Symbolic Algorithms for Qualitative Analysis of Markov Decision Processes with Büchi Objectives [⋆,⋆⋆]

Krishnendu Chatterjee[1], Monika Henzinger[2], Manas Joglekar[3], and Nisarg Shah[3]

[1] IST Austria
[2] University of Vienna
[3] IIT Bombay

Abstract. We consider Markov decision processes (MDPs) with ω-regular specifications given as parity objectives. We consider the problem of computing the set of *almost-sure* winning states from where the objective can be ensured with probability 1. The algorithms for the computation of the almost-sure winning set for parity objectives iteratively use the solutions for the almost-sure winning set for Büchi objectives (a special case of parity objectives). Our contributions are as follows: First, we present the first subquadratic symbolic algorithm to compute the almost-sure winning set for MDPs with Büchi objectives; our algorithm takes $O(n \cdot \sqrt{m})$ symbolic steps as compared to the previous known algorithm that takes $O(n^2)$ symbolic steps, where n is the number of states and m is the number of edges of the MDP. In practice MDPs often have constant out-degree, and then our symbolic algorithm takes $O(n \cdot \sqrt{n})$ symbolic steps, as compared to the previous known $O(n^2)$ symbolic steps algorithm. Second, we present a new algorithm, namely *win-lose* algorithm, with the following two properties: (a) the algorithm iteratively computes subsets of the almost-sure winning set and its complement, as compared to all previous algorithms that discover the almost-sure winning set upon termination; and (b) requires $O(n \cdot \sqrt{K})$ symbolic steps, where K is the maximal number of edges of strongly connected components (scc's) of the MDP. The win-lose algorithm requires symbolic computation of scc's. Third, we improve the algorithm for symbolic scc computation; the previous known algorithm takes linear symbolic steps, and our new algorithm improves the constants associated with the linear number of steps. In the worst case the previous known algorithm takes $5 \cdot n$ symbolic steps, whereas our new algorithm takes $4 \cdot n$ symbolic steps.

1 Introduction

Markov decision processes. The model of systems in verification of probabilistic systems are *Markov decision processes (MDPs)* that exhibit both probabilistic and nondeterministic behavior [12]. MDPs have been used to model and solve control problems for stochastic systems [10]: there, nondeterminism represents the freedom of the controller to choose a control action, while the probabilistic component of the behavior describes the system response to control actions. MDPs have also been adopted as models for concurrent probabilistic systems [6], probabilistic systems operating in open environments [18], and under-specified probabilistic systems [1]. A *specification*

⋆ This work was partially supported by FWF NFN Grant S11407-N23 (RiSE) and a Microsoft faculty fellowship.

⋆⋆ Details omitted due to lack of space available in [3].

describes the set of desired behaviors of the system, which in the verification and control of stochastic systems is typically an ω-regular set of paths. The class of ω-regular languages extends classical regular languages to infinite strings, and provides a robust specification language to express all commonly used specifications, such as safety, liveness, fairness, etc [21]. Parity objectives are a canonical way to define such ω-regular specifications. Thus MDPs with parity objectives provide the theoretical framework to study problems such as the verification and control of stochastic systems.

Qualitative and quantitative analysis. The analysis of MDPs with parity objectives can be classified into qualitative and quantitative analysis. Given an MDP with parity objective, the *qualitative analysis* asks for the computation of the set of states from where the parity objective can be ensured with probability 1 (almost-sure winning). The more general *quantitative analysis* asks for the computation of the maximal probability at each state with which the controller can satisfy the parity objective.

Importance of qualitative analysis. The qualitative analysis of MDPs is an important problem in verification that is of interest irrespective of the quantitative analysis problem. There are many applications where we need to know whether the correct behavior arises with probability 1. For instance, when analyzing a randomized embedded scheduler, we are interested in whether every thread progresses with probability 1 [8]. Even in settings where it suffices to satisfy certain specifications with probability $p < 1$, the correct choice of p is a challenging problem, due to the simplifications introduced during modeling. For example, in the analysis of randomized distributed algorithms it is quite common to require correctness with probability 1 (see, e.g., [16, 15, 20]). Furthermore, in contrast to quantitative analysis, qualitative analysis is robust to numerical perturbations and modeling errors in the transition probabilities, and consequently the algorithms for qualitative analysis are combinatorial. Finally, for MDPs with parity objectives, the best known algorithms and all algorithms used in practice first perform the qualitative analysis, and then performs a quantitative analysis on the result of the qualitative analysis [6, 7, 5]. Thus qualitative analysis for MDPs with parity objectives is one of the most fundamental and core problems in verification of probabilistic systems. One of the key challenges in probabilistic verification is to obtain efficient and symbolic algorithms for qualitative analysis of MDPs with parity objectives, as symbolic algorithms allow to handle MDPs with a large state space.

Previous results. The qualitative analysis for MDPs with parity objectives is achieved by iteratively applying solutions of the qualitative analysis of MDPs with Büchi objectives [6, 7, 5]. The qualitative analysis of an MDP with a parity objective with d priorities can be achieved by $O(d)$ calls to an algorithm for qualitative analysis of MDPs with Büchi objectives, and hence we focus on the qualitative analysis of MDPs with Büchi objectives. The classical algorithm for qualitative analysis for MDPs with Büchi objectives works in $O(n \cdot m)$ time, where n is the number of states, and m is the number of edges of the MDP [6, 7]. The classical algorithm can be implemented symbolically, and it takes at most $O(n^2)$ symbolic steps. An improved algorithm for the problem was given in [4] that works in $O(m \cdot \sqrt{m})$ time. The algorithm of [4] crucially depends on maintaining the same number of edges in certain forward searches. Thus the algorithm needs to explore edges of the graph explicitly and is inherently non-symbolic. In the literature, there is no symbolic subquadratic algorithm for qualitative analysis of MDPs with Büchi objectives.

Our contribution. In this work our main contributions are as follows.

