

Efficient algorithms for solving stochastic games on finite graphs

Nitesh Kumar and Shashi Mittal
Supervisor – Prof. Anil Seth

Department of Computer Science and Engineering,
Indian Institute of Technology, Kanpur

Abstract. Games are an effective modeling tool for systems involving interaction with the environment. Situations where there are two players having conflicting goals as well as a random agent are best depicted by stochastic games. This paper deals mainly with stochastic games with ω -regular objectives, specifically Büchi winning condition. We present a sub-quadratic algorithm for solving stochastic games with such winning conditions for a graph with n vertices and $O(n)$ edges.

Markov Decision Processes(MDPs) are games involving just the system and a random agent. We present a new algorithm for solving such games, which in the worst case has a quadratic running time, but works well on graphs with “few” random nodes.

1 Introduction

Games are a useful model for modeling open systems. This is because the interaction between system and environment can be easily represented as interaction between two players. The two players choose their moves after each round (either simultaneously or in turns) which determines the next state of the game. Such games are played on a finite state space (which is usually represented as a graph). We will generally deal with infinite games, i.e. games in which the players keep playing indefinitely. The graph on which the game is played can be finite or infinite, but here we will discuss games on finite graphs only. A nice survey on the role of application of games in logic and verification can be found in [1].

Winning condition in infinite games are generally ω -regular (i.e. defined in terms of properties of strings of infinite length), e.g. **safety game** (a player should always remain in a given subset of states), **reachability** (a player should reach a given subset of states at least once) and so on. The goal of the player (usually the player 1) is to satisfy these winning conditions and of the other player (usually the player 2) is to stop player 1 from satisfying the condition. The problem of solving the game, is then, to find from which starting states the player 1 has a winning strategy, i.e. a strategy to satisfy the winning condition irrespective of the moves of the player 2.

Games can be classified according to the following parameters:

- Number of players: $1\frac{1}{2}$ player (Markov Decision Process or MDPs), 2 players, $2\frac{1}{2}$ players (Stochastic games) [2]
- *Concurrent* (i.e. next state is decided by the choice of moves of both the players) [3, 4], turn based or *simple* (the players take turns in deciding the next state of the game)
- Games on finite graphs ([3, 4, 2] or games on infinite graphs ([5, 6])
- Information available to the players - *perfect information games* (players have full knowledge of the game graph), *Semi-perfect information games* (players have knowledge of only a finite part of the graph in any state)[7]
- Winning conditions - ω regular conditions: finite objectives such as reachability or safety, or infinite conditions such as Büchi, co-Büchi or Parity objective (the winning conditions will be explained in detail later)
- Winning modes - Sure (player can win the game deterministically) , Almost Sure(player can win the game with probability 1) and Limit Sure (player can win the game with probability arbitrarily close to 1); see [4] for more details on these winning conditions.

Here, we present algorithms for solving the $2\frac{1}{2}$ and $1\frac{1}{2}$ games on finite directed graphs for the almost-sure Büchi winning condition. In Büchi winning condition, we are given a subset T of the vertices, V and the aim is to visit atleast one of the vertices in the set T infinitely many times. The game is a turn based. In $1\frac{1}{2}$ games, each vertex belongs to either the *even* player or the *random* player. In $2\frac{1}{2}$ player game, there is also an *odd* player in addition to these two players. From a random node, the game can proceed to any of the successor vertices with equal probability (and hence the $\frac{1}{2}$ player). The problem is to find the set of strategies from which the player even can win the game with probability 1.

de Alfaro and Henzinger[4] gave a quadratic time algorithm for solving concurrent games with Büchi winning conditions. In an interesting result, Jurdziński et al[8] showed that any concurrent game with Büchi or co-Büchi winning condition can be reduced a turn based $2\frac{1}{2}$ player game with the same winning conditions in linear time. Hence, a sub-quadratic time algorithm for $2\frac{1}{2}$ player games will automatically imply a sub-quadratic algorithm for the concurrent games.

