New Results on Simple Stochastic Games

Decheng Dai*[1] and Rong Ge[2]

[1] Tsinghua University, ddc02@mails.tsinghua.edu.cn
[2] Princeton University, rongge@cs.princeton.edu

Abstract. We study the problem of solving simple stochastic games, and give both an interesting new algorithm and a hardness result. We show a reduction from fine approximation of simple stochastic games to coarse approximation of a polynomial sized game, which can be viewed as an evidence showing the hardness to approximate the value of simple stochastic games. We also present a randomized algorithm that runs in $\tilde{O}(\sqrt{|V_{\mathrm{R}}|!})$ time, where $|V_{\mathrm{R}}|$ is the number of RANDOM vertices and $\tilde{O}$ ignores polynomial terms. This algorithm is the fastest known algorithm when $|V_{\mathrm{R}}| = \omega(\log n)$ and $|V_{\mathrm{R}}| = o(\sqrt{\min|V_{\min}|, |V_{\max}|})$ and it works for general (non-stopping) simple stochastic games.

1 Introduction

1.1 Simple Stochastic Games

Simple stochastic games are games played by two players on a graph, it is a restricted version of general stochastic games introduced by Shapley [1]. In a simple stochastic game, two players (MAX and MIN) move a pebble along directed edges in a graph. The vertices in the graph can have one of the three labels: MAX, MIN or RANDOM. If the pebble is on a vertex labeled MAX(or MIN), then MAX(or MIN) player decides through which out going edge the pebble should move; if the pebble is on a vertex labeled RANDOM, then the pebble moves along a randomly chosen edge. The graph also has a special vertex called the "1-sink". The MAX player wins if and only if the pebble is moved to 1-sink.

SSGs have many interesting applications. In complexity theory, SSGs are used in the analysis of space bounded computations with alternations and randomness [2].In practice, SSGs are used to model reactive systems. In such systems, RANDOM vertices are used to model stochastic environmental changes, MAX vertices are used to model adversary or arbitrary behaviors, MIN vertices are used to model choices of the system. The 1-sink vertex represents a failure. The goal of the system is thus minimizing the probability of failure (reaching 1-sink vertex).

Finding the optimal strategies for SSGs has been an interesting open problem for a long time. A lot of algorithms have been purposed. Condon [2] proved the

* Supported by the National Natural Science Foundation of China Grant 60553001 and the National Basic Research Program of China Grant 2007CB807900, 2007CB807901.

decision version of SSG is in $\mathbf{NP} \cap \mathbf{coNP}$, and later in 1993, she showed several iterative algorithms for SSG in [3], but all of these algorithms require exponential time. She also suggested an approximation version of SSG problem, but there are no polynomial time algorithms known. Our gap amplification result gives an evidence on why the approximation problem is also difficult. Ludwig gave a sub-exponential ($\tilde{O}(2^{\sqrt{n}})$, $\tilde{O}$ hides polynomial terms) time randomized algorithm for SSGs in [4], which uses local search techniques. Somla [5] purposed a new iterative algorithm in 2004 which might be better than previous algorithms, however there's no evidence that shows the algorithm runs in polynomial time. Recently, Gimbert and Horn [6] presented a new non-iterative algorithm that runs in time $\tilde{O}(|V_{\mathrm{R}}!|)$. This highlights one of the main reasons the problem has exponential complexity: the existence of random vertices.

1.2 Our Results

In this paper, we investigate the SSG problem in both hardness and algorithmic aspects. On the hardness side, we show that a coarse approximation of SSGs is as hard as a fine approximation. This is done by constructing a new game G' from a game G, such that G' has polynomial size and a coarse approximation to G' would give a fine approximation of game G. Viewed pessimistically this can be an evidence that shows it is hard to even approximate SSGs; viewed optimistically, this may give hope of deriving a good approximation algorithm for SSGs.

At the algorithmic side, we present an algorithm based on the algorithm of Gimbert and Horn [6]. They considered a set of strategies called **f**-strategies, and showed at least one of **f**-strategies is optimal. However they were not able to distinguish "good" **f**-strategies and "bad" **f**-strategies. By finding a way to evaluate the "correctness" of **f**-strategies, we are able to apply local search algorithms to find the optimal **f**-strategy, and reduce the running time to $\tilde{O}(\sqrt{|V_{\mathrm{R}}|!})$. Our algorithm is the fastest known randomized algorithm for solving SSGs when $|V_{\mathrm{R}}| = \omega(\log n)$ and $|V_{\mathrm{R}}| = o(\sqrt{\min\{|V_{\max}|, |V_{\min}|\}})$.