1. We present a new and simpler subquadratic algorithm for qualitative analysis of MDPs with Büchi objectives that runs in $O(m \cdot \sqrt{m})$ time, and show that the algorithm can be implemented symbolically. The symbolic algorithm takes at most $O(n \cdot \sqrt{m})$ symbolic steps, and thus we obtain the first symbolic subquadratic algorithm. In practice, MDPs often have constant out-degree: for example, see [9] for MDPs with large state space but constant number of actions, or [10, 17] for examples from inventory management where MDPs have constant number of actions (the number of actions correspond to the out-degree of MDPs). For MDPs with constant out-degree our new symbolic algorithm takes $O(n \cdot \sqrt{n})$ symbolic steps, as compared to $O(n^2)$ symbolic steps of the previous best known algorithm.
2. All previous algorithms for the qualitative analysis of MDPs with Büchi objectives iteratively discover states that are guaranteed to be not almost-sure winning, and only when the algorithm terminates the almost-sure winning set is discovered. We present a new algorithm (namely *win-lose* algorithm) that iteratively discovers both states in the almost-sure winning set and its complement. Thus if the problem is to decide whether a given state s is almost-sure winning, and the state s is almost-sure winning, then the win-lose algorithm can stop at an intermediate iteration unlike all the previous algorithms. Our algorithm works in time $O(\sqrt{K_E} \cdot m)$ time, where K_E is the maximal number of edges of any scc of the MDP (in this paper we write scc for maximal scc). We also show that the win-lose algorithm can be implemented symbolically, and it takes at most $O(\sqrt{K_E} \cdot n)$ symbolic steps.
3. Our win-lose algorithm requires to compute the scc decomposition of a graph in $O(n)$ symbolic steps. The scc decomposition problem is one of the most fundamental problem in the algorithmic study of graph problems. The symbolic scc decomposition problem has many other applications in verification: for example, checking emptiness of ω-automata, and bad-cycle detection problems in model checking, see [2] for other applications. An $O(n \cdot \log n)$ symbolic step algorithm for scc decomposition was presented in [2], and the algorithm was improved in [11]. The algorithm of [11] is a linear symbolic step scc decomposition algorithm that requires at most $\min\{\, 5 \cdot n, 5 \cdot D \cdot N + N \,\}$ symbolic steps, where D is the diameter of the graph, and N is the number of scc's of the graph. We present an improved version of the symbolic scc decomposition algorithm. Our algorithm improves the constants of the number of the linear symbolic steps. Our algorithm requires at most $\min\{\, 3 \cdot n + N, 5 \cdot D^* + N \,\}$ symbolic steps, where D^* is the sum of the diameters of the scc's of the graph. Thus, in the worst case, the algorithm of [11] requires $5 \cdot n$ symbolic steps, whereas our algorithm requires $4 \cdot n$ symbolic steps. Moreover, the number of symbolic steps of our algorithm is always bounded by the number of symbolic steps of the algorithm of [11] (i.e. our algorithm is never worse).

Our experimental results show that our new algorithms perform better than the previous known algorithms both for qualitative analysis of MDPs with Büchi objectives and symbolic scc computation.

2 Definitions

Markov decision processes (MDPs). A *Markov decision process (MDP)* $G = ((S, E), (S_1, S_P), \delta)$ consists of a directed graph (S, E), a partition (S_1, S_P) of the *finite* set S of states, and a probabilistic transition function δ: $S_P \to \mathcal{D}(S)$, where $\mathcal{D}(S)$ denotes the set of probability distributions over the state space S. The states in S_1 are the *player-*1 states, where player 1 decides the successor state, and the states in S_P are the *probabilistic (or random)* states, where the successor state is chosen according to the probabilistic transition function δ. We assume that for $s \in S_P$ and $t \in S$, we have $(s, t) \in E$ iff $\delta(s)(t) > 0$, and we often write $\delta(s, t)$ for $\delta(s)(t)$. For a state $s \in S$, we write $E(s)$ to denote the set $\{ t \in S \mid (s, t) \in E \}$ of possible successors. For technical convenience we assume that every state in the graph (S, E) has at least one outgoing edge, i.e., $E(s) \neq \emptyset$ for all $s \in S$.

Plays and strategies. An infinite path, or a *play*, of the game graph G is an infinite sequence $\omega = \langle s_0, s_1, s_2, \ldots \rangle$ of states such that $(s_k, s_{k+1}) \in E$ for all $k \in \mathbb{N}$. We write Ω for the set of all plays, and for a state $s \in S$, we write $\Omega_s \subseteq \Omega$ for the set of plays that start from the state s. A *strategy* for player 1 is a function σ: $S^* \cdot S_1 \to \mathcal{D}(S)$ that chooses the probability distribution over the successor states for all finite sequences $\boldsymbol{w} \in S^* \cdot S_1$ of states ending in a player-1 state (the sequence represents a prefix of a play). A strategy must respect the edge relation: for all $\boldsymbol{w} \in S^*$ and $s \in S_1$, if $\sigma(\boldsymbol{w} \cdot s)(t) > 0$, then $t \in E(s)$. A strategy is *deterministic (pure)* if it chooses a unique successor for all histories (rather than a probability distribution), otherwise it is *randomized*. Player 1 follows the strategy σ if in each player-1 move, given that the current history of the game is $\boldsymbol{w} \in S^* \cdot S_1$, she chooses the next state according to $\sigma(\boldsymbol{w})$. We denote by Σ the set of all strategies for player 1. A *memoryless* player-1 strategy does not depend on the history of the play but only on the current state; i.e., for all $\boldsymbol{w}, \boldsymbol{w'} \in S^*$ and for all $s \in S_1$ we have $\sigma(\boldsymbol{w} \cdot s) = \sigma(\boldsymbol{w'} \cdot s)$. A memoryless strategy can be represented as a function σ: $S_1 \to \mathcal{D}(S)$, and a pure memoryless strategy can be represented as $\sigma : S_1 \to S$.

Once a starting state $s \in S$ and a strategy $\sigma \in \Sigma$ is fixed, the outcome of the MDP is a random walk ω_s^σ for which the probabilities of events are uniquely defined, where an *event* $\mathcal{A} \subseteq \Omega$ is a measurable set of plays. For a state $s \in S$ and an event $\mathcal{A} \subseteq \Omega$, we write $\Pr_s^\sigma(\mathcal{A})$ for the probability that a play belongs to $\mathcal{A}$ if the game starts from the state s and player 1 follows the strategy σ.

Objectives. We specify *objectives* for the player 1 by providing a set of *winning* plays $\Phi \subseteq \Omega$. We say that a play ω *satisfies* the objective Φ if $\omega \in \Phi$. We consider *ω-regular objectives* [21], specified as parity conditions. We also consider the special case of Büchi objectives.

- *Büchi objectives.* Let T be a set of target states. For a play $\omega = \langle s_0, s_1, \ldots \rangle \in \Omega$, we define $\mathrm{Inf}(\omega) = \{ s \in S \mid s_k = s$ for infinitely many $k \}$ to be the set of states that occur infinitely often in ω. The Büchi objective requires that some state of T be visited infinitely often, and defines the set of winning plays $\text{Büchi}(T) = \{ \omega \in \Omega \mid \mathrm{Inf}(\omega) \cap T \neq \emptyset \}$.
- *Parity objectives.* For $c, d \in \mathbb{N}$, we write $[c..d] = \{ c, c+1, \ldots, d \}$. Let p: $S \to [0..d]$ be a function that assigns a *priority* $p(s)$ to every state $s \in S$, where $d \in \mathbb{N}$. The *parity objective* is defined as $\mathrm{Parity}(p) = \{ \omega \in \Omega \mid$

$\min\big(p(\mathrm{Inf}(\omega))\big)$ is even $\}$. In other words, the parity objective requires that the minimum priority visited infinitely often is even. In the sequel we will use Φ to denote parity objectives.

Qualitative analysis: almost-sure winning. Given a player-1 objective Φ, a strategy $\sigma \in \Sigma$ is *almost-sure winning* for player 1 from the state s if $\mathrm{Pr}_s^{\sigma}(\Phi) = 1$. The *almost-sure winning set* $\langle\langle 1 \rangle\rangle_{almost}(\Phi)$ for player 1 is the set of states from which player 1 has an almost-sure winning strategy. The qualitative analysis of MDPs correspond to the computation of the almost-sure winning set for a given objective Φ. It follows from the results of [6, 7] that for all MDPs and all reachability and parity objectives, if there is an almost-sure winning strategy, then there is a memoryless almost-sure winning strategy. The qualitative analysis of MDPs with parity objectives is achieved by iteratively applying the solutions of qualitative analysis for MDPs with Büchi objectives [7, 5], and hence in this work we will focus on qualitative analysis for Büchi objectives.

