Solving Simple Stochastic Games with Few Random Vertices

Hugo Gimbert, Florian Horn

SOLVING SIMPLE STOCHASTIC GAMES WITH FEW RANDOM VERTICES

HUGO GIMBERT AND FLORIAN HORN

LaBRI, CNRS, Bordeaux, France
e-mail address: hugo.gimbert@labri.fr

CWI, Amsterdam, The Netherlands
e-mail address: f.horn@cwi.nl

Abstract. Simple stochastic games are two-player zero-sum stochastic games with turn-based moves, perfect information, and reachability winning conditions.

We present two new algorithms computing the values of simple stochastic games. Both of them rely on the existence of optimal *permutation strategies*, a class of positional strategies derived from permutations of the random vertices. The "permutation-enumeration" algorithm performs an exhaustive search among these strategies, while the "permutation-improvement" algorithm is based on successive improvements, *à la* Hoffman-Karp.

Our algorithms improve previously known algorithms in several aspects. First they run in polynomial time when the number of random vertices is fixed, so the problem of solving simple stochastic games is fixed-parameter tractable when the parameter is the number of random vertices. Furthermore, our algorithms do not require the input game to be transformed into a stopping game. Finally, the permutation-enumeration algorithm does not use linear programming, while the permutation-improvement algorithm may run in polynomial time.

Introduction

Simple stochastic games (SSGs) are played by two players called Max and Min in a sequence of steps. The players move a pebble along the edges of a directed graph (V, E) whose vertices are partionned into three sets: V_{Max}, V_{Min}, and V_{R}. When the pebble is on a vertex of V_{Max} or V_{Min}, the corresponding player chooses an outgoing edge and moves the pebble along it. When the pebble is on a vertex of V_{R} (a *random* vertex), the outgoing edge is chosen randomly according to a fixed probability distribution. The players have opposite goals, as Max wants to reach a special sink vertex ◎ while Min wants to avoid it forever. An example of SSG is depicted in Figure 1, with vertices of V_{Max} represented as ○'s, vertices of V_{Min} represented as □'s, and vertices of V_{R} represented as △'s.

SSGs are a natural model of reactive systems. Consider, for example, a hardware component. It can be modelled as an SSG, whose vertices represent the global states of the component and the target is some error state to avoid. The nature of a given vertex

2000 ACM Subject Classification: Games, Stochastic Processes.

Key words and phrases: simple stochastic games, algorithm.

This research was partially supported by the french project ANR "DOTS". The second author held the tenure of an ERCIM "Alain Bensoussan" fellowship programme.

LOGICAL METHODS
IN COMPUTER SCIENCE

DOI:10.2168/LMCS-???

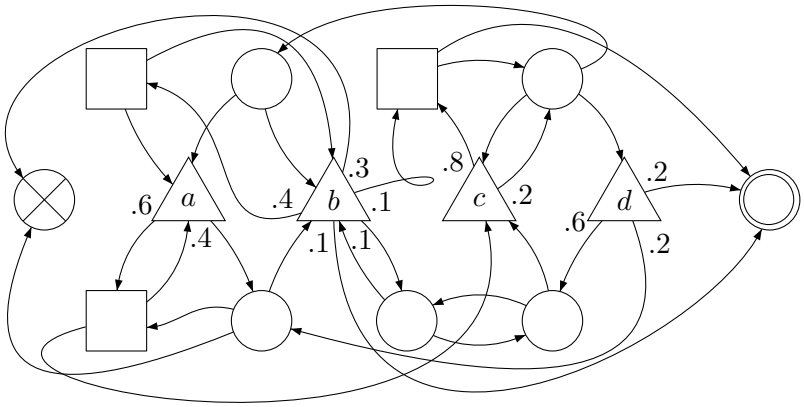

Figure 1: A Simple Stochastic Game

depends on who can influence the immediate evolution of the system: it is a Min vertex if the software can choose between different options, a Max vertex if there is a (non-deterministic) input asked from the user, and a random vertex if the evolution depends on a stochastic environment. An optimal strategy for Min can then be used as the basis for the synthesis of a "good" driver, *i.e.* one which minimises the probability of entering the error state independently of the behaviour of the user.

The main algorithmic problem about SSGs is the computation of values of the vertices and optimal strategies for the players. This problem was first adressed by Condon, who showed that deciding whether the value of a vertex is greater than $\frac{1}{2}$ belongs to NP and co-NP [Con92]. Condon's algorithm guesses non-deterministically the values of vertices, which are rational numbers of linear size, and checks that they are solutions of some *local optimality equations*. This algorithm is correct only for *stopping* games, where the pebble reaches either the target or a sink target with probability one, regardless of the players' strategies. Any SSG can be transformed in polynomial time into a stopping SSG with (almost) the same values, but it incurs a quadratic blow-up of the size of the game.

