Symbolic Algorithms for Graphs and Markov Decision Processes with Fairness Objectives

Krishnendu Chatterjee[1(✉)], Monika Henzinger[2], Veronika Loitzenbauer[3], Simin Oraee[4], and Viktor Toman[1]

[1] IST Austria, Klosterneuburg, Austria
krish.chat@gmail.com
[2] University of Vienna, Vienna, Austria
[3] Johannes Kepler University Linz, Linz, Austria
[4] Max Planck Institute for Software Systems, Kaiserslautern, Germany

Abstract. Given a model and a specification, the fundamental model-checking problem asks for algorithmic verification of whether the model satisfies the specification. We consider graphs and Markov decision processes (MDPs), which are fundamental models for reactive systems. One of the very basic specifications that arise in verification of reactive systems is the strong fairness (aka Streett) objective. Given different types of requests and corresponding grants, the objective requires that for each type, if the request event happens infinitely often, then the corresponding grant event must also happen infinitely often. All ω-regular objectives can be expressed as Streett objectives and hence they are canonical in verification. To handle the state-space explosion, symbolic algorithms are required that operate on a succinct implicit representation of the system rather than explicitly accessing the system. While explicit algorithms for graphs and MDPs with Streett objectives have been widely studied, there has been no improvement of the basic symbolic algorithms. The worst-case numbers of symbolic steps required for the basic symbolic algorithms are as follows: quadratic for graphs and cubic for MDPs. In this work we present the first sub-quadratic symbolic algorithm for graphs with Streett objectives, and our algorithm is sub-quadratic even for MDPs. Based on our algorithmic insights we present an implementation of the new symbolic approach and show that it improves the existing approach on several academic benchmark examples.

1 Introduction

In this work we present faster symbolic algorithms for graphs and Markov decision processes (MDPs) with strong fairness objectives. For the fundamental model-checking problem, the input consists of a *model* and *a specification*, and the algorithmic verification problem is to check whether the model *satisfies* the specification. We first describe the specific model-checking problem we consider and then our contributions.

H. Chockler and G. Weissenbacher (Eds.): CAV 2018, LNCS 10982, pp. 178–197, 2018.
https://doi.org/10.1007/978-3-319-96142-2_13

Models: Graphs and MDPs. Two standard models for reactive systems are graphs and Markov decision processes (MDPs). Vertices of a graph represent states of a reactive system, edges represent transitions of the system, and infinite paths of the graph represent non-terminating trajectories of the reactive system. MDPs extend graphs with probabilistic transitions that represent reactive systems with uncertainty. Thus graphs and MDPs are the de-facto model of reactive systems with nondeterminism, and nondeterminism with stochastic aspects, respectively [3,19].

Specification: Strong Fairness (aka Streett) Objectives. A basic and fundamental property in the analysis of reactive systems is the *strong fairness condition*, which informally requires that if events are enabled infinitely often, then they must be executed infinitely often. More precisely, the strong fairness conditions (aka Streett objectives) consist of k types of requests and corresponding grants, and the objective requires that for each type if the request happens infinitely often, then the corresponding grant must also happen infinitely often. After safety, reachability, and liveness, the strong fairness condition is one of the most standard properties that arise in the analysis of reactive systems, and chapters of standard textbooks in verification are devoted to it (e.g., [19, Chap. 3.3], [32, Chap. 3], [2, Chaps. 8, 10]). Moreover, all ω-regular objectives can be described by Streett objectives, e.g., LTL formulas and non-deterministic ω-automata can be translated to deterministic Streett automata [34] and efficient translation has been an active research area [16,23,28]. Thus Streett objectives are a canonical class of objectives that arise in verification.

Satisfaction. The basic notions of satisfaction for graphs and MDPs are as follows: For graphs the notion of satisfaction requires that there is a trajectory (infinite path) that belongs to the set of paths described by the Streett objective. For MDPs the satisfaction requires that there is a policy to resolve the nondeterminism such that the Streett objective is ensured almost-surely (with probability 1). Thus the algorithmic model-checking problem of graphs and MDPs with Streett objectives is a core problem in verification.

Explicit vs Symbolic Algorithms. The traditional algorithmic studies consider *explicit* algorithms that operate on the explicit representation of the system. In contrast, *implicit* or *symbolic* algorithms only use a set of predefined operations and do not explicitly access the system [20]. The significance of symbolic algorithms in verification is as follows: to combat the state-space explosion, large systems must be succinctly represented implicitly and then symbolic algorithms are scalable, whereas explicit algorithms do not scale as it is computationally too expensive to even explicitly construct the system.

Relevance. In this work we study symbolic algorithms for graphs and MDPs with Streett objectives. Symbolic algorithms for the analysis of graphs and MDPs are at the heart of many state-of-the-art tools such as SPIN, NuSMV for graphs [18,27] and PRISM, LiQuor, Storm for MDPs [17,22,29]. Our contributions are related to the algorithmic complexity of graphs and MDPs with Streett objectives for symbolic algorithms. We first present previous results and then our contributions.

Previous Results. The most basic algorithm for the problem for graphs is based on repeated SCC (strongly connected component) computation, and informally can be described as follows: for a given SCC, (a) if for every request type that is present in the SCC the corresponding grant type is also present in the SCC, then the SCC is identified as "good", (b) else vertices of each request type that has no corresponding grant type in the SCC are removed, and the algorithm recursively proceeds on the remaining graph. Finally, reachability to good SCCs is computed. The current best-known symbolic algorithm for SCC computation requires $O(n)$ symbolic steps, for graphs with n vertices [25], and moreover, the algorithm is optimal [15]. For MDPs, the SCC computation has to be replaced by MEC (maximal end-component) computation, and the current best-known symbolic algorithm for MEC computation requires $O(n^2)$ symbolic steps. While there have been several explicit algorithms for graphs with Streett objectives [12, 26], MEC computation [8–10], and MDPs with Streett objectives [7], as well as symbolic algorithms for MDPs with Büchi objectives [11], the current best-known bounds for symbolic algorithms with Streett objectives are obtained from the basic algorithms, which are $O(n \cdot \min(n, k))$ for graphs and $O(n^2 \cdot \min(n, k))$ for MDPs, where k is the number of types of request-grant pairs.

Our Contributions. In this work our main contributions are as follows:

- We present a symbolic algorithm that requires $O(n \cdot \sqrt{m \log n})$ symbolic steps, both for graphs and MDPs, where m is the number of edges. In the case $k = O(n)$, the previous worst-case bounds are quadratic ($O(n^2)$) for graphs and cubic ($O(n^3)$) for MDPs. In contrast, we present the first sub-quadratic symbolic algorithm both for graphs as well as MDPs. Moreover, in practice, since most graphs are sparse (with $m = O(n)$), the worst-case bounds of our symbolic algorithm in these cases are $O(n \cdot \sqrt{n \log n})$. Another interesting contribution of our work is that we also present an $O(n \cdot \sqrt{m})$ symbolic steps algorithm for MEC decomposition, which is relevant for our results as well as of independent interest, as MEC decomposition is used in many other algorithmic problems related to MDPs. Our results are summarized in Table 1.
- While our main contribution is theoretical, based on the algorithmic insights we also present a new symbolic algorithm implementation for graphs and MDPs with Streett objectives. We show that the new algorithm improves (by around 30%) the basic algorithm on several academic benchmark examples from the VLTS benchmark suite [21].