Theorem 1 ([6, 7]). *For all MDPs G and all parity objectives Φ, there exists a pure memoryless strategy σ_* such that for all $s \in \langle\langle 1 \rangle\rangle_{almost}(\Phi)$ we have $\mathrm{Pr}_s^{\sigma_*}(\Phi) = 1$.*

Scc and bottom scc. Given a graph $G = (S, E)$, a set C of states is an scc if for all $s, t \in C$ there is a path from s to t going through states in C. In sequel we write scc for maximal scc. An scc C is a bottom scc if for all $s \in C$ all out-going edges are in C, i.e., $E(s) \subseteq C$.

Markov chains, closed recurrent sets. A Markov chain is a special case of MDP with $S_1 = \emptyset$, and hence for simplicity a Markov chain is a tuple $((S, E), \delta)$ with a probabilistic transition function $\delta : S \to \mathcal{D}(S)$, and $(s, t) \in E$ iff $\delta(s, t) > 0$. A *closed recurrent* set C of a Markov chain is a bottom scc in the graph (S, E). Let $\mathcal{C} = \bigcup_{C \text{ is closed recurrent}} C$. It follows from the results on Markov chains [14] that for all $s \in S$, the set $\mathcal{C}$ is reached with probability 1 in finite time, and for all C such that C is closed recurrent, for all $s \in C$ and for all $t \in C$, if the starting state is s, then the state t is visited infinitely often with probability 1.

Markov chain from a MDP and memoryless strategy. Given a MDP $G = ((S, E), (S_1, S_P), \delta)$ and a memoryless strategy $\sigma_* : S_1 \to \mathcal{D}(S)$ we obtain a Markov chain $G' = ((S, E'), \delta')$ as follows: $E' = E \cap (S_P \times S) \cup \{ (s, t) \mid s \in S_1, \sigma_*(s)(t) > 0 \}$; and $\delta'(s, t) = \delta(s, t)$ for $s \in S_P$, and $\delta'(s, t) = \sigma(s)(t)$ for $s \in S_1$ and $t \in E(s)$. We will denote by G_{σ_*} the Markov chain obtained from an MDP G by fixing a memoryless strategy σ_* in the MDP.

Symbolic encoding of an MDP. All algorithms of the paper will only depend on the graph (S, E) of the MDP and the partition (S_1, S_P), and not on the probabilistic transition function δ. Thus the symbolic encoding of an MDP is obtained as the standard encoding of a transition system (with an OBDD [19]), with one additional bit, and the bit denotes whether a state belongs to S_1 or S_P.

3 Symbolic Algorithms for Büchi Objectives

In this section we will present a new improved algorithm for the qualitative analysis of MDPs with Büchi objectives, and then present a symbolic implementation of the algorithm. Thus we obtain the first symbolic subquadratic algorithm for the problem. We start with the notion of *attractors* that is crucial for our algorithm.

Random and player 1 attractor. Given an MDP G, let $U \subseteq S$ be a subset of states. The *random attractor* $Attr_R(U)$ is defined inductively as follows: $X_0 = U$, and for $i \geq 0$, let $X_{i+1} = X_i \cup \{ s \in S_P \mid E(s) \cap X_i \neq \emptyset \} \cup \{ s \in S_1 \mid E(s) \subseteq X_i \}$. In other words, X_{i+1} consists of (a) states in X_i, (b) player-1 states whose all successors are in X_i and (c) random states that have at least one edge to X_i. Then $Attr_R(U) = \bigcup_{i \geq 0} X_i$. The definition of *player-1 attractor* $Attr_1(U)$ is analogous and is obtained by exchanging the role of random states and player 1 states in the above definition.

Property of attractors. Given an MDP G, and set U of states, let $A = Attr_R(U)$. Then from A player 1 cannot force to avoid U, in other words, for all states in A and for all player 1 strategies, the set U is reached with positive probability. For $A = Attr_1(U)$ there is a player 1 memoryless strategy to ensure that the set U is reached with certainty. The computation of random and player 1 attractor is the computation of alternating reachability and can be achieved in $O(m)$ time [13], and can be achieved in $O(n)$ symbolic steps.

3.1 A new subquadratic algorithm

The classical algorithm for computing the almost-sure winning set in MDPs with Büchi objectives has $O(n \cdot m)$ running time, and the symbolic implementation of the algorithm takes at most $O(n^2)$ symbolic steps. A subquadratic algorithm, with $O(m \cdot \sqrt{m})$ running time, for the problem was presented in [4]. The algorithm of [4] uses a mix of backward exploration and forward exploration. Every forward exploration step consists of executing a set of DFSs (depth first searches) simultaneously for a specified number of edges, and must maintain the exploration of the same number of edges in each of the DFSs. The algorithm thus depends crucially on maintaining the number of edges traversed explicitly, and hence the algorithm has no symbolic implementation. In this section we present a new subquadratic algorithm to compute $\langle\langle 1 \rangle\rangle_{almost}(\text{Büchi}(T))$. The algorithm is simpler as compared to the algorithm of [4] and we will show that our new algorithm can be implemented symbolically. Our new algorithm has some similar ideas as the algorithm of [4] in mixing backward and forward exploration, but the key difference is that the new algorithm never stops the forward exploration after a certain number of edges, and hence need not maintain the traversed edges explicitly. Thus the new algorithm is simpler, and our correctness and running time analysis proofs are different. We show that our new algorithm works in $O(m \cdot \sqrt{m})$ time, and requires at most $O(n \cdot \sqrt{m})$ symbolic steps.

Improved algorithm for almost-sure Büchi. Our algorithm iteratively removes states from the graph, until the almost-sure winning set is computed. At iteration i, we denote the remaining subgraph as (S_i, E_i), where S_i is the set of remaining states, E_i is the set of remaining edges, and the set of remaining target states as T_i (i.e., $T_i = S_i \cap T$). The set of states removed will be denoted by Z_i, i.e., $S_i = S \setminus Z_i$. The algorithm will ensure that (a) $Z_i \subseteq S \setminus \langle\langle 1 \rangle\rangle_{almost}(\text{Büchi}(T))$; and (b) for all $s \in S_i \cap S_P$ we have $E(s) \cap Z_i = \emptyset$. In every iteration the algorithm identifies a set Q_i of states such that there is no path from Q_i to the set T_i. Hence clearly $Q_i \subseteq S \setminus \langle\langle 1 \rangle\rangle_{almost}(\text{Büchi}(T))$. By the random attractor property from $Attr_R(Q_i)$ the set Q_i is reached with positive probability against any strategy for player 1. The algorithm maintains the set L_{i+1} of states that were removed from the graph since (and including) the last iteration of Case 1, and the set J_{i+1} of states that lost an edge to states removed from the graph since the last iteration of Case 1. Initially $L_0 := J_0 := \emptyset$, $Z_0 := \emptyset$, and let $i := 0$ and we describe the iteration i of our algorithm, and we call our algorithm ImprAlgo (Improved Algorithm) and the formal pseudocode is in [3].