Three other algorithms for solving SSGs are presented in [Con93]. The first one computes the values of the vertices using a quadratic program with linear constraints. The second one computes iteratively from below the values of the vertices, and the third is a strategy improvement algorithm *à la* Hoffman-Karp [HK66]. The two latter algorithms, as the ones recently proposed in [Som05], solve a series of linear programs which could be of exponential length. Furthermore, solving a linear program requires high-precision arithmetic, even if it can be done in polynomial time [Kha79, Ren88]. The best randomised algorithms achieve sub-exponential expected time $e^{O(\sqrt{n})}$ [Lud95, Hal07].

In this paper we present two algorithms computing the values and optimal strategies in SSGs: the "permutation-enumeration" and the "permutation-improvement" algorithms. The common basis for both algorithms is that optimal strategies can be looked for in a subset of the positional strategies called *permutation strategies*. Permutation strategies are derived from permutations over the random vertices. In order to find optimal strategies, the permutation-enumeration algorithm performs an exhaustive search among all permutation strategies, whereas the permutation-improvement algorithm performs successive improvements of permutation strategies, *à la* Hoffman-Karp [HK66].

The permutation-enumeration and the permutation-improvement algorithms share two advantages over existing algorithms. First, they perform much better on SSGs with few random vertices, as they run in polynomial time when the number of random vertices is logarithmic in the size of the game: it follows that the problem of solving SSGs is fixed-parameter tractable when the parameter is the number of random vertices. Second, they do not rely on the transformation of the input SSG into a stopping SSG, which avoids the quadratic blow-up of the size of the game. Moreover, the permutation-enumeration algorithm does not use linear or quadratic programming, (it just computes the solutions to linear systems) and its worst-case complexity is $\mathrm{O}\big(|V_{\mathrm{R}}|! \cdot (|E| + |\delta|)\big)$, where $|V_{\mathrm{R}}|$ is the number of random vertices, $|E|$ is the number of edges and $|\delta|$ is the maximal bit-length of transition probabilities. The nominal complexity of the permutation-improvement algorithm is higher but we do not know any non-trivial lower bound for its complexity: the permutation-improvement algorithm may actually run in polynomial time.

Outline. In Section 1, we provide formal definitions for SSGs, values and optimal strategies. We describe then in Section 2 the central notion of permutation strategies. Section 3 presents the permutation-enumeration algorithm, based on the *self-consistency* and *liveness* properties. Section 4 introduces an improvement policy for permutations which leads to the permutation-improvement algorithm.

1. Simple Stochastic Games

1.1. **Plays and strategies.** A *simple stochastic game* is a tuple $(V, V_{\mathrm{Max}}, V_{\mathrm{Min}}, V_{\mathrm{R}}, E, \delta, \circledcirc)$, where (V, E) is a graph, $(V_{\mathrm{Max}}, V_{\mathrm{Min}}, V_{\mathrm{R}})$ is a partition of V, and $\circledcirc$ is a distinguished sink vertex in V called the *target* of the game. The transitions from the random vertices are equipped with probabilities described by the function $\delta : V_{\mathrm{R}} \to V \to [0, 1]$, such that for all $v \in V_{\mathrm{R}}$, $w \in V$, $\delta(v)(w) > 0 \Rightarrow (v, w) \in E$, and $\sum_{w \in V} \delta(v)(w) = 1$.

An *infinite play* ρ is an infinite sequence $\rho_0\rho_1 \cdots \in V^\omega$ of vertices such that for all $i \in \mathbb{N}, (\rho_i, \rho_{i+1}) \in E$. It is *winning for* Max if there is a $i \in \mathbb{N}$ such that $\rho_i = \circledcirc$ (as $\circledcirc$ is a sink, it follows that $\forall j > i, \rho_j = \circledcirc$). Otherwise, ρ is *winning for* Min. A *finite play* is a finite prefix of an infinite play.

A (pure) *strategy* for Max is a mapping $\sigma : V^* V_{\mathrm{Max}} \to V$ such that for each finite play $h = h_0 \dots h_i$ ending in a Max vertex, $(h_i, \sigma(h)) \in E$. It is *positional* if it only depends on the last vertex of h: $\sigma(h) = \sigma(h_i)$. A play $\rho_0\rho_1 \dots$ is *consistent with* σ if for every i such that $\rho_i \in V_{\mathrm{Max}}$, $\rho_{i+1} = \sigma(\rho_0 \dots \rho_i)$. Strategies for Min are defined analogously and are generally denoted by τ.

