Solving Simple Stochastic Games with Few Random Nodes Faster Using Bland's Rule

David Auger
DAVID laboratory, University of Versailles Saint-Quentin-en-Yvelines, France
david.auger@uvsq.fr

Pierre Coucheney
DAVID laboratory, University of Versailles Saint-Quentin-en-Yvelines, France
pierre.coucheney@uvsq.fr

Yann Strozecki
DAVID laboratory, University of Versailles Saint-Quentin-en-Yvelines, France
yann.strozecki@uvsq.fr

Abstract

The best algorithm so far for solving Simple Stochastic Games is Ludwig's randomized algorithm [21] which works in expected $2^{O(\sqrt{n})}$ time. We first give a simpler iterative variant of this algorithm, using Bland's rule from the simplex algorithm, which uses exponentially less random bits than Ludwig's version. Then, we show how to adapt this method to the algorithm of Gimbert and Horn [15] whose worst case complexity is $O(k!)$, where k is the number of random nodes. Our algorithm has an expected running time of $2^{O(k)}$, and works for general random nodes with arbitrary outdegree and probability distribution on outgoing arcs.

2012 ACM Subject Classification Theory of computation → Algorithmic game theory

Keywords and phrases simple stochastic games, randomized algorithm, parametrized complexity, strategy improvement, Bland's rule

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.9

Related Version A full version of the paper is available at `https://arxiv.org/abs/1901.05316`.

1 Introduction

A *simple stochastic game*, SSG for short, is a two-player zero-sum game, a turn-based version of *stochastic games* introduced by Shapley [22]. SSGs were introduced by Condon [11] and provide a simple framework that allows to study algorithmic complexity issues underlying reachability objectives. An SSG is played by moving a pebble on a graph. Some nodes are divided between players MIN and MAX: if the pebble reaches a node controlled by a player then she has to move the pebble along an arc leading to another node. Some other nodes are ruled by chance, the pebble following one outgoing arc according to some given probability distribution. Finally, there are sink nodes with a rational value, which is the gain that MAX-player achieves when the pebble reaches this sink.

Player MAX's objective is, given a starting node for the pebble, to maximize the expectation of her gain against any strategy of MIN. One can show that it is enough to consider stationary deterministic strategies for both players [11]. Though seemingly simple since the number of stationary deterministic strategies is finite, the task of finding a pair of optimal strategies, or equivalently, of computing the so-called *optimal values* of nodes, is in complexity class PPAD [13] but not known to be in P.

Simple stochastic games are a powerful model since they can simulate many other games such as parity games, mean or discounted payoff games [2, 7]. However these games are believed to be simpler than SSGs and better algorithms are known for them; in particular, parity game is in quasi-polynomial time [5]. Stochastic versions of the previous games

also exist and are computationally equivalent to SSGs [2]. Interestingly, SSGs have many application domains, for instance autonomous urban driving [9], smart energy management [8], model checking of the modal μ-calculus [23], etc.

There are some restrictions for SSGs for which the problem of finding optimal strategies is tractable. If the game is acyclic, it can be solved in linear time, and in polynomial time for almost acyclic games (few cycles or small feedback arc sets) [3]. If there is no randomness, the game can be solved in almost linear time [1]. Furthermore, Gimbert and Horn were the first to extend this result by giving Fixed Parameter Tractable (FPT) algorithms in the number of random nodes [15]. They indeed show that optimal strategies depend only on the ordering of the values of random nodes, and not on their actual values. Using this idea, they devise two algorithms. The first one exhaustively enumerates these orders until it finds one that actually corresponds to optimal values. The second one is a strategy improvement algorithm based on an iterative refinement of the orders. Both have a complexity of $k!n^{O(1)}$, where k is the number of random nodes. It has been improved to $\sqrt{k!}n^{O(1)}$ expected time in [12], by randomly selecting a good strategy as a starting point for a strategy improvement algorithm. In fact, as remarked in [6], the distance between the values of two consecutive strategies in any strategy improvement algorithm depends on the number of random nodes. Hence any SSG can be solved in time $4^k n^{O(1)}$ (in fact $\sqrt{6}^k n^{O(1)}$ using Lemma 1.1 in [3]). The complexity has been further improved to $2^k n^{O(1)}$ in [19], by using a value iteration algorithm. Here a bit of caution is in order; in some papers, random nodes can have an arbitrary outdegree and probability distribution on outgoing arcs, and in some other they must be binary with uniform distribution. In the former case, if we denote by p the bit-size of the largest probability distribution on a random node, the first two cited algorithms have a complexity of $p \cdot k!$ and $p \cdot \sqrt{k!}$. On the other hand, the two algorithms with an exponential complexity in k have an exponential dependency on p when adapted to this context.