Technical Contributions. The two key technical contributions of our work are as follows:

- *Symbolic Lock Step Search:* We search for newly emerged SCCs by a local graph exploration around vertices that lost adjacent edges. In order to find small new SCCs first, all searches are conducted "in parallel", i.e., in lock-step, and the searches stop as soon as the first one finishes successfully. This approach has successfully been used to improve explicit algorithms [7,9,14,26]. Our contribution is a non-trivial symbolic variant (Sect. 3) which lies at the core of the theoretical improvements.

Table 1. Symbolic algorithms for Streett objectives and MEC decomposition.

Problem	Symbolic operations		
	Basic algorithm	Improved algorithm	Reference
Graphs with Streett	$O(n \cdot \min(n,k))$	$\mathbf{O(n\sqrt{m}\log n)}$	Theorem 2
MDPs with Streett	$O(n^2 \cdot \min(n,k))$	$\mathbf{O(n\sqrt{m}\log n)}$	Theorem 4
MEC decomposition	$O(n^2)$	$\mathbf{O(n\sqrt{m})}$	Theorem 3

– *Symbolic Interleaved MEC Computation:* For MDPs the identification of vertices that have to be removed can be interleaved with the computation of MECs such that in each iteration the computation of SCCs instead of MECs is sufficient to make progress [7]. We present a symbolic variant of this interleaved computation. This interleaved MEC computation is the basis for applying the lock-step search to MDPs.

2 Definitions

2.1 Basic Problem Definitions

Markov Decision Processes (MDPs) and Graphs. An MDP $P = ((V,E),(V_1,V_R),\delta)$ consists of a finite directed graph $G = (V,E)$ with a set of n vertices V and a set of m edges E, 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 $u \in V_1$ a *player 1 edge* and an edge (v,w) with $v \in V_R$ a *random edge*. For $v \in V$ we define $In(v) = \{w \in V \mid (w,v) \in E\}$ and $Out(v) = \{w \in V \mid (v,w) \in E\}$. 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 $\delta(v)[w] > 0$. Graphs are a special case of MDPs with $V_R = \emptyset$.

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 $(v, \sigma(\omega)) \in E$; 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. An *objective* ϕ is a subset of Ω said to be winning for player 1. We say that a play $\omega \in \Omega$ *satisfies the objective* if $\omega \in \phi$. For a vertex set $T \subseteq V$ the *reachability objective* is the set of infinite paths that contain a vertex of T, i.e., $\text{Reach}(T) = \{\langle v_0, v_1, v_2, \ldots\rangle \in \Omega \mid \exists j \geq 0 : v_j \in T\}$. Let $\text{Inf}(\omega)$ for $\omega \in \Omega$ denote the set of vertices that occur infinitely often in ω. Given a set TP 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., $\text{Streett(TP)} = \{\omega \in \Omega \mid L_i \cap \text{Inf}(\omega) = \emptyset \text{ or } U_i \cap \text{Inf}(\omega) \neq \emptyset \text{ for all } 1 \leq i \leq k\}$.

Almost-Sure Winning Sets. 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 ϕ if $\Pr_v^\sigma(\phi) = 1$. The *almost-sure winning set* $\langle\langle 1 \rangle\rangle_{as}(P, \phi)$ of player 1 is the set of vertices for which player 1 has an almost-sure winning strategy. In graphs the existence of an almost-sure winning strategy corresponds to the existence of a play in the objective, and the set of vertices for which player 1 has an (almost-sure) winning strategy is called the *winning set* $\langle\langle 1 \rangle\rangle(P, \phi)$ of player 1.

Symbolic Encoding of MDPs. Symbolic algorithms operate on sets of vertices, which are usually described by Binary Decision Diagrams (BDDs) [1,30]. In particular Ordered Binary Decision Diagrams [6] (OBDDs) provide a canonical symbolic representation of Boolean functions. For the computation of almost-sure winning sets of MDPs it is sufficient to encode MDPs with OBDDs and one additional bit that denotes whether a vertex is in V_1 or V_R.

Symbolic Steps. One symbolic step corresponds to one primitive operation as supported by standard symbolic packages like CUDD [35]. In this paper we only allow the same basic *set-based symbolic operations* as in [5,11,24,33], namely set operations and the following one-step symbolic operations for a set of vertices Z: (a) the one-step predecessor operator $\mathsf{Pre}(Z) = \{v \in V \mid Out(v) \cap Z \neq \emptyset\}$; (b) the one-step successor operator $\mathsf{Post}(Z) = \{v \in V \mid In(v) \cap Z \neq \emptyset\}$; and (c) the one-step *controllable* predecessor operator $\mathsf{CPre}_R(Z) = \{v \in V_1 \mid Out(v) \subseteq Z\} \cup \{v \in V_R \mid Out(v) \cap Z \neq \emptyset\}$; i.e., the CPre_R operator computes all vertices such that the successor belongs to Z with positive probability. This operator can be defined using the Pre operator and basic set operations as follows: $\mathsf{CPre}_R(Z) = \mathsf{Pre}(Z) \setminus (V_1 \cap \mathsf{Pre}(V \setminus Z))$. We additionally allow cardinality computation and picking an arbitrary vertex from a set as in [11].

Symbolic Model. Informally, a symbolic algorithm does not operate on explicit representation of the transition function of a graph, but instead accesses it through Pre and Post operations. For explicit algorithms, a $\mathsf{Pre}/\mathsf{Post}$ operation on a set of vertices (resp., a single vertex) requires $O(m)$ (resp., the order of indegree/outdegree of the vertex) time. In contrast, for symbolic algorithms $\mathsf{Pre}/\mathsf{Post}$ operations are considered unit-cost. Thus an interesting algorithmic question is whether better algorithmic bounds can be obtained considering $\mathsf{Pre}/\mathsf{Post}$ as unit operations. Moreover, the basic set operations are computationally less expensive (as they encode the relationship between the state variables) compared to the $\mathsf{Pre}/\mathsf{Post}$ symbolic operations (as they encode the transitions and thus the relationship between the present and the next-state variables). In all presented algorithms, the number of set operations is asymptotically at most the number of $\mathsf{Pre}/\mathsf{Post}$ operations. Hence in the sequel we focus on the number of $\mathsf{Pre}/\mathsf{Post}$ operations of algorithms.