Chatterjee et al[2] showed that a $2\frac{1}{2}$ games with parity winning condition (of which the Büchi winning condition is a special case) can be reduced to a 2-player game. However, the reduction yields a graph which is quadratic in size of the input graph. They also gave the first sub-quadratic time algorithm for solving 2 player game with Büchi winning condition. For a graph with n vertices and m edges, their algorithm takes $O(n^2/\log n)$ time when the graph is binary (i.e. there are atmost two outgoing edges from a vertex). Since any game graph with n vertices and m vertices can be reduced to a binary graph with $O(m)$ vertices and $O(m)$ edges in linear time, therefore the algorithm solves the 2-player game in sub-quadratic time for any game graph. In the same paper, they give

an $O(m\sqrt{m})$ algorithm for solving the $1\frac{1}{2}$ game with Büchi winning condition.

Since the $1\frac{1}{2}$ and 2 player games can be solved in sub-quadratic times, it was widely believed that the $2\frac{1}{2}$ player games can also be solved in sub-quadratic time. In this paper, we achieve this. Our algorithm runs in $O(n^2/\log n)$ time. We present this algorithm in the next section. We also present an alternative algorithm for solving $1\frac{1}{2}$ games, which in the worst case has quadratic running time but is expected to have better running time when there are “fewer” random nodes in the graph.

2 Quadratic Algorithm for Simple Stochastic Games

A simple stochastic game (a turn-based $2\frac{1}{2}$ player game) consists of a game graph G and 3 players, Even(\square), Odd(\diamond) and Random(\circ). The game graph G consists of a directed graph (V, E) and a partition $(V_\square, V_\diamond, V_\circ)$. We will assume that the graph has $O(n)$ nodes and $O(n)$ vertices. Every node has at least one outgoing edge. Only Even(\square) player can move from edges in V_\square , Odd(\diamond) player can move from edges in V_\diamond and so on. The game starts from any particular node and the player who owns that node moves a token to any of the successor vertices. The edges going out from a random node (node in V_\circ) are assigned some probabilities which add up to 1. The random player chooses one of the successor nodes according to the probability assigned to the corresponding outgoing edge. The game proceeds by moving the token from node to node thus, forming an infinite path. The winning conditions for the players e.g. Reachability, Büchi etc. are defined over this infinite path formed. The set of all infinite paths is denoted by Ω .

We will first discuss an algorithm to solve such games with Büchi winning condition and almost-sure winning mode. The almost-sure winning mode means that from any node in the solution set and for all strategies of the Odd(\diamond) player, the Even(\square) player has a strategy to win the game with probability 1. Precisely, if ϕ is the winning condition and $[\phi]$ is the set of infinite paths that satisfy ϕ , then $s \in \text{Almost}(\phi)$ iff $\exists \pi_\square \in \Pi_\square. \forall \pi_\diamond \in \Pi_\diamond. Pr_s^{\pi_\square \pi_\diamond}([\phi]) = 1$. where Π_\square is set of strategies of player Even and Π_\diamond is set of strategies of player Odd.

The Büchi winning condition is an ω -regular condition which specifies a set of the nodes of the graph as the target set $T (\subseteq V)$. The objective of the player Even is to visit this set infinitely often. Formally, $\text{Büchi}(T) = \{\langle v_0, v_1, v_2, \dots \rangle \in \Omega : v_k \in T \text{ for infinitely many } k \geq 0\}$.