Without the previous restrictions, only algorithms running in exponential time are known. Most of them are strategy improvement algorithms, which produce a sequence of strategies of increasing values. These algorithms, such as the classical Hoffman-Karp [18] algorithm, rely on the switch operation, which by a local best-response, produces a strategy with better value. Several ways of choosing the nodes that are switched have been proposed [24], which can be compared to the rules for pivot selection for the simplex algorithm in linear programming. Though efficient in practice, these algorithms fail to run in polynomial time on a well designed input [14]. The best algorithm so far, proposed by Ludwig [21, 16], is also a strategy iteration algorithm using a randomized version of Bland's rule [4] to choose a switch. It solves any SSG in expected time $2^{O(\sqrt{n})}$. The first analysis of this kind of algorithm is due to Kalai [20] and it has been slightly improved recently [17].

Our contributions

In Section 3, we present an iterative variant of Ludwig's recursive algorithm which uses less random bits. In the rest of the paper we adapt the idea of this algorithm to carefully enumerate orders of random nodes in an SSG. First, in Section 4, we present a pivot operation yielding a strategy improvement algorithm, which improves the one of [15]. This pivot operation comes from a randomized dichotomy on all orders that we explain in details in Section 5, using an auxilliary game similar to the one of [12]. We prove that our algorithm finds the optimal strategies in expected time polynomial in 2^k and p, where k is the number of random nodes and p is the maximum bit-length of a distribution on a random node, answering positively a question of Ibsen-Jensen and Miltersen [19].

2 Definitions and classic results on simple stochastic games

We here review definitions and results related to SSGs. We only sketch what we need and refer to longer expositions such as [11, 24] for more details.

▶ **Definition 1** (SSG). *A simple stochastic game (SSG) is defined by a directed graph $G = (V, A)$, where V is the set of nodes and A the set of arcs, together with a partition of V in four parts V_{MAX}, V_{MIN}, V_{RAN} and V_{SINK}, whose elements are respectively called* MAX*-nodes,* MIN*-nodes,* RAN*-nodes (for random) and sinks. We require that every node $x \in V$ has outdegree at least one, while sink nodes have outdegree exactly* 1 *consisting of a single loop on themselves. We also specify for every sink $x \in V_{\text{SINK}}$ a value $Val(x)$ which is a rational number, and for every random node $x \in V_{\text{RAN}}$ a rational probability distribution $p(x)$ on the outneighbours of x.*

In the original version of Condon [11], all nodes except sinks have outdegree exactly two, the probability distribution on every RAN-node is $(\frac{1}{2}, \frac{1}{2})$, and there are only two sinks, one with value 0 and another with value 1. Here, we allow more than two sinks, with general rational values, and also allow more than outdegree two for all non-sink nodes, with an arbitrary probability distribution for RAN-nodes. However, for Ludwig's Algorithm (see Algorithms 1, 2 and 3 in Section 3) we shall suppose that all MAX-nodes have outdegree 2 and call such games MAX*-binary*.

Strategies and values

We now define strategies, by which we mean stationary and pure strategies. This is enough for our purpose and it turns out to be sufficient for optimality, see [11]. Such strategies specify the choice of a neighbour for every node of a given player.

▶ **Definition 2** (Strategy). *A* strategy *for player* MAX *is a map σ from V_{MAX} to V such that $\forall x \in V_{\text{MAX}}$, $(x, \sigma(x)) \in A$.*

Strategies for player MIN are defined analogously on MIN-nodes and are usually denoted by τ.

▶ **Definition 3** (Play). *A* play *is a sequence of nodes $x_0, x_1, x_2, \dots$ such that for all $t \geq 0$, $(x_t, x_{t+1}) \in A$. Such a play is* consistent *with strategies σ and τ, respectively for player* MAX *and player* MIN*, if for all $t \geq 0$, $x_t \in V_{\text{MAX}} \Rightarrow x_{t+1} = \sigma(x_t)$ and $x_t \in V_{\text{MIN}} \Rightarrow x_{t+1} = \tau(x_t)$.*

A couple of strategies σ, τ and an initial node $x_0 \in V$ define recursively a random play consistent with σ, τ by setting (i) $x_{t+1} = \sigma(x_t)$ if $x_t \in V_{\text{MAX}}$, (ii) $x_{t+1} = \tau(x_t)$ if $x_t \in V_{\text{MIN}}$, (iii) $x_{t+1} = x_t$ if $x_t \in V_{\text{SINK}}$, and finally (iv) x_{t+1} is one of the outneighbours of x_t, randomly chosen independently of everything else according to probability distribution $p(x)$, if $x_t \in V_{\text{RAN}}$.