In Sect.2 we give definitions for Simple Stochastic Games and strategies. Then we describe the reduction from fine approximation to coarse approximation in Sect.3. After that, we give a brief introduction to **f**-strategies and then present our algorithm.

2 Basic Definitions

There are many variations of SSGs, we define the game formally as follows

Definition 1 (Simple Stochastic Games). *A simple stochastic game is specified by a directed graph $G =< V, E >$ and a starting vertex $v_{start} \in V$. Each vertex $v \in V$ has 2 outgoing edges and a label (MAX, MIN or RANDOM). V_{min}, V_{max}, V_R are the sets of vertices with label MIN, MAX and RANDOM respectively. There's a special vertex v_1 (the 1-sink) in the graph.*

Initially the pebble is at v_{start}. If the pebble is at a MAX/MIN vertex, then the corresponding player moves the pebble along one of the outgoing edges. If the pebble is at a RANDOM vertex, then the pebble moves along a random outgoing edge (both edges are chosen with probability 1/2). If the pebble reaches v_1 then MAX player wins, otherwise MIN player wins.

Solving SSGs means calculating the winning probabilities for the players if they all follow optimal strategy. Informally, the strategy of a player decides which edge should the pebble follow in the game. Although a strategy can decide the edge by considering history or using random coins, it's well known that *positional* optimal strategies exist for simple stochastic games ([1, 2]). A positional strategy makes the decision only by the current position of the pebble. Formally, a positional strategy for MAX player α is a function from $V_{\max}$ to V, for any vertex $v \in V$, $(v, \alpha(v))$ is an edge and it is the outgoing edge that the MAX player would choose if the pebble is currently at vertex v. Similarly, a positional strategy for MIN player β is a function from $V_{\min}$ to V. From now on when we mention strategy we mean positional strategy.We define the value of a vertex to be the winning probability of the MAX player if initially the pebble is at this vertex, and denote this by $\mathsf{val}(v)$, the value of the game is $\mathsf{val}(v_{\text{start}})$. When it's not clear which game we are talking about, we use $\mathsf{val}[G](v)$ to specify the value of v in game G.

In simple stochastic games, MIN player wins the game by forcing the pebble to move infinitely many steps without reaching the 1-sink. Sometimes it's easier to consider the situation that the game has two sinks: a 0-sink and a 1-sink. The game guarantees no matter what strategies the players use, with probability 1 the game will reach one of the sinks in finitely many steps. The goal of MAX player is to reach the 1-sink and the goal of MIN player is to reach the 0-sink. This variation of SSG is called stopping simple stochastic games (stopping-SSG). Condon showed in [2] that any SSG can be converted to a stopping-SSG in polynomial time while the change in the value of the game is exponentially small.

3 Coarse Approximation is As Hard As Fine Approximation

Since no polynomial time algorithms has been discovered for exactly solving SSGs, Condon[2] purposed the following "approximation" version of the problem. Consider the following sets,

$$L_{yes} = \{G : \text{ the value of } G \text{ is at least } \frac{1}{2} + \epsilon\} \ ,$$

$$L_{no} = \{G : \text{ the value of } G \text{ is at most } \frac{1}{2} - \epsilon\} \ .$$

An ϵ-gap SSG decision problem is to determine whether G is in L_{yes} or L_{no} given it is in one of them.Intuitively it might seem for some large enough ϵ

this problem is easy to solve. However, we give a gap amplification reduction showing that when enlarging ϵ from $(1/\operatorname{poly}(n))$ to $(1/2 - e^{-n^\rho})$ for any $\rho < 1$, the problem does not become easier. This reduction is analogue to the hardness amplification results for clique and chromatic number problem.

Theorem 1. *For any fixed constant $0 < \rho < 1$ and $c > 0$, if the $(1/2 - e^{-n^\rho})$-gap SSG decision problem is in* $\mathbf{P}$*, then the (n^{-c})-gap SSG decision problem is in* $\mathbf{P}$*.*

Proof. First we prove the theorem for stopping SSG.