Before proceeding, we define the set of nodes $\text{DualAttr}_{\square, \circ}(S, V)$ for $S \subseteq V$ as the set of nodes in V from which the Even and the Random player together can force the game to reach the set S . $\text{DualAttr}_{\square, \circ}(S, V)$ can be evaluated inductively as

$$\begin{aligned}
S_0 &= S \\
S_{m+1} &= S_m \cup \{v \in V_{\square} \cup V_{\circ} : (v, u) \in E \text{ for some } u \in S_m\} \\
&\quad \cup \{v \in V_{\diamond} : u \in S_m \text{ for all } (v, u) \in E\} \\
DualAttr_{\square, \circ}(S, V) &= \bigcup_k S_m
\end{aligned}$$

In a similar manner, $DualAttr_{\diamond, \circ}(S, V)$ can also be defined. One can also evaluate $DualAttr_{\square, \circ}(S, V)$ by first changing all random nodes to Even and then evaluating the attractor set $Attr_{\square}(S, V)$ in the normal fashion. The algorithm for finding attractor can be implemented in such a way that no edge is visited twice giving an $O(n)$ complexity.

2.1 The Algorithm

The algorithm works by first finding the vertices from which Even and Random player together can force the game to the target set, R_i . The rest of vertices, Tr_i serve as a trap for the Even and Random players and hence, these nodes are eliminated. Now, the nodes from which there is a finite probability to reach the eliminated nodes should also be eliminated. So, the $DualAttr_{\diamond, \circ}(Tr_i, V_i)$ is also eliminated. This algorithm then iterates on the game graph left i.e. V_{i+1} . The algorithm stops when no more nodes can be eliminated and the game graph left is the solution set.

Algorithm 1 Quadratic Algorithm for $2\frac{1}{2}$ -player Büchi Games

Input: $2\frac{1}{2}$ player Büchi game graph G, Target Set T

Output: Set of vertices W_{\square} from where Even player can win

1. $V_0 \leftarrow V, i \leftarrow 0$
 2. do
 - 2.1 $R_i \leftarrow DualAttr_{\square, \circ}(T \cap V_i, V_i)$
 - 2.2 $Tr_i \leftarrow V_i \setminus R_i$
 - 2.3 $W_i \leftarrow DualAttr_{\diamond, \circ}(Tr_i, V_i)$
 - 2.4 $V_{i+1} \leftarrow V_i \setminus W_i$
 - 2.5 $i \leftarrow i + 1$
 - while($V_i \neq V_{i-1}$)
 3. $W_{\square} \leftarrow V_i$
-

2.2 Termination and Correctness

In every iteration, at least one node is eliminated (otherwise $V_i \neq V_{i-1}$ and the algorithm will terminate). Also, there are a total of $O(n)$ nodes. So, there can't be more than $O(n)$ iterations and the algorithm will eventually terminate.

We first prove that after the i^{th} iteration, only those nodes are retained from

where the game will reach $T \cap V_{i-1}$ with probability 1 and also, will not reach any node with finite probability from where one can't visit $T \cap V_{i-1}$ with probability 1. (i.e. eliminated nodes)

Lemma 1. *All nodes that are eliminated in i^{th} iteration don't satisfy the above criteria*

Proof. The nodes eliminated in the i^{th} iteration are

- Trap Sets for Even and Random The Odd player will force the game to remain in the trap set and so, $T \cap V_{i-1}$ can't be visited with probability 1.
- DualAttractor of Trap Sets From these nodes, the Odd and the Random player can force the game to go to trap sets (eliminated sets). So, there exists a finite probability to reach an eliminated node.

Lemma 2. *All nodes retained after i^{th} iteration satisfy the above criteria.*

Proof. All retained nodes are in $Dual - Attractor_{\square, \circ}(T \cap V_{i-1}, V_{i-1})$ and so, there is a finite probability to reach $T \cap V_{i-1}$ from each of these nodes. Let the minimum of these probabilities be p . Now, if the game is at a particular node v_1 in this set, it either goes to $T \cap V_{i-1}$ with probability p or goes to some other node v_2 with probability $1-p$. Since, v_2 is also in the set of retained nodes, again there is at least a probability of p to reach the target nodes. Continuing in this way, the total probability to reach the target nodes is $p + p(1-p) + p(1-p)^2 + \dots = 1$. Also, one can't reach any eliminated node from this set because this set is a trap for Odd and Random players.