Hence, this defines a probability measure $\mathbb{P}^{x_0}_{\sigma,\tau}$ on plays consistent with σ, τ. Note that if a play contains a sink node x_s, then at every subsequent time the play stays in x_s. Such a play is said to *reach* sink x_s. To every play $x_0, x_1, \dots$ we associate a value which is the value of the sink reached by the play if any, and 0 otherwise. If we denote by X this value, then X is a random variable once two strategies and an initial node x are fixed. We are interested in the expected value of this quantity, which we call the value of a node $x \in V$ under strategies σ, τ: $\text{Val}_{\sigma,\tau}(x) = \mathbb{E}^x_{\sigma,\tau}(X)$ where $\mathbb{E}^x_{\sigma,\tau}$ is the expected value under probability $\mathbb{P}^x_{\sigma,\tau}$.

The goal of player MAX is to maximize this (expected) value, and the best he can ensure against a strategy τ is $\text{Val}_{*,\tau}(x) := \max_\sigma \text{Val}_{\sigma,\tau}(x)$ where the maximum is considered over all MAX-strategies (which are in finite number). Similarly, against σ player MIN can ensure that the expected value is at most $\text{Val}_{\sigma,*}(x) := \min_\tau \text{Val}_{\sigma,\tau}(x)$.

Finally, the value of a node x is $\mathrm{Val}_{*,*}(x) := \max_\sigma \mathrm{Val}_{\sigma,*}(x) = \min_\tau \mathrm{Val}_{*,\tau}(x)$. The fact that these two quantities are equal is nontrivial, and it can be found for instance in [11]. A pair of strategies σ^*, τ^* such that, for all nodes x, $\mathrm{Val}_{\sigma^*,\tau^*}(x) = \mathrm{Val}_{*,*}(x)$ always exists and these strategies are said to be *optimal strategies*. It is polynomial-time equivalent to compute optimal strategies or to compute the values of all nodes in the game.

▶ **Definition 4** (Stopping SSG). *An SSG is said to be* stopping *if for every couple of strategies almost all plays eventually reach a sink node.*

Usually, this condition is required in order to ensure simple optimality conditions (Thm. 5 below). Condon [11] proved that every SSG G can be reduced in polynomial time to a stopping SSG G' whose size is quadratic in the size of G, and whose values almost remain the same. The values of the new game are close enough to recover the values of the original game. A problem for us is that squaring the size of the game does not behave well relatively to precise complexity bounds.

However, in our case we need a milder condition. We call a MAX-strategy σ stopping if, for any MIN-strategy τ, the random play consistent with (σ, τ) reaches a sink with probability one.

▶ **Theorem 5** (Optimality conditions, [11]). *Let G be an SSG, σ a stopping* MAX*-strategy and τ a* MIN*-strategy. Then (σ, τ) are optimal strategies if and only if*

- *for every $x \in V_{\text{MAX}}$, $Val_{\sigma,\tau}(x) = \max_{(x,y)\in A} Val_{\sigma,\tau}(y)$;*
- *for every $x \in V_{\text{MIN}}$, $Val_{\sigma,\tau}(x) = \min_{(x,y)\in A} Val_{\sigma,\tau}(y)$.*

Switches and strategy improvement

Consider the usual partial order on real vectors indexed by V, i.e. for $w_1, w_2 \in \mathbb{R}^V$, denote $w_1 \leq w_2$ if $w_1(x) \leq w_2(x)$ for all $x \in V$, and denote $w_1 < w_2$ if $w_1 \leq w_2$ and at least one inequality is strict. For two MAX-strategies σ, σ', simply denote $\sigma \leq \sigma'$ (resp. $\sigma < \sigma'$) if $\mathrm{Val}_{\sigma,*} \leq \mathrm{Val}_{\sigma',*}$ (resp. $\mathrm{Val}_{\sigma,*} < \mathrm{Val}_{\sigma',*}$). Define a similar order on MIN-strategies.

A switch, given a strategy, is the fact of changing this strategy at a node (or a set of nodes) in order to obtain a new one.

▶ **Definition 6.** *Let σ, σ' be* MAX*-strategies. We say that σ' is a profitable switch of σ if for all $x \in V_{\text{MAX}}$, one has $Val_{\sigma,*}(\sigma'(x)) \geq Val_{\sigma,*}(\sigma(x))$ with this condition strict for at least one* MAX*-node (such a node is said to be switchable).*

Indeed, the following result states that such a switch actually improves values

▶ **Theorem 7** ([10], [24]). *If σ' is a profitable switch of σ, then $\sigma' > \sigma$.*

Before ending this section, please note that Th. 5 can be restated in terms of nonexistence of switchable node. Hence, we have the following result:

▶ **Theorem 8.** *A stopping* MAX*-strategy is optimal if and only if it has no switchable nodes.*

For the last section, we require another form of switch.