Algorithmic Problem. Given an MDP P (resp. a graph G) and a set of Streett pairs TP, the problem we consider asks for a symbolic algorithm to

compute the almost-sure winning set $\langle\langle 1\rangle\rangle_{as}(P, \text{Streett(TP)})$ (resp. the winning set $\langle\langle 1\rangle\rangle(G, \text{Streett(TP)})$), which is also called the *qualitative analysis* of MDPs (resp. graphs).

2.2 Basic Concepts Related to Algorithmic Solution

Reachability. For a graph $G = (V, E)$ and a set of vertices $S \subseteq V$ the set $\textsc{GraphReach}(G, S)$ is the set of vertices of V that *can reach* a vertex of S within G, and it can be identified with at most $|\textsc{GraphReach}(G, S) \backslash S| + 1$ many Pre operations.

Strongly Connected Components. For a set of vertices $S \subseteq V$ we denote by $G[S] = (S, E \cap (S \times S))$ the subgraph of the graph G induced by the vertices of S. An induced subgraph $G[S]$ is strongly connected if there exists a path in $G[S]$ between every pair of vertices of S. A *strongly connected component* (*SCC*) of G is a set of vertices $C \subseteq V$ such that the induced subgraph $G[C]$ is strongly connected and C is a maximal set in V with this property. We call an SCC *trivial* if it only contains a single vertex and no edges; and *non-trivial* otherwise. The SCCs of G partition its vertices and can be found in $O(n)$ symbolic steps [25]. A bottom SCC C in a directed graph G is an SCC with no edges from vertices of C to vertices of $V \backslash C$, i.e., an SCC without *outgoing* edges. Analogously, a top SCC C is an SCC with no *incoming* edges from $V \backslash C$. For more intuition for bottom and top SCCs, consider the graph in which each SCC is contracted into a single vertex (ignoring edges within an SCC). In the resulting directed acyclic graph the sinks represent the bottom SCCs and the sources represent the top SCCs. Note that every graph has at least one bottom and at least one top SCC. If the graph is not strongly connected, then there exist at least one top and at least one bottom SCC that are disjoint and thus one of them contains at most half of the vertices of G.

Random Attractors. In an MDP P the *random attractor* $Attr_R(P, W)$ of a set of vertices W is defined as $Attr_R(P, W) = \bigcup_{j \geq 0} Z_j$ where $Z_0 = W$ and $Z_{j+1} = Z_j \cup \mathsf{CPre}_R(Z_j)$ for all $j > 0$. The attractor can be computed with at most $|Attr_R(P, W) \backslash W| + 1$ many CPre_R operations.

Maximal End-Components. Let X be a vertex set without outgoing random edges, i.e., with $Out(v) \subseteq X$ for all $v \in X \cap V_R$. A sub-MDP of an MDP P induced by a vertex set $X \subseteq V$ without outgoing random edges is defined as $P[X] = ((X, E \cap (X \times X), (V_1 \cap X, V_R \cap X), \delta)$. Note that the requirement that X has no outgoing random edges is necessary in order to use the same probabilistic transition function δ. An *end-component* (EC) of an MDP P is a set of vertices $X \subseteq V$ such that (a) X has no outgoing random edges, i.e., $P[X]$ is a valid sub-MDP, (b) the induced sub-MDP $P[X]$ is strongly connected, and (c) $P[X]$ contains at least one edge. Intuitively, an end-component is a set of vertices for which player 1 can ensure that the play stays within the set and almost-surely reaches all the vertices in the set (infinitely often). 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.

Good End-Components. All algorithms for MDPs with Streett objectives are based on finding good end-components, defined below. Given the union of all good end-components, the almost-sure winning set is obtained by computing the almost-sure winning set for the reachability objective with the union of all good end-components as the target set. The correctness of this approach is shown in [7,31] (see also [3, Chap. 10.6.3]). For Streett objectives a good end-component is defined as follows. In the special case of graphs they are called good components.

Definition 1 (Good end-component). *Given an MDP P and a set* $\mathrm{TP} = \{(L_j, U_j) \mid 1 \leq j \leq k\}$ *of target pairs, a* good end-component *is an end-component X of P such that for each $1 \leq j \leq k$ either $L_j \cap X = \emptyset$ or $U_j \cap X \neq \emptyset$. A maximal good end-component is a good end-component that is maximal with respect to set inclusion.*

Lemma 1 (Correctness of Computing Good End-Components [31, Corollary 2.6.5, Proposition 2.6.9]). *For an MDP P and a set* TP *of target pairs, let $\mathcal{X}$ be the set of all maximal good end-components. Then $\langle\!\langle 1 \rangle\!\rangle_{as}\left(P, \mathit{Reach}(\bigcup_{X \in \mathcal{X}} X)\right)$ is equal to $\langle\!\langle 1 \rangle\!\rangle_{as}(P, \mathit{Streett}(\mathrm{TP}))$.*

Iterative Vertex Removal. 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 2 (Correctness of Vertex Removal [31, Lemma 2.6.10]). *Given an MDP $P = ((V, E), (V_1, V_R), \delta)$, let X be an end-component with $X \subseteq S$ for some $S \subseteq V$. Then*

(a) $X \subseteq C$ for one SCC C of $P[S]$ and
(b) $X \subseteq S \setminus \mathit{Attr}_R(P', W)$ for each $W \subseteq V \setminus X$ and each sub-MDP P' containing X.

Let X be a good end-component. Then X is an end-component and for each index j, $X \cap U_j = \emptyset$ implies $X \cap L_j = \emptyset$. Hence we obtain the following corollary.

Corollary 1 ([31, Corollary 4.2.2]). *Given an MDP P, let X be a* good *end-component with $X \subseteq S$ for some $S \subseteq V$. For each i with $S \cap U_i = \emptyset$ it holds that $X \subseteq S \setminus \mathit{Attr}_R(P[S], L_i \cap S)$.*

For an index j with $S \cap U_j = \emptyset$ we call the vertices of $S \cap L_j$ *bad vertices*. The set of all bad vertices $\textsc{Bad}(S) = \bigcup_{1 \leq i \leq k} \{v \in L_i \cap S \mid U_i \cap S = \emptyset\}$ can be computed with $2k$ set operations.

3 Symbolic Divide-and-Conquer with Lock-Step Search

In this section we present a symbolic version of the lock-step search for strongly connected subgraphs [26]. This symbolic version is used in all subsequent results,

i.e., the sub-quadratic symbolic algorithms for graphs and MDPs with Streett objectives, and for MEC decomposition.