Once we are not able to eliminate any nodes, $V_{i-1} = V_i$ and so, in the remaining set of nodes, the game will reach the target set $T \cap V_i$ from all nodes with probability 1 and will never go out of V_i . So, the game will continually visit $T \cap V_i$ infinitely often satisfying the Büchi condition.

2.3 Analysis of the Algorithm

Each iteration of the algorithm involves two *DualAttractor* finding sub-routines which take $O(n)$ time. Since in every iteration at least one node is eliminated, the total number of iterations can't exceed $O(n)$. So, the time complexity of the algorithm is $O(n^2)$.

3 Sub-quadratic algorithm for solving simple stochastic games

Using the above quadratic algorithm, we design a more efficient algorithm that solves a Büchi game in $O(n^2/\log n)$ time. The algorithm uses a *forward search* technique. The algorithm differs from the earlier algorithm in the method of finding trap sets for player Even and Random. In the earlier case, we found the $DualAttr_{\square, \circ}$ and then took its complement to get the trap set. In this algorithm,

we try to isolate small subgraphs and look for trap sets in them. This forward exploration of trap sets is cheap improving the complexity of the overall algorithm.

In the rest of the section we use the following notations for a graph $G = (V, E)$ and a set $S \subseteq V$ of vertices. We define $succ(v, G) = \{u \in V | (v, u) \in E\}$ for the set of immediate successors of vertex v . We define $In(R, S) = \{(v, u) \in E | v \in R \text{ and } u \in S\}$ to be the set of edges that enter R from S (R and S are disjoint). Also, we define $Source(R, S) = \{v \in V | (v, u) \in In(R, S) \text{ for some } u\}$ is the set of vertices in R from which one can enter S.

Algorithm 2 Sub-Quadratic Algorithm for $2\frac{1}{2}$ -player Büchi Games

Input: $2\frac{1}{2}$ player Büchi game graph G, Target Set T

Output: Set of vertices W_{\square} from where Even player can win

1. $V_0 = V, W_0 \leftarrow \Phi, i \leftarrow 0$
 2. do
 - 2.1 if $|In(W_i, V_i)| \geq k$
 - 2.1.1 $R_i \leftarrow DualAttr_{\square, \circ}(T \cup V_i, V_i)$
 - 2.1.2 $Tr_i \leftarrow V_i \setminus R_i$
 - 2.1.3 $W_{i+1} \leftarrow DualAttr_{\diamond, \circ}(Tr_i, V_i)$
 - 2.1.4 $V_{i+1} \leftarrow V_i \setminus W_i$
 - 2.1.5 $i \leftarrow i + 1$
 - 2.2 else $|In(W_i, V_i)| < k$
 - 2.2.1 $Tr_i \leftarrow \Phi$
 - 2.2.2 for each $v \in Source(W_i, V_i)$
 - 2.2.2.1 $T_v \leftarrow$ Set of nodes in BFS of depth $\log l$ from v
 - 2.2.2.2 $L_v \leftarrow$ Set of nodes at depth $\log l$
 - 2.2.2.3 $N'_v \leftarrow \{v \in V_{\diamond} \cup L_v | Succ(v, G) \cup V_i \cup R_v = \Phi\} \cup (V_{\square} \cap L_v)$
 - 2.2.2.4 $N_v \leftarrow N'_v \cup (T \cup V_i \cup T_v)$
 - 2.2.2.5 $R_v \leftarrow DualAttr_{\square, \circ}(N_v, T_v)$
 - 2.2.2.6 $Tr_i \leftarrow Tr_i \cup (T_v \setminus R_v)$
 - 2.2.3 if $Tr_i \neq \Phi$ then
 - 2.2.3.1 $W_{i+1} \leftarrow DualAttr_{\diamond, \circ}(Tr_i, V_i)$
 - 2.2.3.2 $V_{i+1} \leftarrow V_i \setminus W_i$
 - 2.2.3.3 $i \leftarrow i + 1$
 - 2.2.4 else
 - 2.2.4.1 $R_i \leftarrow DualAttr_{\square, \circ}(T \cup V_i, V_i)$
 - 2.2.4.2 $Tr_i \leftarrow V_i \setminus R_i$
 - 2.2.4.3 $W_{i+1} \leftarrow DualAttr_{\diamond, \circ}(Tr_i, V_i)$
 - 2.2.4.4 $V_{i+1} \leftarrow V_i \setminus W_i$
 - 2.2.4.5 $i \leftarrow i + 1$
 3. while $(V_i \neq V_{i-1})$
 4. $W_{\square} = V_i$
-