▶ **Theorem 9** ([10], [24]). *Let σ, σ' be stopping* MAX*-strategies and τ, τ' be* MIN*-strategies such that for all $x \in V_{\text{MAX}}$, $Val_{\sigma,\tau}(\sigma'(x)) \geq Val_{\sigma,\tau}(\sigma(x))$ and for all $x \in V_{\text{MIN}}$, $Val_{\sigma,\tau}(\tau'(x)) \geq Val_{\sigma,\tau}(\tau(x))$ with one of these conditions strict for at least one node. Then $Val_{\sigma',\tau'} > Val_{\sigma,\tau}$.*

Orders

For $k \geq 1$ consider the set of integers $[1,k] = \{1, 2, \cdots, k\}$ and let $\mathcal{T}(k)$ denote the set of total orders on $[1,k]$. For sake of clarity we view these orders as sets of couples $(i,j) \in [1,k]^2$ satisfying reflexivity, transitivity and antisymmetry.

If $t \in \mathcal{T}(k)$, it can also be described in *ascending ordering* such as $[x_1, x_2, \ldots, x_k]$ where $(x_i, x_j) \in t$ if and only if $i \leq j$. An *interval* in t is a sequence of consecutive elements in ascending ordering. The rank of an element $x \in [1,k]$ is the number of elements that are lower of equal to x in t, i.e. it is i if $x = x_i$ with notation above.

For lack of a better word, we define a *pretotal order* as an antisymmetric and reflexive relation and denote by $\mathcal{P}(k)$ the set of pretotal orders on $[1,k]$. If $p \in \mathcal{P}(k)$ and $(i,j) \notin p$ is such that $p \cup \{(i,j)\}$ is still antisymmetric, we denote simply by $p + (i,j)$ this new pretotal order.

If $t \in \mathcal{T}(k)$ and $v_1, v_2, \cdots, v_k$ are real numbers, we say that the v_i's are *nondecreasing* along t if $(i,j) \in t \Rightarrow v_i \leq v_j$. Likewise, we say that t is a *nondecreasing order* for $v_1, v_2, \ldots, v_k$.

3 Iterative formulation of Ludwig's algorithm

In this part, we suppose that G is MAX-binary. Hence, if a node x is switchable there is a single possibility for changing the strategy's choice at this node. Let $switch(\sigma, x)$ denote the profitable switch obtained from σ by switching σ at node x.

3.1 Bland's rule version

In [21], Ludwig mentions that his algorithm is a version of Bland's rule, however he does not make it explicit and gives a recursive definition. We formulate his algorithm iteratively (see Algorithm 1), and show that instead of randomly choosing a node at every step, we can choose a total order on nodes prior to the execution of the algorithm. This version uses much less random bits : $O(n \log n)$ bits instead of $2^{O(\sqrt{n})}$ in average in Ludwig's version.

Algorithm 1: Bland's rule formulation for Ludwig's Algorithm.

input : G max-binary SSG, initial stopping MAX-strategy σ.
output : an optimal MAX-strategy

· Pick randomly and uniformly a total order Θ on MAX-nodes
while σ *is not optimal* **do**
  · compute the set of *switchable* nodes for σ
  · let x be the first switchable node in order Θ
  · $\sigma \longleftarrow switch(\sigma, x)$
return σ

By Theorems 7 and 8 if we proceed by switching Strategy σ until there are no more switchable nodes, we reach an optimal strategy in a finite number of steps. The number of steps is at most the number of MAX-strategies, i.e. $2^{|V_{\text{MAX}}|}$. However, we have the following:

▶ **Theorem 10.** *The expected number of strategies considered by Alg. 1 is at most* $e^{2\sqrt{|V_{\text{MAX}}|}}$.

3.2 Analysis of Algorithm 1

Our strategy to prove Theorem 10 is to reformulate Alg. 1 as a recursive algorithm (see Alg. 3), which is close to Ludwig's algorithm in [21]. The proofs of this section will be provided in a long version of this article; they are quite similar to Ludwig's, with a bit of caution on the moments where random choices are made. In particular, we detail our strategy in this part since it will be helpful to understand our results in section 4 where the context is more involved.

Stated as above, it is perhaps unclear how Alg. 1 has a recursive structure. Too see this, consider an execution of Alg. 1, and let x_1 be the last MAX-node in the order Θ. In the beginning, the current strategy σ makes an initial choice $\sigma(x_1)$ on x_1, which does not change until the first time when x_1 becomes switchable (if this happens). If x_1 is switched, then $\sigma(x_1)$ will then remain unchanged until the end of this algorithm. Hence, once Θ is fixed, we can think of this execution as two parts, where $\sigma(x_1)$ is fixed in each part. These can then be decomposed as subparts where $\sigma(x_1)$ and $\sigma(x_2)$ are fixed (where x_2 is the second-to-last MAX-node in order Θ), and so on.

Generalization to partially fixed strategies

To formalize the discussion above, we give a generalization which can be applied to the case where $\sigma(x)$ is fixed for some vertices in a given set F (see Alg. 2).