Now let's assume $G = \langle V, E\rangle$ is a stopping-SSG with n vertices. There are 3 special vertices in a stopping SSG: v_{start}, the starting vertex; v_1, the 1-sink vertex; v_0, the 0-sink vertex. We construct another game $G' = \langle V', E'\rangle$ of size N (which is polynomial in n) such that,

- G' has value larger than $(1 - e^{-N^\rho})$ if G has value larger than $(1/2 + n^{-c})$
- G' has value less than e^{-N^ρ} if G has value less than $(1/2 - n^{-c})$

Let $\{G_0, G_{i,j} | i \in \{0,1\}, j \in \{1, ..., K\}\}$ be $2K+1$ copies of G. We replace the out-going edges for v_0 and v_1 in each of these games to connect them together in the following way (their two outgoing edges will point to the same location, so it doesn't matter what label they have)

- Connect v_0 in G_0 to v_{start} in $G_{0,1}$, connect v_1 in G_0 to v_{start} in $G_{1,1}$.
- Connect v_1 in $G_{0,j} (1 \le j \le K)$ to v_{start} in G_0, connect v_0 in $G_{0,j} (j < K)$ to v_{start} in $G_{0,j+1}$.
- Connect v_0 in $G_{1,j} (1 \le j \le K)$ to v_{start} in G_0, connect v_1 in $G_{1,j} (j < K)$ to v_{start} in $G_{1,j+1}$.
- The starting vertex in G' is v_{start} in G_0, and the 0-sink vertex is v_0 in $G_{0,K}$ and the 1-sink vertex is v_1 in $G_{1,K}$.

In this constructed game G', the MIN(MAX) player must win G_0 and all $G_{0,j} (G_{1,j})$ to win G'. Let p to be the value in G. By induction, it is easy to prove the probability to reach v_1 in $G_{1,K}$ is $(p^K)/(p^K + (1-p)^K)$.

Let $K = n^d$ where $d = (c+1)/(1-\rho)$, then $N = (2K+1)n = O(n^{d+1})$ when $p \le 1/2 - n^{-c}$ we have,

$$\begin{aligned}\frac{p^K}{p^K + (1-p)^K} &\le \frac{(\frac{1}{2} - n^{-c})^K}{(\frac{1}{2} - n^{-c})^K + (\frac{1}{2} + n^{-c})^K} \\ &\le (1 - 2n^{-c})^K \le e^{-n^{d-c}} \\ &\le e^{-N^\rho} \quad ,\end{aligned}$$

so the value of G' is less than e^{-N^ρ} in this case. Similarly, when $p \ge 1/2 + n^{-c}$, we have the value of G' is larger than $1 - e^{-N^\rho}$. That is,

$$\begin{aligned}\mathsf{val}(G) \le \frac{1}{2} - n^{-c} &\Rightarrow \mathsf{val}(G') \le e^{-N^\rho} \quad , \\ \mathsf{val}(G) \ge \frac{1}{2} + n^{-c} &\Rightarrow \mathsf{val}(G') \ge 1 - e^{-N^\rho} \quad .\end{aligned}$$

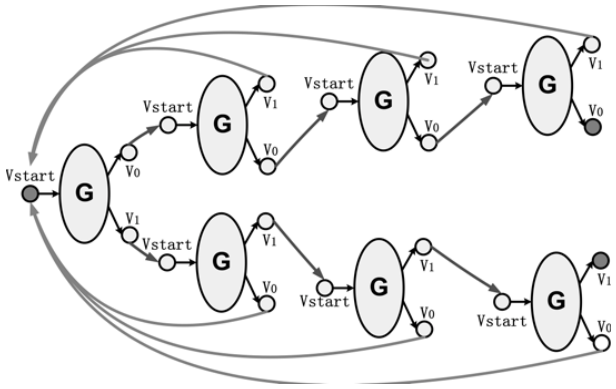

Fig. 1. An example on constructing G' from G, for $K = 3$. Every ellipse is a copy of G. Three solid vertices are v_{start},v_0,v_1. It is easy to check that the probability to reach the v_0 in G' is exactly $p^3/(p^3 + (1-p)^3)$, in which $p = \mathsf{val}[G](v_0)$.

By applying the algorithm for $(1/2 - e^{-n^\rho})$-gap SSG decision problem on G', the algorithm would be able to distinguish between $\mathsf{val}(G) > 1/2 + n^{-c}$ and $\mathsf{val}(G) < 1/2 - n^{-c}$.