1. *Case 1.* If $((|J_i| \geq \sqrt{m})$ or $i = 0)$, then
   (a) Let Y_i be the set of states that can reach the current target set T_i (this can be computed in $O(m)$ time by a graph reachability algorithm).
   (b) Let $Q_i := S_i \setminus Y_i$, i.e., there is no path from Q_i to T_i.
   (c) $Z_{i+1} := Z_i \cup \mathit{Attr}_R(Q_i)$. The set $\mathit{Attr}_R(Q_i)$ is removed from the graph.
   (d) The set L_{i+1} is the set of states removed from the graph in this iteration (i.e., $L_{i+1} := \mathit{Attr}_R(Q_i)$) and J_{i+1} be the set of states in the remaining graph with an edge to L_{i+1}.
   (e) If Q_i is empty, the algorithm stops, otherwise $i := i + 1$ and go to the next iteration.
2. *Case 2.* Else $(|J_i| \leq \sqrt{m})$, then
   (a) We do a lock-step search from every state s in J_i as follows: we do a DFS from s and (a) if the DFS tree reaches a state in T_i, then we stop the DFS search from s; and (b) if the DFS is completed without reaching a state in T_i, then we stop the entire lock-step search, and all states in the DFS tree are identified as Q_i. The set $\mathit{Attr}_R(Q_i)$ is removed from the graph and $Z_{i+1} := Z_i \cup \mathit{Attr}_R(Q_i)$. If DFS searches from all states s in J_i reach the set T_i, then the algorithm stops.
   (b) The set L_{i+1} is the set of states removed from the graph since the last iteration of Case 1 (i.e., $L_{i+1} := L_i \cup \mathit{Attr}_R(Q_i)$, where Q_i is the DFS tree that stopped without reaching T_i in the previous step of this iteration) and J_{i+1} be the set of states in the remaining graph with an edge to L_{i+1}, i.e., $J_{i+1} := (J_i \setminus \mathit{Attr}_R(Q_i)) \cup X_i$, where X_i is the subset of states of S_i with an edge to $\mathit{Attr}_R(Q_i)$.
   (c) $i := i + 1$ and go to the next iteration.

The proof of Lemma 1 is available in [3]. We then present the running time analysis.

Lemma 1. *Algorithm* ImprAlgo *correctly computes the set* $\langle\langle 1 \rangle\rangle_{almost}(\textit{Büchi}(T))$.

Lemma 2. *Given an MDP G with m edges, Algorithm* ImprAlgo *takes $O(m \cdot \sqrt{m})$ time.*

Proof. The total work of the algorithm, when Case 1 is executed, over all iterations is at most $O(\sqrt{m} \cdot m)$: this follows because between two iterations of Case 1 at least $\sqrt{m}$ edges must have been removed from the graph (since $|J_i| \geq \sqrt{m}$ everytime Case 1 is executed other than the case when $i = 0$), and hence Case 1 can be executed at most $m/\sqrt{m} = \sqrt{m}$ times. Since each iteration can be achieved in $O(m)$ time, the $O(m \cdot \sqrt{m})$ bound for Case 1 follows. We now show that the total work of the algorithm, when Case 2 is executed, over all iterations is at most $O(\sqrt{m} \cdot m)$. The argument is as follows: consider an iteration i such that Case 2 is executed. Then we have $|J_i| \leq \sqrt{m}$. Let Q_i be the DFS tree in iteration i while executing Case 2, and let $E(Q_i) = \cup_{s \in Q_i} E(s)$. The lock-step search ensures that the number of edges explored in this iteration is at most $|J_i| \cdot |E(Q_i)| \leq \sqrt{m} \times |E(Q_i)|$. Since Q_i is removed from the graph we *charge* the work of $\sqrt{m} \cdot |E(Q_i)|$ to edges in $E(Q_i)$, charging work $\sqrt{m}$ to each edge. Since there are at most m edges, the total charge of the work over all iterations when Case 2 is executed is at most $O(m \cdot \sqrt{m})$. Note that if instead of $\sqrt{m}$ we would have used a bound k in distinguishing Case 1 and Case 2, we would have achieved a running time bound of $O(m^2/k + m \cdot k)$, which is optimized by $k = \sqrt{m}$. ∎

Theorem 2. *Given an MDP G and a set T of target states, the algorithm* IMPRALGO *correctly computes the set $\langle\langle 1 \rangle\rangle_{almost}(\textit{Büchi}(T))$ in time $O(m \cdot \sqrt{m})$.*

3.2 Symbolic implementation of IMPRALGO

In this subsection we will a present symbolic implementation of each of the steps of algorithm IMPRALGO. The symbolic algorithm depends on the following symbolic operations that can be easily achieved with an OBDD implementation. For a set $X \subseteq S$ of states, let

$$\mathsf{Pre}(X) = \{ s \in S \mid E(s) \cap X \neq \emptyset \}; \qquad \mathsf{Post}(X) = \{ t \in S \mid t \in \bigcup_{s \in X} E(s) \};$$
$$\mathsf{CPre}(X) = \{ s \in S_P \mid E(s) \cap X \neq \emptyset \} \cup \{ s \in S_1 \mid E(s) \subseteq X \}.$$

In other words, $\mathsf{Pre}(X)$ is the predecessors of states in X; $\mathsf{Post}(X)$ is the successors of states in X; and $\mathsf{CPre}(X)$ is the set of states Y such that for every random state in Y there is a successor in X, and for every player 1 state in Y all successors are in Y.

We now present a symbolic version of IMPRALGO. For the symbolic version the basic steps are as follows: (i) Case 1 of the algorithm is same as Case 1 of IMPRALGO, and (ii) Case 2 is similar to Case 2 of IMPRALGO, and the only change in Case 2 is instead of lock-step search exploring the same number of edges, we have lock-step search that executes the same number of symbolic steps. The details of the symbolic implementation are as follows, and we will refer to the algorithm as SYMBIMPRALGO.

1. *Case 1.* In Case 1(a) we need to compute reachability to a target set T. The symbolic implementation is standard and done as follows: $X_0 = T$ and $X_{i+1} := X_i \cup \mathsf{Pre}(X_i)$ until $X_{i+1} = X_i$. The computation of the random attractor is also standard and is achieved as above replacing Pre by CPre. It follows that every iteration of Case 1 can be achieved in $O(n)$ symbolic steps.
2. *Case 2.* For analysis of Case 2 we present a symbolic implementation of the lock-step forward search. The lock-step ensures that each search executes the same number of symbolic steps. The implementation of the forward search from a state s in iteration i is achieved as follows: $P_0 := \{ s \}$ and $P_{j+1} := P_j \cup \mathsf{Post}(P_j)$ unless $P_{j+1} = P_j$ or $P_j \cap T_i \neq \emptyset$. If $P_j \cap T_i \neq \emptyset$, then the forward search is stopped from s. If $P_{j+1} = P_j$ and $P_j \cap T_i = \emptyset$, then we have identified that there is no path from states in P_j to T_i.
3. *Symbolic computation of cardinality of sets.* The other key operation required by the algorithm is determining whether the size of set J_i is at least $\sqrt{m}$ or not. The details of this symbolic operation is in [3].