1.2. **Measures and values.** The set of plays is made into a measurable space on the σ-algebra generated by the canonical projections $\{V_i\}_{i \in \mathbb{N}}$, where $V_i(\rho_0\rho_1 \dots) = \rho_i$ [Bil95]. Once an initial vertex v and two strategies σ and τ for players Max and Min have been fixed, the probability measure $\mathbb{P}_v^{\sigma,\tau}$ is defined by:

$$\begin{aligned}
&\mathbb{P}_v^{\sigma,\tau}\left(V_0 = v\right) = 1 \ , \\
&\mathbb{P}_v^{\sigma,\tau}\left(V_{i+1} = \sigma(V_0 \dots V_i) \mid V_i \in V_{\mathrm{Max}}\right) = 1 \ , \\
&\mathbb{P}_v^{\sigma,\tau}\left(V_{i+1} = \tau(V_0 \dots V_i) \mid V_i \in V_{\mathrm{Min}}\right) = 1 \ , \\
&\mathbb{P}_v^{\sigma,\tau}\left(V_{i+1} \mid V_i \in V_{\mathrm{R}}\right) = \delta(V_i)(V_{i+1}) \ .
\end{aligned}$$

The expectation of a real-valued, measurable and bounded function φ under $\mathbb{P}_v^{\sigma,\tau}$ is denoted $\mathbb{E}_v^{\sigma,\tau}[\varphi]$. We will often use implicitly the following formulae which rule the probabilities and expectations once a finite prefix $h = h_0 \dots h_i$ is fixed:

$$\mathbb{P}_v^{\sigma,\tau}(\Gamma \mid V_0 \dots V_i = h_0 \dots h_i) = \mathbb{P}_{h_i}^{\sigma[h],\tau[h]}(\Gamma[h]) \ , \tag{1.1}$$

$$\mathbb{E}_v^{\sigma,\tau}[\varphi \mid V_0 \dots V_i = h_0 \dots h_i] = \mathbb{E}_{h_i}^{\sigma[h],\tau[h]}[\varphi[h]] \ , \tag{1.2}$$

where $\sigma[h](\rho_0\rho_1\dots) = \sigma(h_0 \dots h_{i-1}\rho_0\rho_1\dots)$, and $\tau[h]$, $\Gamma[h]$, and $\varphi[h]$ are defined analogously.

If we fix only Max's strategy σ and the initial vertex v, the target vertex will be reached with probability at least:

$$\inf_\tau \mathbb{P}_v^{\sigma,\tau}(\mathrm{Reach}(\circledcirc)) \ ,$$

where $\mathrm{Reach}(\circledcirc)$ is the event $\{\exists i \in \mathbb{N}, V_i = \circledcirc\}$. Starting from v, player Max has strategies that guarantee a winning outcome with a probability greater than:

$$\mathrm{val}_*(v) = \sup_\sigma \inf_\tau \mathbb{P}_v^{\sigma,\tau}(\mathrm{Reach}(\circledcirc)) \ ,$$

minus ϵ for any $\epsilon > 0$. Symmetrically, Min has strategies that guarantee a winning outcome with a probability less than:

$$\mathrm{val}^*(v) = \inf_\tau \sup_\sigma \mathbb{P}_v^{\sigma,\tau}(\mathrm{Reach}(\circledcirc)) \ ,$$

plus ϵ for any $\epsilon > 0$. It is clear that $\mathrm{val}_*(v) \leq \mathrm{val}^*(v)$. In the case of SSGs, stronger results are known:

Theorem 1.1 ([Sha53, Gil57, LL69])**.** *Let $G = (V, V_{\mathrm{Max}}, V_{\mathrm{Min}}, V_R, E, \circledcirc, \delta)$ be a SSG. Then, for any vertex $v \in V$,*

$$\mathrm{val}_*(v) = \mathrm{val}^*(v) \ .$$

This common value is denoted by $\mathrm{val}(v)$*. Furthermore, there are positional optimal strategies for both players, i.e. positional strategies $\sigma^\#$ and $\tau^\#$ such that, for any strategies σ and τ:*

$$\mathbb{P}_v^{\sigma,\tau^\#}(\mathrm{Reach}(\circledcirc)) \leq \mathrm{val}(v) \leq \mathbb{P}_v^{\sigma^\#,\tau}(\mathrm{Reach}(\circledcirc)) \ .$$

1.3. **Normalised games.** A SSG is *normalised* if the only vertex with value 1 is the target $\circledcirc$ and there is only one (sink) vertex $\otimes$ with value 0. Our motivations for the introduction of this notion are twofold. First, several proofs are much simpler for normalised games. Second, any SSG can be reduced to an equivalent normalised game in linear time and the resulting game is smaller than the original one. This reduction is presented on Figure 2: it simply consists in merging the region with value one into $\circledcirc$ and the region with value zero into a new sink vertex $\otimes$.

In the remainder of this article, we assume that we are working on a normalised SSG $G = (V, V_{\mathrm{Max}}, V_{\mathrm{Min}}, V_{\mathrm{R}}, E, \delta, \circledcirc, \otimes)$, with k random vertices.