For general (possibly non-stopping) SSG, we use Condon's reduction in [2] that transforms a SSG G to a stopping SSG G' whose value is arbitrarily close to the value of G. The constructed stopping game G' adopts all the vertices of G and inserts cnm new vertices($m = |E|$ in G). For any vertex v, $|\mathsf{val}[G](v) - \mathsf{val}[G'](v)| \leq 2^{(2-c)n}$. By combining these two constructions, we can reduce solving the (n^{-c})-gap decision problem to $(1/2 - e^{-n^\rho})$-gap SSG decision problem.

4 Fast Algorithm for SSGs with few random vertices

An interesting case for solving simple stochastic games is when there are a few random vertices. Gimbert and Horn[6] found an algorithm that runs in $\tilde{O}(|V_{\mathrm{R}}|!)$ time. Their algorithm is based on enumerating a special kind of strategies called **f**-strategies. To avoid simple and time consuming enumerations, our algorithm relies on the following Lemma:

Lemma 1 (Main Lemma). *There's a partial order in* **f***-strategies such that the following holds:*

1. *Any maximal* **f** *corresponds to a pair of optimal strategies.*
2. *Two* **f***-strategies can be compared in polynomial time*
3. *If* **f** *is not maximal, then in polynomial time we can find* **g** *which is better than* **f**

The **f**-strategies are first introduced in [6], and they proved a theorem(Lemma 3 in this paper) on testing whether the **f**-strategy is optimal or not. The brute-force idea is to enumerate all possible $O(n!)$ **f**-strategies and use Lemma 3 to find

the optimal one. Our major contrubution is this Main Lemma, which reduces the problem to a local maximal searching problem and thus enabled us to design faster algorithms.

4.1 f-strategies

In this section we'll first briefly describe [6]'s ideas on what are **f**-strategies and how to test their optimality; this is first introduced in [6] and we mention it again for completeness. Then we show how the partial order in the Main Lemma is defined and prove the Main Lemma. Finally we use existing randomized algorithms for local search problems to improve the expected running time to $\tilde{O}(\sqrt{|V_{\mathrm{R}}|}!)$.

Let $\mathbf{f} = \langle r_1, \ldots, r_m \rangle$ (for simplicity let $r_0 = v_1$) be a permutation of the random vertices, where $m = |V_{\mathrm{R}}|$ is the the number of the random vertices. A **f**-strategy is a pair of positional strategies associated to **f**.

Let R_i be the first i random vertices in the permutation **f**. The *consuming set* C_i is a set of vertices from which player MAX has a strategy $\sigma_{\mathbf{f}}$ for moving the pebble to R_i and at the same time avoid touching any other random vertices, no matter what strategy player MIN chooses. Similarly, there's also a strategy $\tau_{\mathbf{f}}$ for player MIN, such that no matter what player MAX does, vertices outside C_i can never reach a vertex in R_i without touching other random vertices.Obviously $C_0 \subseteq C_1 \subseteq \ldots \subseteq C_m$. This pair of strategies $(\sigma_{\mathbf{f}}, \tau_{\mathbf{f}})$ is called the **f**-strategy regarding to the permutation **f**. For any permutation **f**, let $\mathsf{val}_{\mathbf{f}}(r_i)$ be the probability for player MAX to win if the game starts at vertex r_i, when players follow the **f**-strategies $(\sigma_{\mathbf{f}}, \tau_{\mathbf{f}})$. The following lemmas are first proved in [6].

Lemma 2 (f-strategy). *Given any permutation* **f***, the corresponding* $\sigma_{\mathbf{f}}$ *and* $\tau_{\mathbf{f}}$ *always exist and can be found in polynomial time.*

Lemma 3. *If* **f** *satisfies the* Consistency *and* Progressive *conditions, then the* **f***-strategy is an optimal strategy for the game.*

Consistency: $\mathsf{val}_{\mathbf{f}}(r_1) \geq \cdots \geq \mathsf{val}_{\mathbf{f}}(r_m)$

Progressive: For any random vertex r_i *(*$i > 0$*) with* $\mathsf{val}_{\mathbf{f}}(r_i) > 0$*, at least one of its outgoing edges points to a vertex in* C_{i-1}*.*

There always exists a permutation **f** *that satisfy both conditions.*

For constructing $\{C_i\}$ in polynomial time and more discussions about the *Consistency* and *Progressive* conditions, see [6].