In the following, if F is a set of MAX-nodes and σ is a MAX-strategy, a (σ, F)-*compatible strategy* is any MAX-strategy σ' such that $\forall x \in F$, $\sigma'(x) = \sigma(x)$. For F and σ fixed, there is always a (σ, F)-strategy that is better than all others. It can be obtained by solving the game where any $x \in F$ is replaced by a random node with a probability 1 to go to $\sigma(x)$. We call such a (σ, F)-compatible strategy *optimal* and we denote it by $\mathrm{opt}(\sigma, F)$. In particular, an optimal $(\sigma, \emptyset)$-strategy is an optimal strategy for G, whereas σ is the only (σ, V_{MAX})-compatible strategy.

Algorithm 2: Iterative formulation for Ludwig's Algorithm with partial strategies.

input : G MAX-binary SSG, total order Θ on V_{MAX}, subset $F \subset V_{\text{MAX}}$, initial MAX-strategy $\sigma = \sigma_0$.

output : a (σ, F)-compatible optimal MAX-strategy $\mathrm{opt}(\sigma, F)$.

while σ *is not an optimal* (σ_0, F)*-compatible strategy* **do**

- compute the set of *switchable* nodes for σ
- let v be the first switchable node in order Θ which is not in F
- $\sigma \longleftarrow \mathit{switch}(\sigma, v)$

return σ

Recursive reformulation

Finally, we give a recursive version of Alg. 2 (see Alg. 3) which we use to derive the bound. The equivalence between these two algorithms should be clear by the previous explanations.

Evaluating the number of switches

Let $f^{\Theta}(\sigma, F)$ be the total number of switches performed by Algorithm 3 on input σ, Θ, F. We consider for the following lemma an execution of this algorithm.

Algorithm 3: Recursive formulation for Ludwig's Algorithm with partial strategies.

input : G MAX-binary SSG, total order Θ on V_{MAX}, subset $F \subset V_{\text{MAX}}$, initial MAX-strategy σ_0.

output : a (σ, F)-compatible optimal MAX-strategy $\text{opt}(\sigma, F)$.

if $F == V_{\text{MAX}}$ **then** return σ
else
  · Let v_0 be the last node not in F according to order Θ
  · Recursively compute σ_1, an optimal $(\sigma_0, F \cup \{v_0\})$-compatible strategy
  if σ_1 *is an optimal* (σ_0, F)*-compatible strategy* **then return** σ_1
  else
    · Let $\sigma_2 \longleftarrow \mathit{switch}(\sigma_1, v_0)$
    · Recursively compute σ^*, an optimal $(\sigma_2, F \cup \{v_0\})$-compatible strategy
    return σ^*

▶ **Lemma 11.** *Let σ_0 be the initial strategy and v_0 be the last node which is not in F, according to order Θ. Define $B \subset V_{\text{MAX}} \setminus F$ to be the set of nodes v such that $\text{opt}(\sigma_0, F \cup \{v\}) \not> \text{opt}(\sigma_0, F \cup \{v_0\})$. Then $f^\Theta(\sigma_0, F) \leq f^\Theta(\sigma_0, F \cup \{v_0\}) + 1 + f^\Theta(\sigma_2, F \cup B\})$, where σ_2 is $\text{opt}(\sigma_0, F \cup \{v_0\}) = \sigma_1$, switched at v_0.*

Now, let us denote $\Phi(n) = \sup_{G,\sigma} \mathbb{E}^\Theta \left[f^\Theta_G(\sigma, \emptyset) \right]$ where the supremum is considered over all SSG G with n MAX-nodes and all MAX-strategies σ in G. The average is considered over all possible prior choices of order Θ, the rest of the algorithm being deterministic.

▶ **Lemma 12.** *For all $n \geq 1$, $\Phi(n) \leq \Phi(n-1) + 1 + \frac{1}{n} \sum_{i=0}^{n-1} \Phi(i)$.*

In order to conclude and prove Theorem 10, we now just have to infer the bound for sequences satisfying the conclusion of Lemma 12.

▶ **Lemma 13** (Lemma 9 of [21])**.** *Let $\Phi(n)$ be such that $\Phi(0) = 0$ and for all $n \geq 1$, $\Phi(n) \leq \Phi(n-1) + 1 + \frac{1}{n} \sum_{i=0}^{n-1} \Phi(i)$. Then for all $n \geq 0$, $\Phi(n) \leq e^{2\sqrt{n}}$.*

4 Simple stochastic games with few random nodes

The idea that in an SSG, the optimal strategies depend only on the ordering of the values of RAN-nodes, and not on their actual values, has been introduced by Gimbert and Horn in [15]. Their main idea is that, if one gives an ordering $r_1 r_2 \cdots r_k$ of RAN-nodes such that $\text{Val}_{*,*}(r_i)$ is nondecreasing with i, then MAX will try to reach a node r_i with i as high as possible, whereas MIN will try to minimize this index; this idea is hereafter formalized by the notion of forcing sets and forcing strategies (Section 4.1). Gimbert and Horn use this fact to derive an algorithm that will enumerate all possible orders on RAN-nodes an will identify one with the property mentionned above, yiedling the optimal strategies and values for G.