Correctness and runtime analysis. The correctness of SYMBIMPRALGO is established following the correctness arguments for algorithm IMPRALGO. We now analyze the worst case number of symbolic steps. The total number of symbolic steps executed by Case 1 over all iterations is $O(n \cdot \sqrt{m})$ since between two executions of Case 1 at least $\sqrt{m}$ edges are removed, and every execution is achieved in $O(n)$ symbolic steps.

The work done for the symbolic cardinality computation is charged to the edges already removed from the graph, and hence the total number of symbolic steps over all iterations for the size computations is $O(m)$. We now show that the total number of symbolic steps executed over all iterations of Case 2 is $O(n \cdot \sqrt{m})$. The analysis is achieved as follows. Consider an iteration i of Case 2, and let the number of states removed in the iteration be n_i. Then the number of symbolic steps executed in this iteration for each of the forward search is at most n_i, and since $|J_i| \leq \sqrt{m}$, it follows that the number of symbolic steps executed is at most $n_i \cdot \sqrt{m}$. Since we remove n_i states, we *charge* each state removed from the graph with $\sqrt{m}$ symbolic steps for the total $n_i \cdot \sqrt{m}$ symbolic steps. Since there are at most n states, the total charge of symbolic steps over all iterations is $O(n \cdot \sqrt{m})$. Thus it follows that we have a symbolic algorithm to compute the almost-sure winning set for MDPs with Büchi objectives in $O(n \cdot \sqrt{m})$ symbolic steps.

Theorem 3. *Given an MDP G and a set T of target states, the symbolic algorithm* SYMBIMPRALGO *correctly computes $\langle\langle 1 \rangle\rangle_{almost}(\mathit{Büchi}(T))$ in $O(n \cdot \sqrt{m})$ symbolic steps.*

Remark 1. In many practical cases, MDPs have constant out-degree and hence we obtain a symbolic algorithm that works in $O(n \cdot \sqrt{n})$ symbolic steps, as compared to the previous known (symbolic implementation of the classical) algorithm that takes $O(n^2)$ symbolic steps.

4 The Win-Lose Algorithm

All the algorithms known for computing the almost-sure winning set (including the algorithms presented in the previous section) iteratively compute the set of states from where it is guaranteed that there is no almost-sure winning strategy for the player. The almost-sure winning set is discovered only when the algorithm stops. In this section, first we will present an algorithm that iteratively computes two sets W_1 and W_2, where W_1 is a subset of the almost-sure winning set, and W_2 is a subset of the complement of the almost-sure winning set. The algorithm has $O(K \cdot m)$ running time, where K is the size of the maximal strongly connected component (scc) of the graph of the MDP. We then present an improved version of the algorithm, using the techniques to obtain IMPRALGO from the classical algorithm, and finally present the symbolic implementation of the new algorithm.

4.1 The basic win-lose algorithm

The basic steps of the new algorithm are as follows. The algorithm maintains W_1 and W_2, that are guaranteed to be subsets of the almost-sure winning set and its complement respectively. Initially $W_1 = \emptyset$ and $W_2 = \emptyset$. We also maintain that $W_1 = \mathit{Attr}_1(W_1)$ and $W_2 = \mathit{Attr}_R(W_2)$. We denote by W the union of W_1 and W_2. We describe an iteration of the algorithm and we will refer to the algorithm as the WINLOSE algorithm (formal pseudocode in [3]).

1. *Step 1.* Compute the scc decomposition of the remaining graph of the MDP, i.e., scc decomposition of the MDP graph induced by $S \setminus W$.
2. *Step 2.* For every bottom scc C in the remaining graph: if $C \cap \mathsf{Pre}(W_1) \neq \emptyset$ or $C \cap T \neq \emptyset$, then $W_1 = \mathit{Attr}_1(W_1 \cup C)$; else $W_2 = \mathit{Attr}_R(W_2 \cup C)$, and the states in W_1 and W_2 are removed from the graph.

The stopping criterion is as follows: the algorithm stops when $W = S$. Observe that in each iteration, a set C of states is included in either W_1 or W_2, and hence W grows in each iteration.

Correctness of the algorithm. Note that in Step 2 we ensure that $Attr_1(W_1) = W_1$ and $Attr_R(W_2) = W_2$, and hence in the remaining graph there is no state of player 1 with an edge to W_1 and no random state with an edge to W_2. We show by induction that after every iteration $W_1 \subseteq \langle\langle 1 \rangle\rangle_{almost}(\text{Büchi}(T))$ and $W_2 \subseteq S \setminus \langle\langle 1 \rangle\rangle_{almost}(\text{Büchi}(T))$. The base case (with $W_1 = W_2 = \emptyset$) follows trivially. We prove the inductive case considering the following two cases.

1. Consider a bottom scc C in the remaining graph such that $C \cap \mathsf{Pre}(W_1) \neq \emptyset$ or $C \cap T \neq \emptyset$. Consider the randomized memoryless strategy σ for the player that plays all edges in C uniformly at random, i.e., for $s \in C$ we have $\sigma(s)(t) = \frac{1}{|E(s) \cap C|}$ for $t \in E(s) \cap C$. If $C \cap \mathsf{Pre}(W_1) \neq \emptyset$, then the strategy ensures that W_1 is reached with probability 1, since $W_1 \subseteq \langle\langle 1 \rangle\rangle_{almost}(\text{Büchi}(T))$ by inductive hypothesis it follows $C \subseteq \langle\langle 1 \rangle\rangle_{almost}(\text{Büchi}(T))$. Hence $Attr_1(W_1 \cup C) \subseteq \langle\langle 1 \rangle\rangle_{almost}(\text{Büchi}(T))$. If $C \cap T \neq \emptyset$, then since there is no edge from random states to W_2, it follows that under the randomized memoryless strategy σ, the set C is a closed recurrent set of the resulting Markov chain, and hence every state is visited infinitely often with probability 1. Since $C \cap T \neq \emptyset$, it follows that $C \subseteq \langle\langle 1 \rangle\rangle_{almost}(\text{Büchi}(T))$, and hence $Attr_1(W_1 \cup C) \subseteq \langle\langle 1 \rangle\rangle_{almost}(\text{Büchi}(T))$.
2. Consider a bottom scc C in the remaining graph such that $C \cap \mathsf{Pre}(W_1) = \emptyset$ and $C \cap T = \emptyset$. Then consider any strategy for player 1: (a) If a play starting from a state in C stays in the remaining graph, then since C is a bottom scc, it follows that the play stays in C with probability 1. Since $C \cap T = \emptyset$ it follows that T is never visited. (b) If a play leaves C (note that C is a bottom scc of the remaining graph and not the original graph, and hence a play may leave C), then since $C \cap \mathsf{Pre}(W_1) = \emptyset$, it follows that the play reaches W_2, and by hypothesis $W_2 \subseteq S \setminus \langle\langle 1 \rangle\rangle_{almost}(\text{Büchi}(T))$. In either case it follows that $C \subseteq S \setminus \langle\langle 1 \rangle\rangle_{almost}(\text{Büchi}(T))$. It follows that $Attr_R(W_2 \cup C) \subseteq S \setminus \langle\langle 1 \rangle\rangle_{almost}(\text{Büchi}(T))$.