4.2 The Partial Order for f-strategies

A natural way to improve the algorithm by Gimbert and Horn would be smartly updating **f** when it is not *Consistent* or not *Progressive*. However, it is hard to tell which permutation better by simply looking at the values for vertices.

To estimate whether a particular ordering is good or not, we construct a new SSG with respect to the ordering.

Definition 2 (value measure H(f)). *Let G be a SSG and $\mathbf{f}$ be an ordering of random vertices, $G_{\mathbf{f}}$ is a new SSG. $G_{\mathbf{f}}$ has all the vertices and edges in G and m new vertices $u_1, u_2, ..., u_m$, all of them are MAX vertices. The two outgoing edges of u_i go to r_i and r_{i+1} (both outgoing edges of u_m go to r_m). All edges of the form (v, r_i) in G are replaced by (v, u_i) in G_r. Let $H(\mathbf{f}) \triangleq \sum_{i=1}^{m} \mathsf{val}[G_{\mathbf{f}}](u_i)$.*

Let $H_{OPT} = \sum_{i=1}^{m} \mathsf{val}[G](r_i)$. An example on how to compute $H(\mathbf{f})$ is showed in Fig 2. In G, the values are $\mathsf{val}(r_1) = 0, \mathsf{val}(r_2) = 0.5, \mathsf{val}(r_3) = \mathsf{val}(v_1) = 1$. In $G_{\mathbf{f}}$, the values are $\mathsf{val}(r_1) = \mathsf{val}(r_2) = \mathsf{val}(r_3) = \mathsf{val}(v_1) = 1$. So $H(\mathbf{f}) = 4 > H_{OPT}$.

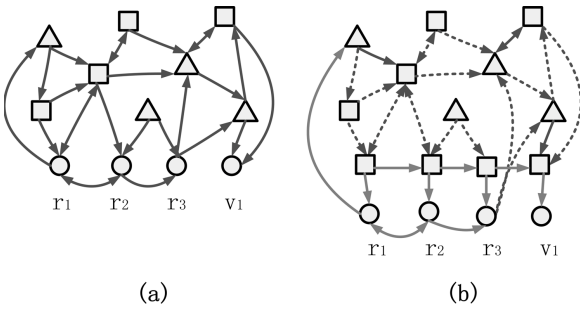

Fig. 2. (a)The game G, Δ are MIN vertices, $\square$ are MAX vertices, $\bigcirc$ are RANDOM vertices. (b)The graph $G_{\mathbf{f}}$, in which $\mathbf{f} = \langle r_1, r_2, r_3, v_1 \rangle$. 4 MAX vertices are added. The dashed lines are the original edges and the solid lines are the added edges.

Lemma 4. *For any permutation $\mathbf{f}$, $H(\mathbf{f}) \geq H_{OPT}$. When $\mathbf{f}$ is both Consistent and Progressive, $H(\mathbf{f}) = H_{OPT}$*

Proof. Consider a permutation $\mathbf{f}$ and its corresponding $G_{\mathbf{f}}$, assume α, β is a pair of optimal strategies for the original game G. Now we construct a strategy α' for player MAX in Game $G_{\mathbf{f}}$: $\alpha'(v) = \alpha(v)$ for all $v \in G$; $\alpha'(u_i) = r_i$ for all $u_i, 1 \leq i \leq m$. When player MAX takes this strategy, it is easy to check β is the also best response for player MIN in G'. So for every $v \in G$, $\mathsf{val}[G_{\alpha,\beta}](v) = \mathsf{val}[G_{\mathbf{f},\alpha',\beta}](v)$ and $\sum_{i=1}^{m} \mathsf{val}(u_i) = \sum_{i=1}^{m} \mathsf{val}[G_{\mathbf{f},\alpha',\beta}](v) = H_{OPT}$. However, MAX may have better strategies in $G_{\mathbf{f}}$, so $H(\mathbf{f}) \geq H_{OPT}$.