3.1 The Algorithm

This algorithm differs from the earlier algorithm in how the eliminations are done during the iterations. During an iteration, if the number of edges entering the set of vertices eliminated during last iteration is at least as big as a constant k , then we run an iteration of the earlier algorithm. Otherwise, i.e., if this number is smaller than k , then let S be the set of sources of edges that enter the set of vertices eliminated in the previous iteration. From each vertex in S , we create a BFS tree of depth $\log l$. If there is a trap set of size smaller than $\log l$ which contains this vertex, it will present in the BFS tree obtained too. So, we look for all trap sets for Even and Random players in this small subgraph(BFS tree). Here, the target set comprises of original target nodes in the BFS tree as well as nodes at the edge of the BFS tree from where the Even or Random player can escape. All such small trap sets are found and their union is now considered as the trap set. $DualAttr_{\diamond, \circ}$ of this trap set is found and is eliminated too. If no trap set is found by the forward search, one iteration of the original algorithm is executed. The algorithm stops when no more nodes can be eliminated.

One can show that in the original algorithm all trap sets eliminated have at least one node in the set of vertices eliminated in the previous iteration, say S . So, if a trap set R is of size smaller than $\log l$ and contains the vertex $v \in S$, it will be obtained as the trap set while operating on the BFS tree rooted at v . The key to the sub-quadratic bound is the fact that if the classical iteration is carried out in Step 2.2.4, it will find a trap set of size larger than $\log l$.

3.2 Termination and Correctness

In every iteration a trap set is eliminated that has at least one node. If no trap set is found, the algorithm terminates. Since there are only $O(n)$ nodes, only $O(n)$ iterations are possible. Also, since the graph is finite, the algorithm has to eventually terminate.

Let W_i and W'_i be the sets eliminated in i_{th} iteration of the earlier and the present algorithms. Also, let W_{\diamond} and W'_{\diamond} be the union of all eliminated sets of quadratic and sub-quadratic algorithms respectively.

Lemma 3. W'_i computed in any iteration of improved algorithm is a subset of W_{\diamond} , i.e. $W'_i \subseteq W_{\diamond}$.

Proof. We prove it by induction over i .

Base Case: $i = 0$. $W'_0 = \emptyset \subseteq W_{\diamond}$.

Induction Step: Let $W'_i \subseteq W_{\diamond}$

The next trap set is eliminated in either of these three steps

1. If Step 2.1 is executed, then $W'_{i+1} \subseteq W_{\diamond}$ by the correctness of the earlier algorithm.

2. If Step 2.2.2 and 2.2.3 are executed, the sets eliminated are trap sets in T_v . For every vertex v in this set, $\text{succ}(v, G) \in T_v$ and so, this serves as a trap set for the whole graph. Union of all such trap sets is also a trap set and from such set the game can never reach the target set. So, by the correctness of the earlier algorithm, this trap set would also have been eliminated, i.e. $W'_{i+1} \subseteq W_\diamond$.
3. If the trap set is eliminated in Step 2.2.4, then $W'_{i+1} \subseteq W_\diamond$ by correctness of earlier algorithm.

Since W'_\diamond is union of all W'_i 's, so $W'_\diamond \subseteq W_\diamond$.

