leq i \leq k$
Output: $\bigvee_{1 \leq i \leq k} \langle\!\langle 1 \rangle\!\rangle_{as} (P, \text{Büchi}\,(T_i))$

1 $\mathcal{X} \leftarrow \texttt{allMECs}(P)$
2 **for** $1 \leq i \leq k$ **do**
3 $\mathsf{winMEC}_i \leftarrow \emptyset$
4 **foreach** $X \in \mathcal{X}$ **do**
5 **for** $1 \leq i \leq k$ **do**
6 **if** $T_i \cap X \neq \emptyset$ **then**
7 $\mathsf{winMEC}_i \leftarrow \mathsf{winMEC}_i \cup \{X\}$
8 **return** $\bigvee_{1 \leq i \leq k} \langle\!\langle 1 \rangle\!\rangle_{as} \left(P, \mathit{Reach}\left(\bigcup_{X \in \mathsf{winMEC}_i} X\right)\right)$

6.4 Algorithms for MDPs with coBüchi Objectives

Again, as coBüchi objectives can be encoded as Rabin pairs, one can use Algorithm RabinMDP to compute the a.s. winning set for disjunctive coBüchi objectives. However, coBüchi objectives allow for some simplifications that result in the simpler and more efficient Algorithm DisjObjCoBüchiMDP. This simplifications are based on the observation that for coBüchi all sets L_i coincide with the set of all vertices and therefore the if conditions in Lines 4 & 7 of Algorithm RabinMDP are always true. That is, whenever there is a vertex in a MEC X of P that is not contained in $Attr(P[X], T_i)$, then there is a MEC in $P[X \setminus Attr(P[X], T_i)]$, which is a good end-component of P. Testing whether a MEC contains a good EC for a coBüchi objective coBüchi (T_i) thus reduces to testing whether the random attractor of T_i covers the whole MEC.

Observation 6.10. *The same ideas can be used for the disjunction of one-pair Streett objectives (Table 5). For each MEC X and each i we check whether $X \cap L_i \neq \emptyset$ and $X \cap U_i = \emptyset$. If this is the case, then we determine whether the random attractor of L_i covers the whole MEC. If not, then the MEC contains a good end-component for the one-pair Streett objective.*

Algorithm DisjObjCoBüchiMDP: Algorithm for MDPs with Disjunctive coBüchi Objectives

---

Input : MDP $P = ((V, E), (V_1, V_R), \delta)$ and coBüchi objectives T_i for $1 \leq i \leq k$
Output: $\langle\!\langle 1 \rangle\!\rangle_{as} \left(P, \bigvee_{1 \leq i \leq k} \text{coBüchi}\,(T_i)\right)$

1 $\mathcal{X} \leftarrow \texttt{allMECs}(P)$; $\mathsf{winMEC} \leftarrow \emptyset$
2 **foreach** $X \in \mathcal{X}$ **do**
3 **for** $1 \leq i \leq k$ **do**
4 **if** $X \not\subseteq Attr(P[X], T_i)$ **then**
5 $\mathsf{winMEC} \leftarrow \mathsf{winMEC} \cup \{X\}$
6 continue with next $X \in \mathcal{X}$
7 **return** $\langle\!\langle 1 \rangle\!\rangle_{as} (P, Reach(\bigcup_{X \in \mathsf{winMEC}} X))$

Proposition 6.11 (Runtime)**.** *Algorithm* DisjObjCoBüchiMDP *can be implemented in $O(k \cdot m + \text{MEC})$ time.*

Proof. The initialization of $\mathcal{X}$ with all MECs of the input MDP P can clearly be done in $O(\text{MEC})$ time. Further by Theorem 3.1 the final almost-sure reachability computation can be done in $O(\text{MEC})$ time. Consider an iteration of the for-each loop, let X denote the considered MEC, and fix some iteration i of the for loop. Let m_X be the number of edges in $P[X]$. In the ith iteration we need $O(|m_X|)$ time to compute the random attractor in Line 4 and constant time in Line 5. Thus the total time is $O(k \cdot m + \text{MEC})$. □

When it comes to disjunctive coBüchi queries with k sets T_i, we have to remember which of the sets T_i are satisfied by a MEC and then compute disjunctive almost-sure reachability queries, one query per set T_i. This increases the running time for the almost-sure reachability computation to $O(k \cdot m)$ (given the MEC-decomposition), which, however, is subsumed by the total running time of $O(k \cdot m + \text{MEC})$. The resulting algorithm is stated as Algorithm DisjQueryCoBüchiMDP.