When $\mathbf{f}$ is Consistent and Progressive in G, we first prove that $\mathbf{f}$ is also Consistent and Progressive in $G_{\mathbf{f}}$. Let C_i be the consuming sets in G regarding to $\mathbf{f}$. By analyzing the structure of graph $G_{\mathbf{f}}$, we have $C'_i = C_i \cup \{u_1, u_2, ..., u_i\}$ are consuming sets in $G_{\mathbf{f}}$. Consider the strategies (α', β) as defined in the former case. Using the definition of $\mathbf{f}$-strategy, it is easy to verify that (α', β) are $\mathbf{f}$-strategy for $G_{\mathbf{f}}$. So $\mathsf{val}[G_{\mathbf{f}}](r_i) = \mathsf{val}[G](r_i)$, which means $\mathbf{f}$ is still Consistent and Progressive for $G_{\mathbf{f}}$. Therefore $H(\mathbf{f}) = H_{OPT}$.

To compute the optimal strategy and values for $G_{\mathbf{f}}$, we use the following Lemma.

Lemma 5. *For any permutation* $\mathbf{f}$ *and the* $\mathbf{f}$*-strategy* (σ, τ) *in* G*, there is an optimal strategy* (σ', τ') *for* $G_{\mathbf{f}}$ *such that for all* $v \in G$*,* $\tau(v) = \tau'(v)$ *and* $\sigma(v) = \sigma'(v)$*.*

Proof. Let the permutation $\mathbf{f} = \langle r_0, \dots r_m \rangle$ and the corresponding $\mathbf{f}$-strategy $(\sigma_{\mathbf{f}}, \tau_{\mathbf{f}})$. Now we construct strategy for $G_{\mathbf{f}}$ satisfies the conditions.

Denote the consuming sets for $\mathbf{f}$ as $\{C_i\}$. Let $\mathbf{g} = (r_0, r_{t_1}, r_{t_2}, ..., r_{t_m})$ be a permutation for $G_{\mathbf{f}}$ which is consistent and progressive (this ordering always exists by Lemma 3). Since G and $G_{\mathbf{f}}$ have the same random vertices, $\mathbf{f}$ and $\mathbf{g}$ are permutations over the same set. Denote the consuming sets for $\mathbf{g}$ as $\{C'_i\}$, we have $\bigcup_i C_i = \bigcup_i C'_i$. By the construction of $G_{\mathbf{f}}$, we have $C_i = \cup_{j=1}^{i}(C_j \cup \{u_j\})$ and $C'_i = \cup_{j=1}^{\max t_1, t_2, \dots, t_i}(C_j \cup \{u_j\})$.This is because in the $\mathbf{f}$-strategy in G, player MAX's strategy ensures C_i can reach R_i while MIN strategy ensures that no other vertices outside C_i can reach R_i.

Now the optimal strategies (σ', τ') are defined as follows.

$$\sigma'(v) = \begin{cases} \sigma(v), \text{ if } v \in G \\ r_{t_i}, \text{ if } v = u_{t_i} \text{ and } t_i = \max\limits_{j \le i} t_j \\ u_{t_i+1}, \text{ if } v = u_{t_i} \text{ and } t_i < \max\limits_{j \le i} t_j \end{cases} \tag{1}$$

Since the MIN vertices in $G_{\mathbf{f}}$ are the same with G, we simply let $\tau'(v) = \tau(v)$.

Then for any i, the strategy σ' makes sure that no matter what strategy the MIN player uses, C'_i always reach a vertex in $\{r_0, r_1, ..., r_{t_i}\}$. Similarly, the strategy τ' makes sure that no matter what strategy that the MAX player uses, vertices outside C'_i can never reach a vertex in $\{r_0, r_1, ..., r_{t_i}\}$. So (σ', τ') is a valid $\mathbf{f}$-strategy for the permutation $\mathbf{g}$. Since $\mathbf{g}$ is consistency and progressive, (σ', τ') is therefore optimal by Lemma 3.

By this lemma we can find the optimal strategy for MIN player in $G_{\mathbf{f}}$ in polynomial time, because we know that the strategy for player MIN in $\mathbf{f}$-strategy for G is also an optimal strategy for MIN player in $G_{\mathbf{f}}$. By using linear programming we can find the optimal strategy for player MAX in polynomial time.