The algorithm that we describe and analyse in the rest of this paper (Alg. 4) uses the same principle, but iterates through orders in a special way, similarly to the iteration through strategies made by Ludwig's algorithms (see Section 3). We will derive a similar bound for the average number of iterations of this randomized algorithm. Hence, our main algorithm is still a variation on Bland's rule for pivot selection. The difficulty here does not lie in the proof of the bound, but in the description of the technique used to iterate on orders.

In [15], the game remains the same during the execution of the algorithm, but we proceed differently:

- in Section 4.1, we describe how to associate to every total order $t \in \mathcal{T}(k)$ a new SSG $G[t]$, and we show that this game can be solved in polynomial time.
- in Section 4.2, we prove that there is an optimal order $t^* \in \mathcal{T}(k)$ such that the optimal values of $G[t^*]$ give directly the optimal values of G; it is also the order that maximises values of $G[t]$ among all total orders t. If an order t is not optimal, we describe a *pivot* operation yielding from t a new order t' such that the optimal values of $G[t']$ improve those of $G[t]$.
- the proof of the bound will be derived in Section 5.

4.1 Modified game and forcing strategies

We need to assume that the games we consider enjoy some basic properties in order to describe our algorithm without considering too many special cases.

▶ **Definition 14.** *An SSG is in* canonical form *(CF) if* MAX *has a stopping strategy and only* RAN*-nodes can have an outgoing arc to a sink.*

To ensure these conditions, one can first in linear time find and remove all nodes from which MIN player can force the game never to reach neither a sink node nor a RAN-node (see e.g. [1, 11]). These nodes have value 0 and can as well be removed from the game. Then, all probabilities on RAN-nodes are modified by giving them a very small probability to go to a sink. One can prove as in [11] that values remain almost the same. The second condition ensures that all MAX and MIN nodes have to reach a RAN-node in order to reach a sink. It can be done by adding a dummy random node before every sink.

In all that follows we suppose that G is an SSG in CF with random nodes $r_1, r_2, \ldots, r_k$. Let $t \in \mathcal{T}(k)$ be a total order on $[1, k]$. We define a game $G[t]$ as follows (the same construction is presented in [12]). Start with a copy of G. For every $1 \leq i \leq k$, add a MIN-node denoted i to $G[t]$, which we call *control node*; add an arc (i, r_i); for every arc $(x, r_i) \in A$, remove this arc and add an arc (x, i); finally, for every $(i, j) \in t$, $i \neq j$, add the arc (i, j) to $G[t]$.

So basically, every control node $i \in [1, k]$ intercepts all arcs entering in r_i (see Fig. 1), and has an arc to every other control node $j \in [1, k]$ which is greater than i in t. In the game $G[t]$, the set of sinks, MAX-nodes and RAN-nodes remain the same as in G, whereas the set of MIN-nodes will be denoted $V_{\text{MIN}} \cup [1, k]$, where V_{MIN} is the set of MIN-nodes in G. This allows us to directly identify MAX-strategies in $G[t]$ and in G, and to identify projections onto V_{MIN} of MIN-strategies in $G[t]$, to MIN-strategies in G.

Now, suppose we remove first all sinks and random nodes of $G[t]$, and then turn every control node i into a sink with a value equal to its rank in t. This transformation clearly turns $G[t]$ into a game G' without random nodes.

▶ **Definition 15** (Forcing strategy)**.** *By identifying strategies in $G[t]$ and G', we say that any optimal strategy for* MAX *or* MIN *in G' is a* t-forcing strategy *of $G[t]$.*

In t-forcing strategies, the players try to ensure the reaching of a control node as high as possible for MAX, and as low as possible for MIN, in the order t. We refer to [1] and [15] for more details about how one can compute these optimal strategies in linear time, using the so-called *deterministic attractors*.

▶ **Definition 16** (Forcing set)**.** *For any control node $i \in [1, k]$, define the* forcing set for i*, denoted* $\text{FOR}[t](i)$*, as the set of* MAX *and* MIN*-nodes that reach i if the game is played with a couple (σ_t, τ_t) of t-forcing strategies (forcing sets are independant of the choice of the strategies as long as they are t-forcing).*