The other inclusion $W_\diamond \subseteq W'_\diamond$ holds because the termination condition implies that no more vertices can be eliminated and so, the last iteration will be an iteration of the quadratic algorithm.

3.3 Analysis of the algorithm

Since the step 2.1.1 is executed only if there are k or more outgoing edges from the last eliminated set and there are only $O(n)$ nodes, so this step is executed $O(n/k)$ times. Each iteration takes $O(n)$ time and so, this step will take $O(n^2/k)$ time in all. Step 2.2.2 is executed for each of $O(k)$ vertices in an iteration. BFS of depth $\log l$ takes $O(l)$ time. So, each iteration takes $O(kl)$ time. There can be a maximum of $O(n)$ iterations and so, this step in all takes $O(kln)$ time. Step 2.2.4 is executed only if the trap set is larger than $\log l$ which can happen only $O(n/\log l)$ times. So, this step can take $O(n^2/\log l)$ in the whole algorithm.

Total work amounts to $O(n^2/k + kln + n^2/\log l)$. Taking $l = n^\epsilon$ with $0 < \epsilon < 1$ and $k = \log n$, we get the total complexity to be $O(n^2/\log n)$.

4 An alternative algorithm for solving $1\frac{1}{2}$ games

In this section we present an alternative algorithm for solving Markov Decision Process. The idea behind the algorithm is to first resolve the graph into strongly connected components, and then find which nodes are winning for player even/random in each of the strongly connected components.

4.1 The algorithm

For $S \subseteq V$, let $\text{Out}(S, V) = \{v \in V : v \notin S \text{ and } \exists w \in S (w, v) \in E\}$. Thus, $\text{Out}(S, V)$ is the subset of vertices for which there is an outgoing edge from any one of the vertices in the set S .

The algorithm works by first finding the strongly connected components of the graph, and then working on each strongly connected component in a backward fashion. During the execution of the algorithm, for any strongly connected component C , following are the different possibilities:

Algorithm Solve-BüchiMDP

Input: $1\frac{1}{2}$ player Büchi game (G, T) . **Output:** W_{\circlearrowleft} and $W_{\square} = V \setminus W_{\circlearrowleft}$.

1. Resolve the graph into strongly connected components.
 2. Topologically sort the acyclic components of the graph.
Let the strongly connected components be V_1, \dots, V_k ; there is no edge from V_i to V_j for $i < j$.
 3. $W_{\circlearrowleft} \leftarrow \phi$, $W_{\square} \leftarrow \phi$
 4. for $i := 1$ to k do
 - 4.1 if $Out(V_i, V) = \phi$
 - 4.1.1 if $V_i \cap T = \phi$ then $W_{\circlearrowleft} \leftarrow W_{\circlearrowleft} \cup V_i$
 - 4.1.2 else $W_{\square} \leftarrow W_{\square} \cup V_i$
 - 4.2 else if $Out(V_i \cap V_{\circlearrowleft}, V) = \phi$
 - 4.2.1 if $V_i \cap T \neq \phi$ or $Out(V_i \cap V_{\square}, V) \cap W_{\square} \neq \phi$ then $W_{\square} \leftarrow W_{\square} \cup V_i$
 - 4.2.2 else $W_{\circlearrowleft} \leftarrow W_{\circlearrowleft} \cup V_i$
 - 4.3 else if $Out(V_i \cap V_{\circlearrowleft}, V) \subseteq W_{\square}$ then $W_{\square} \leftarrow W_{\square} \cup V_i$
 - 4.4 else
 - 4.4.1 $\tilde{V}_{\circlearrowleft} \leftarrow \{v \in V_i \cap V_{\circlearrowleft} : \exists w \in W_{\circlearrowleft} \text{ such that } (v, w) \in E\}$
 - 4.4.2 $\tilde{V}_{\square} \leftarrow \{v \in V_i \cap V_{\square} : \exists w \in W_{\square} \text{ such that } (v, w) \in E\}$
 - 4.4.3 $\hat{V}_{\circlearrowleft} \leftarrow Attr_{\circlearrowleft}(\tilde{V}_{\circlearrowleft}, V_i)$
 - 4.4.4 $\hat{V}_{\square} \leftarrow Attr_{\square}(\tilde{V}_{\square}, V_i)$
 - 4.4.5 $W_{\circlearrowleft} \leftarrow W_{\circlearrowleft} \cup \hat{V}_{\circlearrowleft}$
 - 4.4.6 $W_{\square} \leftarrow W_{\square} \cup \hat{V}_{\square}$
 - 4.4.7 *Solve-BüchiMDP* $(V_i \setminus (\hat{V}_{\circlearrowleft} \cup \hat{V}_{\square}))$
-