Definition 3 (progressiveness measure P(f)). *For an permutation* $\mathbf{f} = \langle r_1, \dots r_m \rangle$*,* $P(\mathbf{f})$ *is the smallest* $i (i > 0)$ *such that* r_i *does not have an outgoing edge to* C_{i-1}*. If there's no such* i *or* $\mathsf{val}(r_i) = 0$ *then* $P(r) = m + 1$*.*

Denote the set of all permutations over the random vertices as Π. Searching this space and output the consistent and progressive one takes $\tilde{O}(m!)$ time. But an partial order over Π may help us to find this ordering. We say $\mathbf{f} > \mathbf{g}$ if (1) $H(\mathbf{f}) < H(\mathbf{g})$ or (2) $H(\mathbf{f}) = H(\mathbf{g})$ and $P(\mathbf{f}) > P(\mathbf{g})$.

Any maximal element in $(\Pi, >)$ corresponds to an permutation that is both Consistent and Progressive. Therefore we have proved the first 2 parts of the Main Lemma. To prove part 3 of the Main Lemma, we use the following lemma as a tool to upperbound the H value.

Lemma 6. *If function $f : V \to [0,1]$ satisfy the following conditions, then* $\mathsf{val}(v) \le f(v)$ *for every vertex v.*

1. *For vertex v_1, $f(v_1) = 1$;*
2. *For vertex $v \in V_R$, assume the two outgoing edges are (v, w_1), (v, w_2), $f(v) \ge (f(w_1) + f(w_2))/2$;*
3. *For vertex $v \in V_{MAX}$, assume the two outgoing edges are (v, w_1), (v, w_2), $f(v) \ge \max(f(w_1), f(w_2))$;*
4. *For vertex $v \in V_{MIN}$, assume the two outgoing edges are (v, w_1), (v, w_2), $f(v) \ge \min(f(w_1), f(w_2))$.*

Due to the limit of space, we defer the proof to the full version of the paper.

Lemma 7. *If an permutation $\mathbf{f} = \langle r_1, \dots r_m \rangle$ is not maximal, then there exists an element r_i in $\mathbf{f}$, by deleting r_i and reinsert it in appropriate place we get a new ordering $\mathbf{g}$ such that $\mathbf{g} > \mathbf{f}$.*

Proof. If the ordering $\mathbf{f}$ is not consistent in $G_{\mathbf{f}}$, then there exists some t such that $\mathsf{val}[G_{\mathbf{f}}](r_t) < \mathsf{val}[G_{\mathbf{f}}](r_{t+1})$. Find a place $q > t$ so that $\mathsf{val}[G_{\mathbf{f}}](u_q) < \mathsf{val}[G_{\mathbf{f}}](r_{t+1})$ (if there's no such place then let $q = m + 1$). Delete r_t and reinsert it right before q (if $q = m + 1$ then insert it at the tail). Define $f(v) = \mathsf{val}[G_{\mathbf{f}}](v)$, then for graph $G_{\mathbf{g}}$ f is a valid value function that satisfy the requirements of Lemma 6. Therefore for any vertex v $\mathsf{val}[G_r](v) \ge \mathsf{val}[G_{\mathbf{g}}](v)$. Particularly for the current position of r_t, the corresponding u vertex is u_q in $G_{\mathbf{g}}$, $f(u_q) > \max(f(u_{q+1}), f(r_t))$, so even after reducing $f(u_q)$ to $\max(f(u_{q+1}), f(r_t))$, f is still valid. That is, $H(\mathbf{g}) < H(\mathbf{f})$, $\mathbf{g} > \mathbf{f}$.

If the ordering $\mathbf{f}$ is consistent but not progressive, then assume $P(\mathbf{f}) = t$. Define a graph among the random vertices and r_0 as follows: if an original outgoing edge of r_i goes to a vertex v that is in $C_j \backslash C_{j-1}$, then there's an edge from r_i to r_j. Use breadth first search to find $t' > t$, such that the following holds:

1. There's an edge from $r_{t'}$ to $\{r_0, r_1, ..., r_{t-1}\}$.
2. There's a path from r_t to $r_{t'}$.

Note that such t' must exist because otherwise following the $\mathbf{f}$-strategy, starting from r_t, the pebble will never be able to reach r_0, and therefore the value of r_t is 0, which contradict with the fact that $P(r) \neq m+1$. Also, $\mathsf{val}(r_{t'}) = \mathsf{val}(r_t)$, because if the path from r_t to $r_{t'}$ is $(w_0, w_1, ..., w_k)$ ($w_0 = r_t$ and $w_k = r_{t'}$), then since $\mathsf{val}(w_i) = (\mathsf{val}(w_{i+1}) + \mathsf{val}(r^*))/2$, both w_{i+1} and r^* are ranked lower than t, $\mathsf{val}(w_{i+1}) \le \mathsf{val}(r_t)$, $\mathsf{val}(r^*) \le \mathsf{val}(r_t)$. But $\mathsf{val}(w_0) = \mathsf{val}(r_t)$, by induction for all i $\mathsf{val}(w_i) = \mathsf{val}(w_0) = \mathsf{val}(r_t)$.