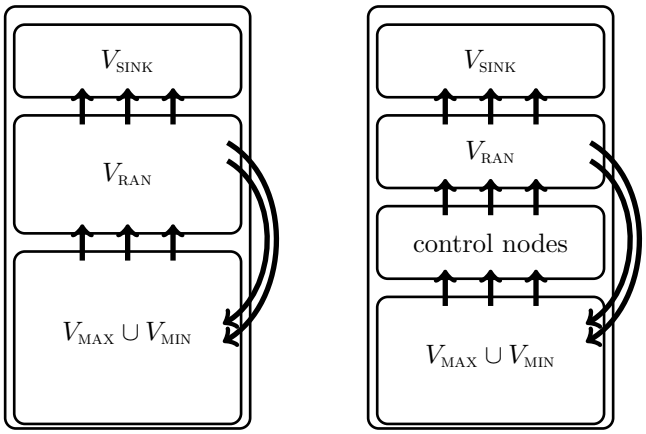
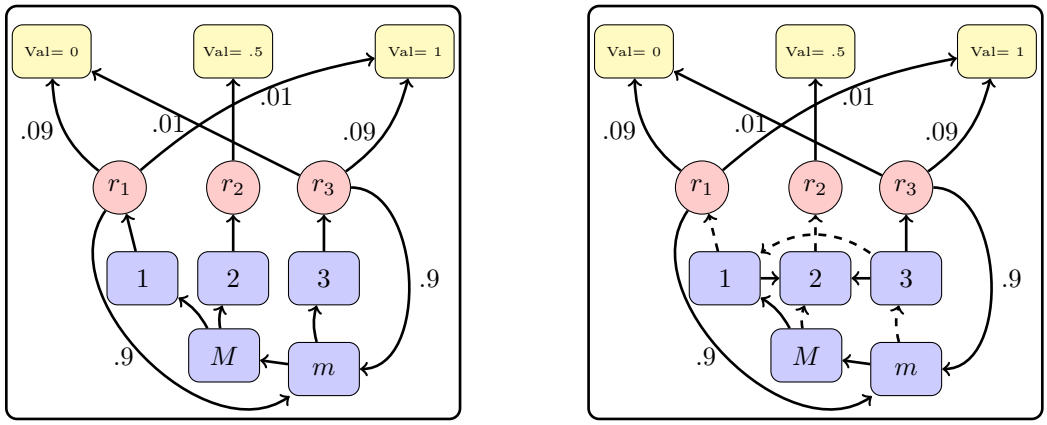


Figure 1 On the left, the structure of a game G in canonical form, only random nodes can directly access sink nodes. On the right, the structure of $G[t]$.

An example of an SSG turned into a modified SSG and of computation of forcing strategies is presented in Fig. 2.



(a) **(b)**

step	total order	forcing strategy for (M, m)	values of RAN-nodes	values of MIN-control nodes
0	$[r_3 r_1 r_2]$	(r_2, r_3)	$.1, .5, .18$	$.1, .5, .1$
1	$[r_1 r_3 r_2]$	(r_2, r_3)	$.46, .5, .54$	$.46, .5, .5$
2	$[r_1 r_2 r_3]$	(r_2, M)	$.46, .5, .54$	$.46, .5, .54$

Figure 2
Fig. (2a): example of an SSG taken from [15] with the additional MIN-control nodes $1, 2, 3$ before each RAN-node. Node m (resp. M) belongs to player MIN(resp. player MAX). Note that we add the dummy RAN-node r_2 so that the game is in CF.
Fig. (2b): solving game $G[t]$ with total order $t = [r_3 r_1 r_2]$. Arcs between control nodes are added according to t. Forcing strategies for m and M, and optimal strategy for nodes m_i are shown with dashed edge. Hence, the forcing set of 1 is $\{1\}$, for 2 it is $\{2, M\}$, and $\{3, m\}$ for 3.
Table: a run of Algorithm 4. The values of the RAN-nodes and the MIN-control nodes are given in the order from left to right.

Here are basic properties on $G[t]$ which should explain why we consider this game.

▶ **Lemma 17.**
(i) *if G has stopping* MAX*-strategy, so does $G[t]$;*
(ii) *optimal values $Val_{*,*}(i)$ of control nodes $i \in [1,k]$ in $G[t]$ are nondecreasing along t;*
(iii) *optimal strategies in $G[t]$ coincide with forcing strategies for order t on $V_{\text{MAX}} \cup V_{\text{MIN}}$;*
(iv) *the game $G[t]$ can be solved in polynomial time.*

Proof. To see why (i) is true, just note that since t is an antisymmetric relation, this does not create new cycles among MIN-nodes.

Suppose now that $(i,j) \in t$. By optimality for the MIN player, and since $G[t]$ is stopping, $\text{Val}_{*,*}(i)$ is the minimum value of $\text{Val}_{*,*}(x)$ for all outneighbours x of i (see Th. 5). Since j is an outneighbour of i in $G[t]$, we have $\text{Val}_{*,*}(i) \leq \text{Val}_{*,*}(j)$. Hence (ii) is true.

Now, consider replacing in $G[t]$ every control node $i \in [1,k]$ by a new sink s_i with value $\text{Val}_{*,*}(i)$. Clearly the values of this new game remain the same. But, by construction of $G[t]$, random nodes have no incoming arcs and they could be as well removed without changing the optimal values on $V_{\text{MAX}} \cup V_{\text{MIN}}$. By reducing the game in this way, we get a deterministic game whose optimal values on $V_{\text{MAX}} \cup V_{\text{MIN}}$ are the same as those of $G[t]$. By definition, optimal strategies of this game are t-forcing strategies, hence (iii) is true.