Divide-and-Conquer. The common property of the algorithmic problems we consider in this work is that the goal is to identify subgraphs of the input graph $G = (V, E)$ that are strongly connected and satisfy some additional properties. The difference between the problems lies in the required additional properties. We describe and analyze the Procedure Lock-Step-Search that we use in all our improved algorithms to efficiently implement a divide-and-conquer approach based on the requirement of strong connectivity, that is, we divide a subgraph $G[S]$, induced by a set of vertices S, into two parts that are not strongly connected within $G[S]$ or detect that $G[S]$ is strongly connected.

Start Vertices of Searches. The input to Procedure Lock-Step-Search is a set of vertices $S \subseteq V$ and two subsets of S denoted by H_S and T_S. In the algorithms that call the procedure as a subroutine, vertices contained in H_S have lost incoming edges (i.e., they were a "head" of a lost edge) and vertices contained in T_S have lost outgoing edges (i.e., they were a "tail" of a lost edge) since the last time a superset of S was identified as being strongly connected. For each vertex h of H_S the procedure conducts a backward search (i.e., a sequence of Pre operations) within $G[S]$ to find the vertices of S that can reach h; and analogously a forward search (i.e., a sequence of Post operations) from each vertex t of T_S is conducted.

Intuition for the Choice of Start Vertices. If the subgraph $G[S]$ is not strongly connected, then it contains at least one top SCC and at least one bottom SCC that are disjoint. Further, if for a superset $S' \supset S$ the subgraph $G[S']$ was strongly connected, then each top SCC of $G[S]$ contains a vertex that had an additional incoming edge in $G[S']$ compared to $G[S]$, and analogously each bottom SCC of $G[S]$ contains a vertex that had an additional outgoing edge. Thus by keeping track of the vertices that lost incoming or outgoing edges, the following invariant will be maintained by all our improved algorithms.

Invariant 1 (Start Vertices Sufficient). *We have $H_S, T_S \subseteq S$. Either (a) $H_S \cup T_S = \emptyset$ and $G[S]$ is strongly connected or (b) at least one vertex of each top SCC of $G[S]$ is contained in H_S and at least one vertex of each bottom SCC of $G[S]$ is contained in T_S.*

Lock-Step Search. The searches from the vertices of $H_S \cup T_S$ are performed in *lock-step*, that is, (a) one step is performed in each of the searches before the next step of any search is done and (b) all searches stop as soon as the first of the searches finishes. This is implemented in Procedure Lock-Step-Search as follows. A step in the search from a vertex $t \in T_S$ (and analogously for $h \in H_S$) corresponds to the execution of the iteration of the for-each loop for $t \in T_S$. In an iteration of a for-each loop we might discover that we do not need to consider this search further (see the paragraph on ensuring strong connectivity below) and update the set T_S (via T'_S) for future iterations accordingly. Otherwise the set C_t is either strictly increasing in this step of the search or the search for t

Procedure. LOCK-STEP-SEARCH(G, S, H_S, T_S)

```
   /* Pre and Post defined w.r.t. to G                                   */
 1 foreach v ∈ H_S ∪ T_S do C_v ← {v}
 2 while true do
 3     H'_S ← H_S, T'_S ← T_S
 4     foreach h ∈ H_S do /* search for top SCC                          */
 5         C'_h ← (C_h ∪ Pre(C_h)) ∩ S
 6         if |C'_h ∩ H'_S| > 1 then H'_S ← H'_S\{h}
 7         else
 8             if C'_h = C_h then return (C_h, H'_S, T'_S)
 9             C_h ← C'_h
10     foreach t ∈ T_S do /* search for bottom SCC                       */
11         C'_t ← (C_t ∪ Post(C_t)) ∩ S
12         if |C'_t ∩ T'_S| > 1 then T'_S ← T'_S\{t}
13         else
14             if C'_t = C_t then return (C_t, H'_S, T'_S)
15             C_t ← C'_t
16     H_S ← H'_S, T_S ← T'_S
```

terminates and we return the set of vertices in $G[S]$ that are reachable from t. So the two for-each loops over the vertices of T_S and H_S that are executed in an iteration of the while-loop perform one step of each of the searches and the while-loop stops as soon as a search stops, i.e., a return statement is executed and hence this implements properties (a) and (b) of lock-step search. Note that the while-loop terminates, i.e., a return statement is executed eventually because for all $t \in T_S$ (and resp. for all $h \in H_S$) the sets C_t are monotonically increasing over the iterations of the while-loop, we have $C_t \subseteq S$, and if some set C_t does not increase in an iteration, then it is either removed from T_S and thus not considered further or a return statement is executed. Note that when a search from a vertex $t \in T_S$ stops, it has discovered a maximal set of vertices C that can be reached from t; and analogously for $h \in H_S$. Figure 1 shows a small intuitive example of a call to the procedure.

Comparison to Explicit Algorithm. In the *explicit* version of the algorithm [7,26] the search from vertex $t \in T_S$ performs a depth-first search that terminates exactly when every *edge* reachable from t is explored. Since any search that starts outside of a bottom SCC but reaches the bottom SCC has to explore more edges than the search started inside of the bottom SCC, the first search from a vertex of T_S that terminates has exactly explored (one of) the smallest (in the number of edges) bottom SCC(s) of $G[S]$. Thus on explicit graphs the explicit lock-step search from the vertices of $H_S \cup T_S$ finds (one of) the smallest (in the number of edges) top or bottom SCC(s) of $G[S]$ in time proportional to the number of searches times the number of edges in the identified SCC. In *symbolically* represented graphs it can happen (1) that a search started outside of a bottom (resp. top) SCC terminates earlier than the search started within

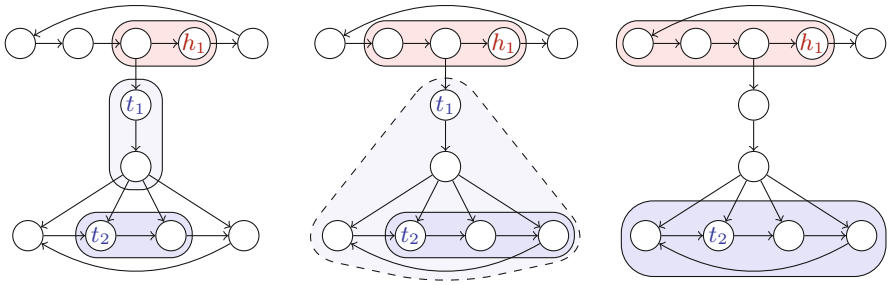


Fig. 1. An example of symbolic lock-step search showing the first three iterations of the main while-loop. Note that during the second iteration, the search started from t_1 is disregarded since it collides with t_2. In the subsequent fourth iteration, the search started from t_2 is returned by the procedure.

the bottom (resp. top) SCC and (2) that a search started in a larger (in the number of vertices) top or bottom SCC terminates before one in a smaller top or bottom SCC. We discuss next how we address these two challenges.