Algorithm DisjQueryCoBüchiMDP: Algorithm for Disjunctive coBüchi Queries on MDPs

Input : MDP $P = ((V, E), (V_1, V_R), \delta)$ and coBüchi objectives T_i for $1 \leq i \leq k$
Output: $\bigvee_{1 \leq i \leq k} \langle\!\langle 1 \rangle\!\rangle_{as} (P, \text{coBüchi}(T_i))$

```
1 X ← allMECs(P)
2 for 1 ≤ i ≤ k do
3     winMEC_i ← ∅
4 foreach X ∈ X do
5     for 1 ≤ i ≤ k do
6         if X ⊈ Attr(P[X], T_i) then
7             winMEC_i ← winMEC_i ∪ {X}
8 return ⋁_{1≤i≤k} ⟨⟨1⟩⟩_as ( P, Reach ( ⋃_{X∈winMEC_i} X ) )
```

7 Algorithm for Graphs with Singleton coBüchi Objectives

In this section we show how to compute in linear time the winning set for graphs with a special type of coBüchi objectives, namely when all sets T_i for $1 \leq i \leq k$ have cardinality one.

Theorem 7.1. *Given a graph $G = (V, E)$ and coBüchi objectives T_i with $|T_i| = 1$ for $1 \leq i \leq k$, the winning set for the disjunction over the coBüchi objectives can be computed in $O(m)$ time.*

To compute the winning set it is sufficient to detect whether a strongly connected graph contains a cycle that does *not contain all* the vertices in the set $T = \bigcup_{1 \leq i \leq k} T_i$. To see this, first note that each non-trivial SCC of the graph (i.e., each SCC that contains at least one edge) that does not contain all vertices of T is winning. If there is no SCC S with $T \subseteq S$, then we can determine the winning set in linear time by computing the vertices that can reach any non-trivial SCC. Thus it remains to consider an SCC S with $T \subseteq S$. For the relevant case of $|T| > 1$ we have that S is a non-trivial SCC. Since S is strongly connected, the vertices of S can reach each other and hence it is sufficient to compute whether S contains a cycle that does not contain all the vertices of T (i.e. solving the *non-emptiness* problem). If such a cycle exists, then also S is winning, otherwise S is not winning. In any case, the winning set can then be determined by computing the vertices that can reach some winning SCC.

Algorithm SINGLETONCBGRAPH: Disjunctive Singleton coBüchi on Graphs

Input : strongly connected graph $G = (V, E)$ and coBüchi objectives T_i with $|T_i| = 1$ for $1 \leq i \leq k$ and $k > 1$, let $T = \bigcup_i T_i$

Output : "yes" if there is a cycle C with $T \not\subseteq C$; "no" otherwise

```
1  S ← SCCs(G[V \ T_1])
2  if S contains non-trivial SCC then
3      return yes
4  else
5      let s be the vertex in T_1
6      replace s with s_in and s_out: s_in gets in-edges and s_out gets out-edges of s
7      Q_0 ← {s_out}; mark s_out
8      for j ← 0 to k − 1 do
9          Q_{j+1} ← ∅
10         while Q_j ≠ ∅ do
11             remove v from Q_j
12             if v = s_in then
13                 return yes
14             foreach (v, w) ∈ E with w not marked do
15                 mark w
16                 if w ∈ T then add w to Q_{j+1}
17                 else add w to Q_j
18     return no
```

We now describe the algorithm to determine whether a strongly connected graph $G = (V, E)$ contains a simple cycle C such that we have $T_i \cap C = \emptyset$ for some $1 \leq i \leq k$, given $|T_i| = 1$ for all i. First we check whether $G[V \setminus T_1]$ contains a non-trivial SCC. If this is true, then G contains a cycle that does not contain T_1 and we are done. Otherwise every cycle of G contains T_1. We assign the edges of G edge lengths as follows: All edges $(v, w) \in E$ for which $w \in T$ have length 1, all other edges have length 0. Let s denote the vertex in T_1. Let δ be the length of the shortest path (w.r.t. the edge lengths defined above) from s to s that uses at least one edge, i.e., the minimum length of a cycle containing s. We have that $\delta < k$ if and only if this cycle with the length δ does not contain all vertices of T. Thus if $\delta < k$, then G is winning for the coBüchi objective, otherwise not. Note that this algorithm would also work for a Rabin objective where we have for each $1 \leq i \leq k$ that (a) $L_i = \{s\}$ for some $s \in V$ and (b) $|U_i| = 1$.