Now delete $r_{t'}$ and insert it back before r_t to get a new ordering $\mathbf{g}$. Define $f(v) = \mathsf{val}[G_{\mathbf{f}}](v)$, then for graph $G_{\mathbf{g}}$ f is a valid value function that satisfy the requirements of Lemma 6. Therefore for any vertex v $\mathsf{val}[G_{\mathbf{f}}](v) \ge \mathsf{val}[G_{\mathbf{g}}](v)$. Either all values are equal, in this case $H(\mathbf{f}) = H(\mathbf{g})$ but $P(\mathbf{g}) > P(\mathbf{f})$ so $\mathbf{g} > \mathbf{f}$; or some values are different, in this case $H(\mathbf{g}) < H(\mathbf{f})$ so $\mathbf{g} > \mathbf{f}$.

Since there are only polynomially many ways to delete and reinsert an element, a better ordering can always be found in polynomial time.

4.3 The Randomized Algorithm

Now we can use the existed randomized local minimum searching algorithm to solve the simple stochastic game. The algorithm to solve the value of a simple stochastic game $G = (V, E)$:

1. Randomly choose $\sqrt{|V_{\mathrm{R}}|!}\log(|V_{\mathrm{R}}|!)$ permutations, and let $\mathbf{f_0}$ be the maximal permutation among them;
2. Starting from $\mathbf{f_0}$, repeatedly find better permutation until a maximal permutation is found

By Lemma 7, we can always find a better permutations unless $\mathbf{f}$ is maximal, and there are only $|V_{\mathrm{R}}|!$ permutations, the algorithm will eventually find a maximal permutation and thus the optimal strategy.

The first step takes $\tilde{O}(\sqrt{|V_{\mathrm{R}}|!})$ time, after that, each iteration of the loop will take poly$(|V_{\mathrm{R}}|)$ time, so the key is how many iterations step 2 needs.

Lemma 8. *The probability that step 2 needs more than* $\sqrt{|V_R|!}$ *steps is no more than* $1/(|V_R|)!$.

Proof. Consider any total ordering of the permutations that agrees with the partial ordering we defined. The probability that none of the $\sqrt{|V_{\mathrm{R}}|!}$ largest elements are chosen is at most $(1-\sqrt{|V_{\mathrm{R}}|!}/(|V_{\mathrm{R}}|!))^{\sqrt{|V_{\mathrm{R}}|!}\log(|V_{\mathrm{R}}|!)} = e^{-\log(|V_{\mathrm{R}}|!)} = 1/(|V_{\mathrm{R}}|)!$.

Therefore, the expectation of number of iterations is at most $\sqrt{|V_{\mathrm{R}}|!}$. The running time is $\tilde{O}(\sqrt{|V_{\mathrm{R}}|!})$.

References

1. Shapley, L.: Stochastic games. In: Proceedings of the National Academy of Sciences. (1953)
2. Condon, A.: The complexity of stochastic games. Inf. Comput. **96**(2) (1992) 203–224
3. Condon, A.: On algorithms for simple stochastic games. In: Advances in Computational Complexity Theory, volume 13 of DIMACS Series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society (1993) 51–73
4. Ludwig, W.: A subexponential randomized algorithm for the simple stochastic game problem. Inf. Comput. **117**(1) (1995) 151–155
5. Somla, R.: New algorithms for solving simple stochastic games. In: Proceedings of the Workshop on Games in Design and Verification (GDV 2004). Volume 119 of Electronic Notes in Theoretical Computer Science., Elsevier (2005) 51C65
6. Gimbert, H., Horn, F.: Solving simple stochastic games. In: CiE '08: Proceedings of the 4th conference on Computability in Europe, Berlin, Heidelberg, Springer-Verlag (2008) 206–209