Ensuring Strong Connectivity. First, we would like the set returned by Procedure Lock-Step-Search to indeed be a top or bottom SCC of $G[S]$. For this we use the following observation for bottom SCCs that can be applied to top SCCs analogously. If a search starting from a vertex of $t_1 \in T_S$ encounters another vertex $t_2 \in T_S$, $t_1 \neq t_2$, there are two possibilities: either (1) both vertices are in the same SSC or (2) t_1 can reach t_2 but not vice versa. In Case (1) the searches from both vertices can explore all vertices in the SCC and thus it is sufficient to only search from one of them. In Case (2) the SCC of t_1 has an outgoing edge and thus cannot be a bottom SCC. Hence in both cases we can remove the vertex t_1 from the set T_S while still maintaining Invariant 1. By Invariant 1 we further have that each search from a vertex of T_S that is not in a bottom SCC encounters another vertex of T_S in its search and therefore is removed from the set T_S during Procedure Lock-Step-Search (if no top or bottom SCC is found earlier). This ensures that the returned set is either a top or a bottom SCC.[1]

Bound on Symbolic Steps. Second, observe that we can still bound the number of symbolic steps needed for the search that terminates first by the number of *vertices* in the smallest top or bottom SCC of $G[S]$, since this is an upper bound on the symbolic steps needed for the search started in this SCC. Thus provided Invariant 1, we can bound the number of symbolic steps in Procedure Lock-Step-Search to identify a vertex set $C \subsetneq S$ such that C and $S \backslash C$ are not strongly connected in $G[S]$ by $O((|H_S| + |T_S|) \cdot \min(|C|, |S \backslash C|))$. In the algorithms that call Procedure Lock-Step-Search we charge the number of symbolic steps in the procedure to the vertices in the smaller set of C and $S \backslash C$; this ensures that each vertex is charged at most $O(\log n)$ times over the whole algorithm. We obtain the following result (proof in [13, Appendix A]).

[1] To improve the practical performance, we return the updated sets H_S and T_S. By the above argument this preserves Invariant 1.

Theorem 1 (Lock-Step Search). *Provided Invariant 1 holds, Procedure* Lock-Step-Search (G, S, H_S, T_S) *returns a top or bottom SCC* C *of* $G[S]$. *It uses* $O((|H_S| + |T_S|) \cdot \min(|C|, |S \setminus C|))$ *symbolic steps if* $C \neq S$ *and* $O((|H_S| + |T_S|) \cdot |C|)$ *otherwise.*

4 Graphs with Streett Objectives

Basic Symbolic Algorithm. Recall that for a given graph (with n vertices) and a Streett objective (with k target pairs) each non-trivial strongly connected subgraph without bad vertices is a good component. The basic symbolic algorithm for graphs with Streett objectives repeatedly removes bad vertices from each SCC and then recomputes the SCCs until all good components are found. The winning set then consists of the vertices that can reach a good component. We refer to this algorithm as StreettGraphBasic. For the pseudocode and more details see [13, Appendix B].

Proposition 1. *Algorithm* StreettGraphBasic *correctly computes the winning set in graphs with Streett objectives and requires* $O(n \cdot \min(n, k))$ *symbolic steps.*

Improved Symbolic Algorithm. In our improved symbolic algorithm we replace the recomputation of all SCCs with the search for a new top or bottom SCC with Procedure Lock-Step-Search from vertices that have lost adjacent edges whenever there are not too many such vertices. We present the improved symbolic algorithm for graphs with Streett objectives in more detail as it also conveys important intuition for the MDP case. The pseudocode is given in Algorithm StreettGraphImpr.

Iterative Refinement of Candidate Sets. The improved algorithm maintains a set goodC of already identified good components that is initially empty and a set $\mathcal{X}$ of candidates for good components that is initialized with the SCCs of the input graph G. The difference to the basic algorithm lies in the properties of the vertex sets maintained in $\mathcal{X}$ and the way we identify sets that can be separated from each other without destroying a good component. In each iteration one vertex set S is removed from $\mathcal{X}$ and, after the removal of bad vertices from the set, either identified as a good component or split into several candidate sets. By Lemma 2 and Corollary 1 the following invariant is maintained throughout the algorithm for the sets in goodC and $\mathcal{X}$.

Invariant 2 (Maintained Sets). *The sets in* $\mathcal{X} \cup$ goodC *are pairwise disjoint and for every good component* C *of* G *there exists a set* $Y \supseteq C$ *such that either* $Y \in \mathcal{X}$ *or* $Y \in$ goodC.

Lost Adjacent Edges. In contrast to the basic algorithm, the subgraph induced by a set S contained in $\mathcal{X}$ is not necessarily strongly connected. Instead, we remember vertices of S that have lost adjacent edges since the last time a superset of S was determined to induce a strongly connected subgraph; vertices that lost

Algorithm. StreettGraphImpr. Improved Alg. for Graphs with Streett Obj.

Input : graph $G = (V, E)$ and Streett pairs $\text{TP} = \{(L_i, U_i) \mid 1 \leq i \leq k\}$
Output : $\langle\langle 1 \rangle\rangle (G, \text{Streett(TP)})$

```
1  X ← allSCCs(G); goodC ← ∅
2  foreach C ∈ X do  H_C ← ∅; T_C ← ∅
3  while X ≠ ∅ do
4      remove some S ∈ X from X
5      B ← ⋃_{1≤i≤k : U_i ∩ S = ∅} (L_i ∩ S)
6      while B ≠ ∅ do
7          S ← S \ B
8          H_S ← (H_S ∪ Post(B)) ∩ S
9          T_S ← (T_S ∪ Pre(B)) ∩ S
10         B ← ⋃_{1≤i≤k : U_i ∩ S = ∅} (L_i ∩ S)
11     if Post(S) ∩ S ≠ ∅ then /* G[S] contains at least one edge */
12         if |H_S| + |T_S| = 0 then goodC ← goodC ∪ {S}
13         else if |H_S| + |T_S| ≥ sqrt(m / log n) then
14             delete H_S and T_S
15             C ← allSCCs(G[S])
16             if |C| = 1 then goodC ← goodC ∪ {S}
17             else
18                 foreach C ∈ C do  H_C ← ∅; T_C ← ∅
19                 X ← X ∪ C
20         else
21             (C, H_S, T_S) ← Lock-Step-Search (G, S, H_S, T_S)
22             if C = S then goodC ← goodC ∪ {S}
23             else /* separate C and S\C */
24                 S ← S \ C
25                 H_C ← ∅; T_C ← ∅
26                 H_S ← (H_S ∪ Post(C)) ∩ S
27                 T_S ← (T_S ∪ Pre(C)) ∩ S
28                 X ← X ∪ {S} ∪ {C}
29 return GraphReach(G, ⋃_{C ∈ goodC} C)
```

incoming edges are contained in H_S and vertices that lost outgoing edges are contained in T_S. In this way we maintain Invariant 1 throughout the algorithm, which enables us to use Procedure Lock-Step-Search with the running time guarantee provided by Theorem 1.