The correctness of the algorithm follows as when the algorithm stops we have $W_1 \cup W_2 = S$ and running time analysis is given in [3].

Theorem 4. *Given an MDP with a Büchi objective, the* WINLOSE *algorithm iteratively computes the subsets of the almost-sure winning set and its complement, and in the end correctly computes the set $\langle\langle 1 \rangle\rangle_{almost}(\text{Büchi}(T))$ and the algorithm runs in time $O(K_S \cdot m)$, where K_S is the maximum number of states in an scc of the graph of the MDP.*

4.2 Improved WINLOSE algorithm and symbolic implementation

Improved WINLOSE algorithm. The improved version of the WINLOSE algorithm performs a forward exploration to obtain a bottom scc like Case 2 of IMPRALGO. At iteration i, we denote the remaining subgraph as (S_i, E_i), where S_i is the set of remaining states, and E_i is the set of remaining edges. The set of states removed will be denoted by Z_i, i.e., $S_i = S \setminus Z_i$, and Z_i is the union of W_1 and W_2. In every iteration the algorithm identifies a set C_i of states such that C_i is a bottom scc in the remaining graph, and then it follows the steps of the WINLOSE algorithm. We will consider two cases. The algorithm maintains the set L_{i+1} of states that were removed from the graph since (and including) the last iteration of Case 1, and the set J_{i+1} of states that lost an edge to states removed from the graph since the last iteration of Case 1. Initially $J_0 := L_0 := Z_0 := W_1 := W_2 := \emptyset$, and let $i := 0$ and we describe the iteration i of our algorithm. We call our algorithm IMPRWINLOSE (formal pseudocode in [3]).

1. *Case 1.* If $((|J_i| \geq \sqrt{m})$ or $i = 0)$, then
   (a) Compute the scc decomposition of the remaining graph.
   (b) For each bottom scc C_i, if $C_i \cap T \neq \emptyset$ or $C_i \cap \mathsf{Pre}(W_1) \neq \emptyset$, then $W_1 := \mathit{Attr}_1(W_1 \cup C_i)$, else $W_2 := \mathit{Attr}_R(W_2 \cup C_i)$.
   (c) $Z_{i+1} := W_1 \cup W_2$. The set $Z_{i+1} \setminus Z_i$ is removed from the graph.
   (d) The set L_{i+1} is the set of states removed from the graph in this iteration and J_{i+1} be the set of states in the remaining graph with an edge to L_{i+1}.
   (e) If Z_i is S, the algorithm stops, otherwise $i := i + 1$ and go to the next iteration.
2. *Case 2.* Else $(|J_i| \leq \sqrt{m})$, then
   (a) Consider the set J_i to be the set of vertices in the graph that lost an edge to the states removed since the last iteration that executed Case 1.
   (b) We do a lock-step search from every state s in J_i as follows: we do a DFS from s, until the DFS stops. Once the DFS stops we have identified a bottom scc C_i.
   (c) If $C_i \cap T \neq \emptyset$ or $C_i \cap \mathsf{Pre}(W_1) \neq \emptyset$, then $W_1 := \mathit{Attr}_1(W_1 \cup C_i)$, else $W_2 := \mathit{Attr}_R(W_2 \cup C_i)$.
   (d) $Z_{i+1} := W_1 \cup W_2$. The set $Z_{i+1} \setminus Z_i$ is removed from the graph.
   (e) The set L_{i+1} is the set of states removed from the graph since the last iteration of Case 1 and J_{i+1} be the set of states in the remaining graph with an edge to L_{i+1}.
   (f) If $Z_i = S$, the algorithm stops, otherwise $i := i+1$ and go to the next iteration.

Correctness and running time. The correctness of the algorithm follows from the correctness of the WINLOSE algorithm. The running time analysis of the algorithm is similar to IMPRALGO algorithm, and this shows the algorithm runs in $O(m \cdot \sqrt{m})$ time. Applying the IMPRWINLOSE algorithm bottom up on the scc decomposition of the MDP gives us a running time of $O(m \cdot \sqrt{K_E})$, where K_E is the maximum number of edges of an scc of the MDP.

Theorem 5. *Given an MDP with a Büchi objective, the* IMPRWINLOSE *algorithm iteratively computes the subsets of the almost-sure winning set and its complement, and in the end correctly computes the set $\langle\langle 1 \rangle\rangle_{almost}(\mathit{Büchi}(T))$ and the algorithm runs in time $O(\sqrt{K_E} \cdot m)$, where K_E is the maximum number of edges in an scc of the graph of the MDP.*

Symbolic implementation. The symbolic implementation of IMPRWINLOSE algorithm is obtained in a similar fashion as SYMBIMPRALGO was obtained from IMPRALGO. The only additional step required is the symbolic scc computation. It follows from the results of [11] that scc decomposition can be computed in $O(n)$ symbolic steps. In the following section we will present an improved symbolic scc computation algorithm.

Corollary 1. *Given an MDP with a Büchi objective, the symbolic* IMPRWINLOSE *algorithm (*SYMBIMPRWINLOSE*) iteratively computes the subsets of the almost-*

sure winning set and its complement, and in the end correctly computes the set $\langle\langle 1\rangle\rangle_{almost}(\text{Büchi}(T))$ *and the algorithm runs in* $O(\sqrt{K_E}\cdot n)$ *symbolic steps, where* K_E *is the maximum number of edges in an scc of the graph of the MDP.*

Remark 2. It is clear from the complexity of the WINLOSE and IMPRWINLOSE algorithms that they would perform better for MDPs where the graph has many small scc's, rather than few large ones.

5 Improved Symbolic SCC Algorithm

A symbolic algorithm to compute the scc decomposition of a graph in $O(n\cdot\log n)$ symbolic steps was presented in [2]. The algorithm of [2] was based on forward and backward searches. The algorithm of [11] improved the algorithm of [2] to obtain an algorithm for scc decomposition that takes at most linear amount of symbolic steps. In this section we present an improved version of the algorithm of [11] that improves the constants of the number of linear symbolic steps required. We first describe the main ideas of the algorithm of [11] and then present our improved algorithm. The algorithm of [11] improves the algorithm of [2] by maintaining the right order for forward sets. The notion of *spine-sets* and *skeleton of a forward set* was designed for this purpose.

Spine-sets and skeleton of a forward set. Let $G=(S,E)$ be a directed graph. Consider a finite path $\tau=(s_0,s_1,\ldots,s_\ell)$, such that for all $0\leq i\leq \ell-1$ we have $(s_i,s_{i+1})\in E$. The path is *chordless* if for all $0\leq i<j\leq\ell$ such that $j-i>1$, there is no edge from s_i to s_j. Let $U\subseteq S$. The pair (U,s) is a *spine-set* of G iff G contains a chordless path whose set of states is U that ends in s. For a state s, let $\mathsf{FW}(s)$ denote the set of states that is reachable from s (i.e., reachable by a forward search from s). The set (U,t) is a *skeleton of* $\mathsf{FW}(s)$ iff t is a state in $\mathsf{FW}(s)$ whose distance from s is maximum and U is the set of states on a shortest path from s to t. The following lemma was shown in [11] establishing relation of skeleton of forward set and spine-set.