1. There are no outgoing edges from the component C . If $C \cap T = \phi$ then clearly player even cannot win the game from any vertex in C . Otherwise, player even has a strategy to visit the target vertices in C infinitely many times, and in that case even can win from any vertex in C .
2. There are outgoing edges from C from only the vertices of even player. If there is atleast one vertex, say \tilde{v} , such that there is an edge from \tilde{v} to a vertex which is already in W_{\square} , then player even can win the game by reaching \tilde{v} . This he can do from any other vertex in C with probability 1. Hence, all the vertices in C are winning for even. Also, if there is atleast one target vertex in C , then player even can force a win by argument similar to that in 1. Otherwise, player even cannot win from any of the vertex in C .
3. If all the outgoing edges from random nodes in C are to vertices already in W_{\square} , then player even can win the game from any vertex in C by reaching any such random vertex: With probability 1, a transition will ultimately occur from such a random node to a vertex in W_{\square} , and hence player even can win the game from any vertex in C .
4. Otherwise, there is atleast one random vertex outgoing edge from C to a vertex in W_{\circlearrowleft} . Let $\tilde{V}_{\circlearrowleft}$ be the collection of all such random vertices. Then the attractor set of $\tilde{V}_{\circlearrowleft}$ with respect to random player in C is winning for player random. Similarly, let \tilde{V}_{\square} be the collection of all even vertices from which there is atleast one outgoing edge to a vertex in W_{\square} , the attractor of this set with respect to player even in C is winning for player even. For

rest of the vertices, we run this algorithm recursively to find which nodes are winning for player even and random.

Lemma 4. *Algorithm Solve-BüchiMDP correctly finds the sets W_{\circlearrowleft} and W_{\square} .*

Proof. The proof is by induction on the number of vertices in the input graph. For $|V| = 1$, the correctness of the algorithm is trivial. Suppose the algorithm correctly finds the winning set of vertices for the two players for $|V| = 1, \dots, n-1$. Consider the run of the algorithm for the input in which $|V| = n$. It can be seen that the algorithm correctly finds the winning sets for the cases 1, 2 and 3 mentioned above. For the case 4, the set $\hat{V}_{\circlearrowleft} = Attr_{\circlearrowleft}(\tilde{V}_{\circlearrowleft}, V_i)$ is winning for player Odd, and the set $\hat{V}_{\square} = Attr_{\square}(\tilde{V}_{\square}, V_i)$ is winning for player even. The set $V_i \setminus (\hat{V}_{\circlearrowleft} \cup \hat{V}_{\square})$ has number of vertices less than n , and therefore by induction hypothesis the algorithm correctly finds the winning sets in this particular subset of V_i . Hence, by induction, the algorithm works correctly for all input graphs.

4.2 Analysis of the algorithm

Steps 1 through 4.3 take $O(m+n)$ time, the only step which may take quadratic time is step 4.4. This case arises only when there is atleast one random node in the strongly connected component with an outgoing edge to a vertex in W_{\circlearrowleft} . The number of recursive calls to the algorithm is therefore atmost r , where r is the number of random nodes in the graph. In each recursive call, the time taken is in $O(m+n)$, and hence the worst case running time of the algorithm is $O(r(m+n))$.