Identifying SCCs. Let S be the vertex set removed from $\mathcal{X}$ in a fixed iteration of Algorithm StreettGraphImpr after the removal of bad vertices in the inner while-loop. First note that if S is strongly connected and contains at least one edge, then it is a good component. If the set S was already identified as strongly connected in a previous iteration, i.e., H_S and T_S are empty, then S is identified

as a good component in line 12. If many vertices of S have lost adjacent edges since the last time a super-set of S was identified as a strongly connected subgraph, then the SCCs of $G[S]$ are determined as in the basic algorithm. To achieve the optimal asymptotic upper bound, we say that many vertices of S have lost adjacent edges when we have $|H_S| + |T_S| \geq \sqrt{m/\log n}$, while lower thresholds are used in our experimental results. Otherwise, if not too many vertices of S lost adjacent edges, then we start a symbolic *lock-step search* for top SCCs from the vertices of H_S and for bottom SCCs from the vertices of T_S using Procedure Lock-Step-Search. The set returned by the procedure is either a top or a bottom SCC C of $G[S]$ (Theorem 1). Therefore we can from now on consider C and $S \backslash C$ separately, maintaining Invariants 1 and 2.

Algorithm StreettGraphImpr. A succinct description of the pseudocode is as follows: Lines 1–2 initialize the set of candidates for good components with the SCCs of the input graph. In each iteration of the main while-loop one candidate is considered and the following operations are performed: (a) lines 5–10 iteratively remove all bad vertices; if afterwards the candidate is still strongly connected (and contains at least one edge), it is identified as a good component in the next step; otherwise it is partitioned into new candidates in one of the following ways: (b) if many vertices lost adjacent edges, lines 13–17 partition the candidate into its SCCs (this corresponds to an iteration of the basic algorithm); (c) otherwise, lines 20–28 use symbolic lock-step search to partition the candidate into one of its SCCs and the remaining vertices. The while-loop terminates when no candidates are left. Finally, vertices that can reach some good component are returned. We have the following result (proof in [13, Appendix B]).

Theorem 2 (Improved Algorithm for Graphs). *Algorithm* StreettGraph*I*mpr *correctly computes the winning set in graphs with Streett objectives and requires* $O(n \cdot \sqrt{m \log n})$ *symbolic steps.*

5 Symbolic MEC Decomposition

In this section we present a succinct description of the basic symbolic algorithm for MEC decomposition and then present the main ideas for the improved algorithm.

Basic symbolic algorithm for MEC decomposition. The basic symbolic algorithm for MEC decomposition maintains a set of identified MECs and a set of candidates for MECs, initialized with the SCCs of the MDP. Whenever a candidate is considered, either (a) it is identified as a MEC or (b) it contains vertices with outgoing random edges, which are then removed together with their random attractor from the candidate, and the SCCs of the remaining sub-MDP are added to the set of candidates. We refer to the algorithm as MECBasic.

Proposition 2. *Algorithm* MECBasic *correctly computes the MEC decomposition of MDPs and requires* $O(n^2)$ *symbolic steps.*

Improved Symbolic Algorithm for MEC Decomposition. The improved symbolic algorithm for MEC decomposition uses the ideas of symbolic lock-step search presented in Sect. 3. Informally, when considering a candidate that lost a few edges from the remaining graph, we use the symbolic lock-step search to identify some bottom SCC. We refer to the algorithm as MECImpr. Since all the important conceptual ideas regarding the symbolic lock-step search are described in Sect. 3, we relegate the technical details to [13, Appendix C]. We summarize the main result (proof in [13, Appendix C]).

Theorem 3 (Improved Algorithm for MEC). *Algorithm* MECImpr *correctly computes the MEC decomposition of MDPs and requires* $O(n \cdot \sqrt{m})$ *symbolic steps.*

6 MDPs with Streett Objectives

Basic Symbolic Algorithm. We refer to the basic symbolic algorithm for MDPs with Streett objectives as StreettMDPBasic, which is similar to the algorithm for graphs, with SCC computation replaced by MEC computation. The pseudocode of Algorithm StreettMDPBasic together with its detailed description is presented in [13, Appendix D].

Proposition 3. *Algorithm* StreettMDPBasic *correctly computes the almost-sure winning set in MDPs with Streett objectives and requires* $O(n^2 \cdot \min(n, k))$ *symbolic steps.*

Remark. The above bound uses the basic symbolic MEC decomposition algorithm. Using our improved symbolic MEC decomposition algorithm, the above bound could be improved to $O(n \cdot \sqrt{m} \cdot \min(n, k))$.

Improved Symbolic Algorithm. We refer to the improved symbolic algorithm for MDPs with Streett objectives as StreettMDPImpr. First we present the main ideas for the improved symbolic algorithm. Then we explain the key differences compared to the improved symbolic algorithm for graphs. A thorough description with the technical details and proofs is presented in [13, Appendix D].

- First, we improve the algorithm by interleaving the symbolic MEC computation with the detection of bad vertices [7,31]. This allows to replace the computation of MECs in each iteration of the while-loop with the computation of SCCs and an additional random attractor computation.
  - *Intuition of interleaved computation.* Consider a candidate for a good end-component S after a random attractor to some bad vertices is removed from it. After the removal of the random attractor, the set S does not have random vertices with outgoing edges. Consider that further $\text{Bad}(S) = \emptyset$ holds. If S is strongly connected and contains an edge, then it is a good end-component. If S is not strongly connected, then $P[S]$ contains at least two SCCs and some of them might have random vertices with outgoing edges. Since end-components are strongly connected and do not have

random vertices with outgoing edges, we have that (1) every good end-component is completely contained in one of the SCCs of $P[S]$ and (2) the random vertices of an SCC with outgoing edges and their random attractor do not intersect with any good end-component (see Lemma 2).

- *Modification from basic to improved algorithm.* We use these observations to modify the basic algorithm as follows: First, for the sets that are candidates for good end-components, we do not maintain the property that they are end-components, but only that they do not have random vertices with outgoing edges (it still holds that every maximal good end-component is either already identified or contained in one of the candidate sets). Second, for a candidate set S, we repeat the removal of bad vertices until $\textsc{Bad}(S) = \emptyset$ holds before we continue with the next step of the algorithm. This allows us to make progress after the removal of bad vertices by computing all SCCs (instead of MECs) of the remaining sub-MDP. If there is only one SCC, then this is a good end-component (if it contains at least one edge). Otherwise (a) we remove from each SCC the set of random vertices with outgoing edges and their random attractor and (b) add the remaining vertices of each SCC as a new candidate set.