Since all edge lengths are zero or one, we can compute δ in linear time. In Algorithm SingletonCBGraph we additionally use that all incoming edges of a vertex have the same length. After checking whether $G[V \setminus T_1]$ contains a non-trivial SCC, the algorithm works as follows. We modify the graph by replacing the vertex s by two vertices, s_{in} and s_{out}, and replacing s in all edges $(v, s) \in E$ with s_{in} and in all edges $(s, v) \in E$ with s_{out}. Then δ is equal to the shortest path from s_{out} to s_{in}. For the algorithm we consider both s_{in} and s_{out} to be contained in T. In the jth iteration of the for-loop we consider two "queues", Q_j and Q_{j+1} (can be implemented as sets). Each vertex is added to a queue at most once during the algorithm, which is ensured by marking vertices when they are added to a queue and only add before unmarked vertices. The following lemma shows that, until the vertex s_{in} is removed from Q_j and the algorithm terminates, precisely the vertices with distance j from s_{out} are added to Q_j for each j. Thus s_{in} is added to Q_j for some $j < k$ if and only if $\delta < k$, which shows the correctness of the algorithm. The runtime of the algorithm is $O(m)$ because each vertex is added to and removed from a queue at most once and thus the outgoing edges of a vertex are only considered once, namely when it is removed from a queue.

Lemma 7.2. *Before each iteration j of the for-loop in Algorithm* SingletonCBGraph*, Q_j contains the vertices of T with distance j from s_{out}. During iteration j, the vertices of $V \setminus T$ with distance j from s_{out} are added to Q_j. No other vertices are added to Q_j.*

Proof. The proof is by induction over the iterations of the for-loop. Before the first iteration ($j = 0$), Q_0 is initialized with s_{out} and all queues Q_j for $j > 0$ are empty, thus the induction base holds. Assume the claim holds before the jth iteration. At the end of the while-loop, Q_j is empty; every vertex v that was added to Q_j before or in the jth iteration of the for-loop is removed from Q_j in some iteration of the while-loop. Then all the unmarked vertices w with $(v, w) \in E$ are marked and added to Q_j if the edge (v, w) has length zero or added to Q_{j+1} if the edge (v, w) has length one. A vertex $u \in V \setminus T$ with distance at least j from s_{out} has distance exactly j if and only if it can be reached from some vertex $v \in T$ that has distance j by a sequence of zero length edges. The while-loop precisely adds these vertices to Q_j. Further, a vertex $u \in V \cap T$ has distance $j + 1$ if and only if it has an edge from some vertex $v \in V$ that has distance j. The while-loop adds exactly these vertices to Q_{j+1}. □

8 Conclusion

In this work we present improved algorithms and the first conditional super-linear lower bounds for several fundamental model-checking problems in graphs and MDPs w.r.t. to ω-regular objectives. Our results establish the first model separation results for graphs and MDPs w.r.t. to classical ω-regular objectives, and the first objective separation results both in graphs and MDPs for dual objectives, and the conjunction and disjunction of objectives of the same type. An interesting direction of future work is to consider similar results for other models, such as, games on graphs.

References

[1] Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. If the current clique algorithms are optimal, so is Valiant's parser. In *FOCS*, pages 98–117, 2015.

[2] Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight Hardness Results For LCS and other Sequence Similarity Measures. In *FOCS*, pages 59–78, 2015.

[3] Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *FOCS*, pages 434–443, 2014.

[4] Amir Abboud, Virginia Vassilevska Williams, and Oren Weimann. Consequences of faster alignment of sequences. In *ICALP 2014, Proceedings, Part I*, pages 39–51, 2014.

[5] Amir Abboud, Virginia Vassilevska Williams, and Huacheng Yu. Matching triangles and basing hardness on an extremely popular conjecture. In *STOC*, pages 41–50, 2015.

[6] Amir Abboud, Virginia Vassilevska Williams, and Joshua R. Wang. Approximation and fixed parameter subquadratic algorithms for radius and diameter in sparse graphs. In *SODA*, pages 377–391, 2016.

[7] Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). In *STOC*, pages 51–58, 2015.

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

[9] Catriel Beeri. On the membership problem for functional and multivalued dependencies in relational databases. *ACM Transactions on Database Systems*, pages 241–259, 1980.

[10] Karl Bringmann. Why walking the dog takes time: Frechet distance has no strongly subquadratic algorithms unless SETH fails. In *FOCS*, pages 661–670, 2014.

[11] Karl Bringmann and Marvin Künnemann. Quadratic Conditional Lower Bounds for String Problems and Dynamic Time Warping. In *FOCS*, pages 79–97, 2015.

[12] Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. The complexity of satisfiability of small depth circuits. In *IWPEC*, pages 75–85, 2009.

[13] Krishnendu Chatterjee, Luca de Alfaro, and Rupak Majumdar. The complexity of coverage. *Int. J. Found. Comput. Sci.*, 24(2):165–186, 2013.

[14] Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Qualitative analysis of partially-observable Markov decision processes. In *MFCS*, pages 258–269, 2010.