Lemma 3 ([11]). *Let* $G=(S,E)$ *be a directed graph, and let* $\mathsf{FW}(s)$ *be the forward set of* $s\in S$. *The following assertions hold: (1) If* (U,t) *is a skeleton of a forward-set* $\mathsf{FW}(s)$*, then* $U\subseteq\mathsf{FW}(s)$*. (2) If* (U,t) *is a skeleton of* $\mathsf{FW}(s)$*, then* (U,t) *is a spine-set in* G*.*

The intuitive idea of the algorithm. The algorithm of [11] is a recursive algorithm, and in every recursive call the scc of a state s is determined by computing $\mathsf{FW}(s)$, and then identifying the set of states in $\mathsf{FW}(s)$ having a path to s. The choice of the state to be processed next is guided by the implicit inverse order associated with a possible spine-set. This is achieved as follows: whenever a forward-set $\mathsf{FW}(s)$ is computed, a skeleton of such a forward set is also computed. The order induced by the skeleton is then used for the subsequent computations. Thus the symbolic steps performed to compute $\mathsf{FW}(s)$ is distributed over the scc computation of the states belonging to a skeleton of $\mathsf{FW}(s)$. The key to establish the linear complexity of symbolic steps is the amortized analysis. We now present the main procedure SCCFIND and the main sub-procedure SKELFWD of the algorithm from [11].

Procedures SCCFIND **and** SKELFWD**.** The main procedure of the algorithm is SCCFIND that calls SKELFWD as a sub-procedure. The input to SCCFIND is a graph

(S, E) and (A, B), where either $(A, B) = (\emptyset, \emptyset)$ or $(A, B) = (U, \{ s \})$, where (U, s) is a spine-set. If S is $\emptyset$, then the algorithm stops. Else, (a) if (A, B) is $(\emptyset, \emptyset)$, then the procedure picks an arbitrary s from S and proceeds; (b) otherwise, the sub-procedure SKELFWD is invoked to compute the forward set of s together with the skeleton (U', s') of such a forward set. The SCCFIND procedure has the following local variables: FWSet, NewSet, NewState and SCC. The variable FWSet that maintains the forward set, whereas NewSet and NewState maintain U' and $\{ s' \}$, respectively. The variable SCC is initialized to s, and then augmented with the scc containing s. The partition of the scc's is updated and finally the procedure is recursively called over:

1. the subgraph of (S, E) is induced by $S \setminus \mathsf{FWSet}$ and the spine-set of such a subgraph is obtained from $(U, \{ t \})$ by subtracting SCC;
2. the subgraph of (S, E) induced by FWSet $\setminus$ SCC and the spine-set of such a subgraph obtained from (NewSet, NewState) by subtracting SCC.

The SKELFWD procedure takes as input a graph (S, E) and a state s, first it computes the forward set $\mathsf{FW}(s)$, and second it computes the skeleton of the forward set. The forward set is computed by symbolic breadth first search, and the skeleton is computed with a stack. The detailed pseudocodes are in [3]. We will refer to this algorithm of [11] as SYMBOLICSCC. The following result was established in [11]: for the proof of the constant 5, refer to the appendix of [11] and the last sentence explicitly claims that every state is charged at most 5 symbolic steps.

Theorem 6 ([11]). *Let $G = (S, E)$ be a directed graph. The algorithm* SYMBOLICSCC *correctly computes the scc decomposition of G in $\min\{ 5 \cdot |S|, 5 \cdot D(G) \cdot N(G) + N(G) \}$ symbolic steps, where $D(G)$ is the diameter of G, and $N(G)$ is the number of scc's in G.*

Improved symbolic algorithm. We now present our improved symbolic scc algorithm and refer to the algorithm as IMPROVEDSYMBOLICSCC. Our algorithm mainly modifies the sub-procedure SKELFWD. The improved version of SKELFWD procedure takes an additional input argument Q, and returns an additional output argument that is stored as a set P by the calling SCCFIND procedure. The calling function passes the set U as Q. The way the output P is computed is as follows: at the end of the forward search we have the following assignment: $P := \mathsf{FWSet} \cap Q$. After the forward search, the skeleton of the forward set is computed with the help of a stack. The elements of the stacks are sets of states stored in the forward search. The spine set computation is similar to SKELFWD, the difference is that when elements are popped of the stack, we check if there is a non-empty intersection with P, if so, we break the loop and return. Moreover, for the backward searches in SCCFIND we initialize SCC by P rather than s. We refer to the new sub-procedure as IMPROVEDSKELFWD (detailed pseudocode in [3]).

Correctness. Since s is the last element of the spine set U, and P is the intersection of a forward search from s with U, it means that all elements of P are both reachable from s (since P is a subset of $\mathsf{FW}(s)$) and can reach s (since P is a subset of U). It follows that P is a subset of the scc containing s. Hence not computing the spine-set beyond P does not change the future function calls, i.e., the value of U', since the omitted parts of NewSet are in the scc containing s. The modification of starting the backward search from P does not change the result, since P will anyway be included in the backward search. So the IMPROVEDSYMBOLICSCC algorithm gives the same result as SYMBOLICSCC, and the correctness follows from Theorem 6.

Symbolic steps analysis. We present two upper bounds on the number of symbolic steps of the algorithm. Intuitively following are the symbolic operations that need to be accounted for: (1) when a state is included in a spine set for the first time in IMPROVEDSKELFWD sub-procedure which has two parts: the first part is the forward search and the second part is computing the skeleton of the forward set; (2) when a state is already in a spine set and is found in forward search of IMPROVEDSKELFWD and (3) the backward search for determining the scc. We now present the number of symbolic steps analysis for IMPROVEDSYMBOLICSCC.

1. There are two parts of IMPROVEDSKELFWD, (i) a forward search and (ii) a backward search for skeleton computation of the forward set. For the backward search, we show that the number of steps performed equals the size of NewSet computed. One key idea of the analysis is the proof where we show that a state becomes part of spine-set at most once, as compared to the algorithm of [11] where a state can be part of spine-set at most twice. Because, when it is already part of a spine-set, it will be included in P and we stop the computation of spine-set when an element of P gets included. We now split the analysis in two cases: (a) states that are included in spine-set, and (b) states that are not included in spine-set.
   (a) We charge one symbolic step for the backward search of IMPROVEDSKELFWD (spine-set computation) to each element when it first gets inserted in a spine-set. For the forward search, we see that the number of steps performed is the size of spine-set that would have been computed if we did not stop the skeleton computation. But by stopping it, we are only omitting states that are part of the scc. Hence we charge one symbolic step to each state getting inserted into spine-set for the first time and each state of the scc. Thus, a state getting inserted in a spine-set is charged two symbolic steps (for forward and backward search) of IMPROVEDSKELFWD the first time it is inserted.
   (b) A state not inserted in any spine-set is charged one symbolic step for backward search which determines the scc.