– Second, as for the improved symbolic algorithm for graphs, we use the symbolic lock-step search to quickly identify a top or bottom SCC every time a candidate has lost a small number of edges since the last time its superset was identified as being strongly connected. The symbolic lock-step search is described in detail in Sect. 3.

Using interleaved MEC computation and lock-step search leads to a similar algorithmic structure for Algorithm StreettMDPImpr as for our improved symbolic algorithm for graphs (Algorithm StreettGraphImpr). The key differences are as follows: First, the set of candidates for good end-components is initialized with the MECs of the input graph instead of the SCCs. Second, whenever bad vertices are removed from a candidate, also their random attractor is removed. Further, whenever a candidate is partitioned into its SCCs, for each SCC, the random attractor of the vertices with outgoing random edges is removed. Finally, whenever a candidate S is separated into C and $S \backslash C$ via symbolic lock-step search, the random attractor of the vertices with outgoing random edges is removed from C, and the random attractor of C is removed from S.

Theorem 4 (Improved Algorithm for MDPs). *Algorithm* Streett *MDP*Impr *correctly computes the almost-sure winning set in MDPs with Streett objectives and requires* $O(n \cdot \sqrt{m \log n})$ *symbolic steps.*

7 Experiments

We present a basic prototype implementation of our algorithm and compare against the basic symbolic algorithm for graphs and MDPs with Streett objectives.

Models. We consider the academic benchmarks from the VLTS benchmark suite [21], which gives representative examples of systems with nondeterminism, and has been used in previous experimental evaluation (such as [4,11]).

Specifications. We consider random LTL formulae and use the tool Rabinizer [28] to obtain deterministic Rabin automata. Then the negations of the formulae give us Streett automata, which we consider as the specifications.

Graphs. For the models of the academic benchmarks, we first compute SCCs, as all algorithms for Streett objectives compute SCCs as a preprocessing step. For SCCs of the model benchmarks we consider products with the specification Streett automata, to obtain graphs with Streett objectives, which are the benchmark examples for our experimental evaluation. The number of transitions in the benchmarks ranges from 300K to 5Million.

MDPs. For MDPs, we consider the graphs obtained as above and consider a fraction of the vertices of the graph as random vertices, which is chosen uniformly at random. We consider 10%, 20%, and 50% of the vertices as random vertices for different experimental evaluation.

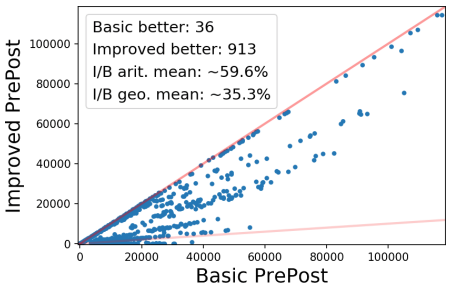
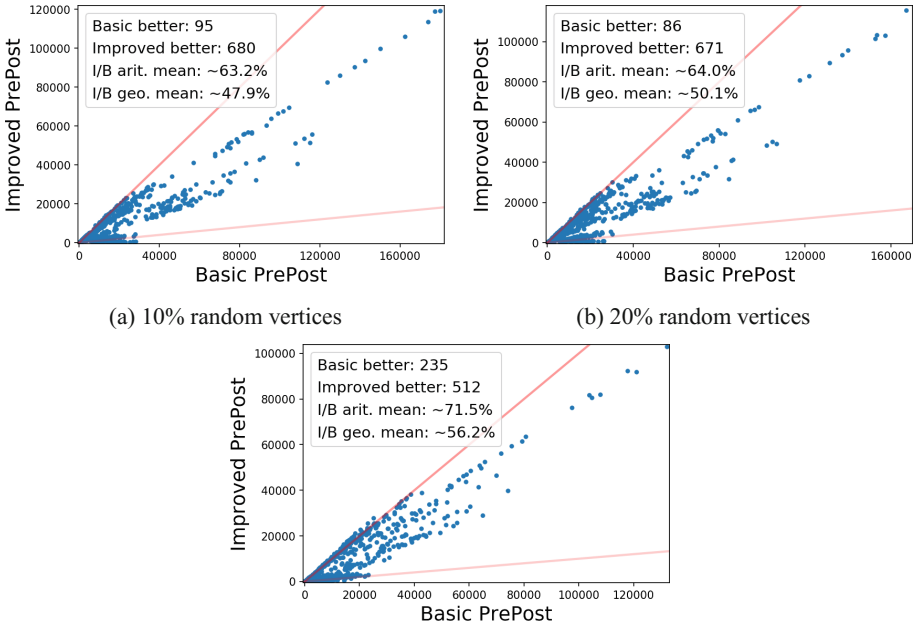


Fig. 2. Results for graphs with Streett objectives.

Experimental Evaluation. In the experimental evaluation we compare the number of symbolic steps (i.e., the number of Pre/Post operations[2]) executed by the algorithms, the comparison of running time yields similar results and is provided in [13, Appendix E]. As the initial preprocessing step is the same for all the algorithms (computing all SCCs for graphs and all MECs for MDPs), the comparison presents the number of symbolic steps executed after the preprocessing. The experimental results for graphs are shown in Fig. 2 and the experimental results for MDPs are shown in Fig. 3 (in each figure the two lines represent equality and an order-of-magnitude improvement, respectively).

Discussion. Note that the lock-step search is the key reason for theoretical improvement, however, the improvement relies on a large number of Streett pairs.

[2] Recall that the basic set operations are cheaper to compute, and asymptotically at most the number of Pre/Post operations in all the presented algorithms.



(a) 10% random vertices

(b) 20% random vertices

(c) 50% random vertices

Fig. 3. Results for MDPs with Streett objectives.

In the experimental evaluation, the LTL formulae generate Streett automata with small number of pairs, which after the product with the model accounts for an even smaller fraction of pairs as compared to the size of the state space. This has two effects:

- In the experiments the lock-step search is performed for a much smaller parameter value ($O(\log n)$ instead of the theoretically optimal bound of $\sqrt{m/\log n}$), and leads to a small improvement.
- For large graphs, since the number of pairs is small as compared to the number of states, the improvement over the basic algorithm is minimal.

In contrast to graphs, in MDPs even with small number of pairs as compared to the state-space, the interleaved MEC computation has a notable effect on practical performance, and we observe performance improvement even in large MDPs.

8 Conclusion

In this work we consider symbolic algorithms for graphs and MDPs with Streett objectives, as well as for MEC decomposition. Our algorithmic bounds match for both graphs and MDPs. In contrast, while SCCs can be computed in linearly

many symbolic steps no such algorithm is known for MEC decomposition. An interesting direction of future work would be to explore further improved symbolic algorithms for MEC decomposition. Moreover, further improved symbolic algorithms for graphs and MDPs with Streett objectives is also an interesting direction of future work.