The algorithm, therefore, can have a running time that is quadratic in the input size. However, if the number of random nodes in the graph is small, then the algorithm achieves a better running time bound. For example, if r is in $O(\log n)$, then the running time is $O((m+n)\log n)$, which is better than the $O(m\sqrt{m})$ algorithm given by Chatterjee et al[2].

4.3 Improving the algorithm

The worst case running time of the algorithm can be furthered improved to $O(m\sqrt{m})$ by using the idea of backward search proposed by Chatterjee et al in [2]. In step 4.4, instead of recursively calling the algorithm, one can use the algorithm of Chatterjee et al to find the vertices winning for player even and random. This algorithm will not improve the asymptotic bound of Chatterjee's algorithm, however it will reduce the constant factor associated with the running time.

5 Future work

Another winning mode called probably-achieving winning mode can be defined which means that from any node in the solution set and for all strategies of

the Odd(\diamond) player, the Even(\square) player has a strategy to win the game at least with probability p where p is a specified constant. Formally, if ϕ is the winning condition and $[\phi]$ is the set of infinite paths satisfying ϕ , then $s \in \text{Probably}_p(\phi)$ iff $\exists \pi_{\square} \in \Pi_{\square}. \forall \pi_{\diamond} \in \Pi_{\diamond}. Pr_s^{\pi_{\square} \pi_{\diamond}}([\phi]) \geq p$. where Π_{\square} is set of strategies of player Even and Π_{\diamond} is set of strategies of player Odd. No work has been done so far on finding algorithms for games with Reachability and Büchi winning conditions and probably-achieving winning mode.

It will be interesting to find whether algorithms for Büchi MDPs can be improved. More specifically, does a linear time algorithm exist for them.

The corresponding problems in infinite graphs pose different challenges. There has been almost no work on this topic. It would be interesting to find whether these algorithms translate to infinite graphs too.

Acknowledgments

We are grateful to our advisor, Prof. Anil Seth for his guidance and encouragement throughout the project work. We thank Prof. Thomas Henzinger for suggesting us this topic to work on as our B. Tech. project. We are also thankful to Krishnendu Chatterjee for several useful discussions, in particular for clarifying our doubts on [2] and [9].

References

- [1] Walukiewicz, I.: From logic to games. In: FSTTCS. (2005) 79–91
- [2] Chatterjee, K., Jurdzinski, M., Henzinger, T.A.: Simple stochastic parity games. In: CSL. (2003) 100–113
- [3] de Alfaro, L., Henzinger, T.A., Kupferman, O.: Concurrent reachability games. In: FOCS. (1998) 564–575
- [4] de Alfaro, L., Henzinger, T.A.: Concurrent omega-regular games. In: LICS. (2000) 141–154
- [5] Walukiewicz, I.: Pushdown processes: Games and model-checking. *Inf. Comput.* **164**(2) (2001) 234–263
- [6] Gimbert, H.: Parity and exploration games on infinite graphs. In: CSL. (2004) 56–70
- [7] Chatterjee, K., Henzinger, T.A.: Semiperfect-information games. In: FSTTCS. (2005) 1–18
- [8] Jurdzinski, M., Kupferman, O., Henzinger, T.A.: Trading probability for fairness. In: CSL. (2002) 292–305
- [9] Chatterjee, K., Jurdzinski, M., Henzinger, T.A.: Quantitative stochastic parity games. In: SODA. (2004) 121–130
- [10] Shapley, L.: Stochastic games. In: Proceedings of National Academy of Sciences. Volume 39. (1953) 1095–1110
- [11] Thomas, W.: On the synthesis of strategies in infinite games. In: STACS. (1995) 1–13
- [12] Jurdzinski, M.: Small progress measures for solving parity games. In: STACS. (2000) 290–301