   Along with the above symbolic steps, one step is charged to each state for the forward search in IMPROVEDSKELFWD at the time its scc is being detected. Hence each state gets charged at most three symbolic steps. Besides, for computing NewState, one symbolic step is required per scc found. Thus the total number of symbolic steps is bounded by $3 \cdot |S| + N(G)$, where $N(G)$ is the number of scc's of G.
2. Let D^* be the sum of diameters of the scc's in a G. Consider a scc with diameter d. In any scc the spine-set is a shortest path, and hence the size of the spine-set is bounded by d. Thus the three symbolic steps charged to states in spine-set contribute to at most $3 \cdot d$ symbolic steps for the scc. Moreover, the number of iterations of forward search of IMPROVEDSKELFWD charged to states belonging to the scc being computed are at most d. And the number of iterations of the backward search to compute the scc is also at most d. Hence, the two symbolic steps charged to states not in any spine-set also contribute at most $2 \cdot d$ symbolic steps for the scc. Finally, computation of NewSet takes one symbolic step per scc. Hence we have

$5 \cdot d + 1$ symbolic steps for a scc with diameter d. We thus obtain an upper bound of $5D^* + N(G)$ symbolic steps.

It is straightforward to argue that the number of symbolic steps of IMPROVEDSCCFIND is at most the number of symbolic steps of SCCFIND. The detailed pseudocode and running time analysis is presented in [3].

Theorem 7. *Let $G = (S, E)$ be a directed graph. The algorithm* IMPROVEDSYMBOLICSCC *correctly computes the scc decomposition of G in $\min\{\ 3 \cdot |S| + \cdot N(G), 5 \cdot D^*(G) + N(G)\ \}$ symbolic steps, where $D^*(G)$ is the sum of diameters of the scc's of G, and $N(G)$ is the number of scc's in G.*

Remark 3. Observe that in the worst case SCCFIND takes $5 \cdot n$ symbolic steps, whereas IMPROVEDSCCFIND takes at most $4 \cdot n$ symbolic steps. Thus our algorithm improves the constant of the number of linear symbolic steps required for symbolic scc decomposition.

6 Experimental Results

In this section we present our experimental results. We first present the results for symbolic algorithms for MDPs with Büchi objectives and then for symbolic scc decomposition. We present the results for symbolic steps comparison and the running time comparison is similar (see [3]).

Symbolic algorithm for MDPs with Büchi objectives. We implemented all the symbolic algorithms (including the classical one) and ran the algorithms on randomly generated graphs. If we consider arbitrarily randomly generated graphs, then in most cases it gives rise to trivial MDPs. Hence we generated large number of MDP graphs randomly, first chose the ones where all the algorithms required the most number of symbolic steps, and then considered random graphs obtained by small uniform perturbations of them. Our results of average symbolic steps required are shown in Table 1 and show that the new algorithms perform significantly better than the classical algorithm.

Number of states	Classical	SYMBIMPRALGO	SYMBIMPRWINLOSE
5000	16508	3382	4007
10000	57438	6807	7146
20000	121376	11110	12519

Table 1. The symbolic steps required by symbolic algorithms for MDPs with Büchi objectives.

Symbolic scc computation. We implemented the symbolic scc decomposition algorithm from [11] and our new symbolic algorithm. We ran the algorithms on randomly generated graphs. Again arbitrarily randomly generated graphs in many cases gives rise to graphs that are mostly disconnected or completely connected. Hence we generated random graphs by first constructing a topologically sorted order of the scc's and then adding edges randomly respecting the topologically sorted order. Our results of average symbolic steps are shown in Table 2 and shows that our new algorithm performs better (around 15% improvement).

Number of states	Algorithm from [11]	Our Algorithm	Percentage Improvement
10000	1045	877	16.06
25000	2642	2262	14.38
50000	6298	5398	14.27

Table 2. The symbolic steps required for scc computation.

References

1. A. Bianco and L. de Alfaro. Model checking of probabilistic and nondeterministic systems. In *FSTTCS 95*, volume 1026 of *LNCS*, pages 499–513. Springer-Verlag, 1995.
2. R. Bloem, H. N. Gabow, and F. Somenzi. An algorithm for strongly connected component analysis in log symbolic steps. In *FMCAD*, pages 37–54, 2000.
3. K. Chatterjee, M. Henzinger, M. Joglekar, and N. Shah. Symbolic algorithms for qualitative analysis of Markov decision processes with Büchi objectives. *CoRR*, 2011. http://arxiv.org/abs/1104.3348.
4. K. Chatterjee, M. Jurdziński, and T.A. Henzinger. Simple stochastic parity games. In *CSL'03*, volume 2803 of *LNCS*, pages 100–113. Springer, 2003.
5. K. Chatterjee, M. Jurdziński, and T.A. Henzinger. Quantitative stochastic parity games. In *SODA'04*, pages 121–130. SIAM, 2004.
6. C. Courcoubetis and M. Yannakakis. The complexity of probabilistic verification. *Journal of the ACM*, 42(4):857–907, 1995.
7. L. de Alfaro. *Formal Verification of Probabilistic Systems*. PhD thesis, Stanford University, 1997.
8. L. de Alfaro, M. Faella, R. Majumdar, and V. Raman. Code-aware resource management. In *EMSOFT 05*. ACM, 2005.
9. L. de Alfaro and P. Roy. Magnifying-lens abstraction for Markov decision processes. In *CAV*, pages 325–338, 2007.
10. J. Filar and K. Vrieze. *Competitive Markov Decision Processes*. Springer-Verlag, 1997.
11. R. Gentilini, C. Piazza, and A. Policriti. Computing strongly connected components in a linear number of symbolic steps. In *SODA*, pages 573–582, 2003.
12. H. Howard. *Dynamic Programming and Markov Processes*. MIT Press, 1960.
13. N. Immerman. Number of quantifiers is better than number of tape cells. *Journal of Computer and System Sciences*, 22:384–406, 1981.
14. J.G. Kemeny, J.L. Snell, and A.W. Knapp. *Denumerable Markov Chains*. D. Van Nostrand Company, 1966.
15. M. Kwiatkowska, G. Norman, and D. Parker. Verifying randomized distributed algorithms with prism. In *Workshop on Advances in Verification (WAVE'00)*, 2000.
16. A. Pogosyants, R. Segala, and N. Lynch. Verification of the randomized consensus algorithm of Aspnes and Herlihy: a case study. *Distributed Computing*, 13(3):155–186, 2000.
17. M. L. Puterman. *Markov Decision Processes*. J. Wiley and Sons, 1994.
18. R. Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems*. PhD thesis, MIT, 1995. Technical Report MIT/LCS/TR-676.
19. F. Somenzi. Colorado university decision diagram package. `http://vlsi.colorado.edu/pub/`, 1998.
20. M.I.A. Stoelinga. Fun with FireWire: Experiments with verifying the IEEE1394 root contention protocol. In *Formal Aspects of Computing*, 2002.
21. 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.