[15] Krishnendu Chatterjee and Monika Henzinger. Faster and Dynamic Algorithms For Maximal End-Component Decomposition And Related Graph Problems In Probabilistic Verification. In *SODA*, pages 1318–1336, 2011.

[16] Krishnendu Chatterjee and Monika Henzinger. Efficient and Dynamic Algorithms for Alternating Büchi Games and Maximal End-component Decomposition. *Journal of the ACM*, 61(3):15, 2014.

[17] Krishnendu Chatterjee, Monika Henzinger, and Veronika Loitzenbauer. Improved Algorithms for One-Pair and k-Pair Streett Objectives. In *LICS*, pages 269–280, 2015.

[18] Krishnendu Chatterjee, Thomas A. Henzinger, and Nir Piterman. Generalized parity games. In *FOSSACS*, volume 4423, pages 153–167, 2007.

[19] Krishnendu Chatterjee, Marcin Jurdziński, and Thomas A. Henzinger. Simple stochastic parity games. In *CSL*, pages 100–113, 2003.

[20] A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri. Nusmv: a new symbolic model checker. *International Journal on Software Tools for Technology Transfer*, 2:410–425, 2000.

[21] Costas Courcoubetis and Mihalis Yannakakis. The complexity of probabilistic verification. *J. ACM*, 42(4):857–907, July 1995.

[22] E. Allen Emerson and Charanjit S. Jutla. The complexity of tree automata and logics of programs. *SIAM J. Comput.*, 29(1):132–158, 1999.

[23] Nathanaël Fijalkow and Florian Horn. The surprizing complexity of reachability games. *CoRR*, abs/1010.2420, 2010.

[24] Anka Gajentaan and Mark H. Overmars. On a class of $O(n^2)$ problems in computational geometry. *Comput. Geom.*, 45(4):140–152, 2012.

[25] Monika Henzinger, Valerie King, and Tandy Warnow. Constructing a Tree from Homeomorphic Subtrees, with Applications to Computational Evolutionary Biology. *Algorithmica*, 24(1):1–13, 1999.

[26] Monika Henzinger, Sebastian Krinninger, and Veronika Loitzenbauer. Finding 2-Edge and 2-Vertex Strongly Connected Components in Quadratic Time. In *ICALP (Track A)*, pages 713–724, 2015.

[27] Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *STOC*, pages 21–30, 2015.

[28] Monika Henzinger and Jan Arne Telle. Faster Algorithms for the Nonemptiness of Streett Automata and for Communication Protocol Pruning. In *SWAT*, pages 16–27, 1996.

[29] Gerard J. Holzmann. The model checker spin. *IEEE Trans. Softw. Eng.*, 23(5):279–295, May 1997.

[30] Neil Immerman. Number of quantifiers is better than number of tape cells. *Journal of Computer and System Sciences*, pages 384–406, 1981.

[31] Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001.

[32] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. Prism 4.0: Verification of probabilistic real-time systems. In *CAV*, LNCS 6806, pages 585–591, 2011.

[33] François Le Gall. Powers of Tensors and Fast Matrix Multiplication. In *ISSAC*, pages 296–303, 2014.

[34] Lillian Lee. Fast context-free grammar parsing requires fast boolean matrix multiplication. *J. ACM*, 49(1):1–15, January 2002.

[35] Mihai Patrascu and Ryan Williams. On the possibility of faster SAT algorithms. In *SODA*, pages 1065–1075, 2010.

[36] Liam Roditty and Uri Zwick. On dynamic shortest paths problems. *Algorithmica*, 61(2):389–401, 2011. Announced at ESA'04.

[37] Robert Endre Tarjan. Depth first search and linear graph algorithms. *SIAM J. Computing*, 1(2):146–160, 1972.

[38] W. Thomas. On the synthesis of strategies in infinite games. In *STACS'95*, LNCS 900, pages 1–13. Springer, 1995.

[39] W. Thomas. Languages, automata, and logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, Beyond Words, chapter 7, pages 389–455. Springer, 1997.

[40] Virginia Vassilevska Williams and Ryan Williams. Subcubic equivalences between path, matrix and triangle problems. In *FOCS 2010*, pages 645–654, 2010.

[41] Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci.*, 348(2-3):357–365, 2005. Announced at ICALP'04.

[42] Ryan Williams. Faster all-pairs shortest paths via circuit complexity. In *STOC 2014*, pages 664–673, 2014.

[43] Ryan Williams. Faster decision of first-order graph properties. In *CSL-LICS '14*, pages 80:1–80:6, 2014.

[44] Pierre Wolper. Constructing automata from temporal logic formulas: A tutorial. In *Lectures on Formal Methods and Performance Analysis*, pages 261–277, 2000.