Acknowledgements. K. C. and M. H. are partially supported by the Vienna Science and Technology Fund (WWTF) grant ICT15-003. K. C. is partially supported by the Austrian Science Fund (FWF): S11407-N23 (RiSE/SHiNE), and an ERC Start Grant (279307: Graph Games). V. T. is partially supported by the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie Grant Agreement No. 665385. V. L. is partially supported by the Austrian Science Fund (FWF): S11408-N23 (RiSE/SHiNE), the ISF grant #1278/16, and an ERC Consolidator Grant (project MPM). For M. H. and V. L. the research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013)/ERC Grant Agreement no. 340506.

References

1. Akers, S.B.: Binary decision diagrams. IEEE Trans. Comput. **C–27**(6), 509–516 (1978)
2. Alur, R., Henzinger, T.A.: Computer-aided verification (2004). http://www.cis.upenn.edu/group/cis673/
3. Baier, C., Katoen, J.P.: Principles of Model Checking. MIT Press, Cambridge (2008)
4. Barnat, J., Chaloupka, J., van de Pol, J.: Distributed algorithms for SCC decomposition. J. Log. Comput. **21**(1), 23–44 (2011)
5. Bloem, R., Gabow, H.N., Somenzi, F.: An algorithm for strongly connected component analysis in n log n symbolic steps. Form. Methods Syst. Des. **28**(1), 37–56 (2006)
6. Bryant, R.E.: Symbolic manipulation of Boolean functions using a graphical representation. In: Conference on Design Automation, DAC, pp. 688–694 (1985)
7. Chatterjee, K., Dvořák, W., Henzinger, M., Loitzenbauer, V.: Model and objective separation with conditional lower bounds: disjunction is harder than conjunction. In: LICS, pp. 197–206 (2016)
8. Chatterjee, K., Henzinger, M.: Faster and dynamic algorithms for maximal end-component decomposition and related graph problems in probabilistic verification. In: SODA, pp. 1318–1336 (2011)
9. Chatterjee, K., Henzinger, M.: An $O(n^2)$ time algorithm for alternating Büchi games. In: SODA, pp. 1386–1399 (2012)
10. Chatterjee, K., Henzinger, M.: Efficient and dynamic algorithms for alternating Büchi games and maximal end-component decomposition. J. ACM **61**(3), 15 (2014)
11. Chatterjee, K., Henzinger, M., Joglekar, M., Shah, N.: Symbolic algorithms for qualitative analysis of Markov decision processes with Büchi objectives. Form. Methods Syst. Des. **42**(3), 301–327 (2013)
12. Chatterjee, K., Henzinger, M., Loitzenbauer, V.: Improved algorithms for one-pair and k-pair Streett objectives. In: LICS, pp. 269–280 (2015)

13. Chatterjee, K., Henzinger, M., Loitzenbauer, V., Oraee, S., Toman, V.: Symbolic algorithms for graphs and Markov decision processes with fairness objectives. arXiv:1804.00206 (2018)
14. Chatterjee, K., Jurdziński, M., Henzinger, T.A.: Simple stochastic parity games. In: Baaz, M., Makowsky, J.A. (eds.) CSL 2003. LNCS, vol. 2803, pp. 100–113. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45220-1_11
15. Chatterjee, K., Dvořák, W., Henzinger, M., Loitzenbauer, V.: Lower bounds for symbolic computation on graphs: strongly connected components, liveness, safety, and diameter. In: SODA, pp. 2341–2356 (2018)
16. Chatterjee, K., Gaiser, A., Křetínský, J.: Automata with generalized Rabin pairs for probabilistic model checking and LTL synthesis. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 559–575. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39799-8_37
17. Ciesinski, F., Baier, C.: LiQuor: a tool for qualitative and quantitative linear time analysis of reactive systems. In: QEST, pp. 131–132 (2006)
18. Cimatti, A., Clarke, E., Giunchiglia, F., Roveri, M.: NUSMV: a new symbolic model checker. Int. J. Softw. Tools Technol. Transf. (STTT) **2**(4), 410–425 (2000)
19. Clarke Jr., E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press, Cambridge (1999)
20. Clarke, E., Grumberg, O., Peled, D.: Symbolic model checking. In: Model Checking. MIT Press (1999)
21. CWI/SEN2 and INRIA/VASY: The VLTS Benchmark Suite. http://cadp.inria.fr/resources/vlts
22. Dehnert, C., Junges, S., Katoen, J.-P., Volk, M.: A Storm is coming: a modern probabilistic model checker. In: Majumdar, R., Kunčak, V. (eds.) CAV 2017. LNCS, vol. 10427, pp. 592–600. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63390-9_31
23. Esparza, J., Křetínský, J.: From LTL to deterministic automata: a safraless compositional approach. In: Biere, A., Bloem, R. (eds.) CAV 2014. LNCS, vol. 8559, pp. 192–208. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08867-9_13
24. Gentilini, R., Piazza, C., Policriti, A.: Computing strongly connected components in a linear number of symbolic steps. In: SODA, pp. 573–582 (2003)
25. Gentilini, R., Piazza, C., Policriti, A.: Symbolic graphs: linear solutions to connectivity related problems. Algorithmica **50**(1), 120–158 (2008)
26. Henzinger, M.R., Telle, J.A.: Faster algorithms for the nonemptiness of Streett automata and for communication protocol pruning. In: Karlsson, R., Lingas, A. (eds.) SWAT 1996. LNCS, vol. 1097, pp. 16–27. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-61422-2_117
27. Holzmann, G.J.: The model checker SPIN. IEEE Trans. Softw. Eng. **23**(5), 279–295 (1997)
28. Komárková, Z., Křetínský, J.: Rabinizer 3: safraless translation of LTL to small deterministic automata. In: Cassez, F., Raskin, J.-F. (eds.) ATVA 2014. LNCS, vol. 8837, pp. 235–241. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11936-6_17
29. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 585–591. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22110-1_47
30. Lee, C.Y.: Representation of switching circuits by binary-decision programs. Bell Syst. Tech. J. **38**(4), 985–999 (1959)

31. Loitzenbauer, V.: Improved algorithms and conditional lower bounds for problems in formal verification and reactive synthesis. Ph.D. thesis. University of Vienna (2016)
32. Manna, Z., Pnueli, A.: Temporal Verification of Reactive Systems: Progress (Draft) (1996)
33. Ravi, K., Bloem, R., Somenzi, F.: A comparative study of symbolic algorithms for the computation of fair cycles. In: Hunt, W.A., Johnson, S.D. (eds.) FMCAD 2000. LNCS, vol. 1954, pp. 162–179. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-40922-X_10
34. Safra, S.: On the complexity of ω-automata. In: FOCS, pp. 319–327 (1988)
35. Somenzi, F.: CUDD: CU decision diagram package release 3.0.0 (2015). http://vlsi.colorado.edu/~fabio/CUDD/