Finally, to solve $G[t]$ we can choose a couple (σ_t, τ_t) of t-forcing strategies and search for optimal strategies in $G[t]$ that match with (σ_t, τ_t) on $V_{\text{MAX}} \cup V_{\text{MIN}}$. Hence, the strategy of all MAX-nodes is fixed, and only MIN-strategies on control nodes are computed by solving a one player SSG. It can be done in polynomial time by linear programming (see [11]). ◀

As explained in the proof above, to solve $G[t]$, it is enough to compute t-forcing strategies on $V_{\text{MAX}} \cup V_{\text{MIN}}$, which can be done in linear time, and then to solve a one player SSG with only $O(k)$ nodes.

4.2 Value intervals and pivot

In what follows, we write $\text{Val}[t]$ for the vector of optimal values of $G[t]$.

▶ **Definition 18** (Constrained control node)**.** *We say that a control node $i \in [1,k]$ is constrained in $G[t]$ if $Val[t](i) < Val[t](r_i)$.*

Constrained control nodes are similar to switchable nodes in SSG. In fact, we can characterize optimality of an order by the absence of constrained node as follows.

▶ **Lemma 19** (Optimal order)**.** *Let $t \in \mathcal{T}(t)$. The game $G[t]$ does not have any constrained control nodes if and only if the forcing strategies (σ_t, τ_t) are optimal strategies for G. In this case we say that t is an optimal order for G.*

Proof. First note that since G is in CF, σ_t is always stopping.

If $G[t]$ does not have any constrained control nodes, then optimal strategies are the forcing strategies (σ_t, τ_t) on $V_{\text{MAX}} \cup V_{\text{MIN}}$, together with the choice (i, r_i) for each control node $i \in [1,k]$. Then, by merging the control nodes with their associated random node while removing the unused arcs between the control nodes (hence recovering the initial game G), the values on the remaining nodes are kept, and so are the optimality conditions of Th. 5.

If (σ_t, τ_t) are optimal strategies for G, then the values $v_1, v_2, \ldots, v_k$ of the RAN-nodes are nondecreasing along order t. Hence, by turning G into $G[t]$ and extending strategies (σ_t, τ_t) with the choice (i, r_i) for each control node $i \in [1,k]$, we will obtain values that satisfy optimality conditions and such that $Val[t](i) = Val[t](r_i)$, showing that i is not constrained. ◀

We define the *value interval* of a control node $i \in [1,k]$ as the set of $j \in [1,k]$ that share the same optimal value in $G[t]$, i.e. $\mathrm{Val}[t](i) = \mathrm{Val}[t](j)$. This set is indeed an interval in order t by (ii) of Lemma 17, i.e. its elements are consecutive in order t.

▶ **Definition 20.** *The pivot operation on a control node $i \in [1,k]$ for the order t is the transformation of t into a new order $t' \in \mathcal{T}(k)$, obtained by moving i just after the end of its value interval in t.*

Note that if i is the last node of its value interval, then the pivot operation does nothing. Also note that if i is constrained, it cannot be the last node of its value interval (we shall only pivot on constrained control nodes).

Example. Let $k = 7$ and let t be in ascending order $[7,2,4,1,3,6,5]$. Suppose that the values of control nodes are, in this order $[0.2, 0.2, 0.3, 0.3, 0.3, 0.4, 0.4]$. The value intervals are $[7,2]$, $[4,1,3]$ and $[6,5]$. The pivot operation on 4 places 4 after 3, so that the obtained order would be $[7,2,1,3,4,6,5]$.

The following theorem shows that the pivot operation increases the value vector, which will enable us to design a strategy improvement algorithm on the forcing strategies (where the improvement is on $\mathrm{Val}[t]$ rather than on values in the original game G). A similar theorem is proved in [12] to build a different strategy improvement algorithm.

▶ **Theorem 21.** *Let $t \in \mathcal{T}(t)$ and $i \in [1,k]$ be a constrained control node. If t' is obtained from $t \in \mathcal{T}(k)$ by pivoting on i, then $Val[t'] > Val[t]$.*

Proof. Consider a new game $G[t+t']$ which is obtained from G like $G[t]$ and $G[t']$ but with arcs (i,j) for $i \neq j$ between control nodes for all $(i,j) \in t \cup t'$. Let (σ,τ) and (σ',τ') be respective optimal strategies in $G[t]$ and $G[t']$. We can interpret these strategies as strategies in $G[t+t']$. Since the only difference between $G[t]$, $G[t']$ and $G[t+t']$ are the arcs between control nodes, all strategies (σ,τ) give exactly the same values in $G[t]$ and in $G[t+t']$, and a similar observation can be made for $G[t']$. Hence, to prove the result, is is enough to show that $\mathrm{Val}_{\sigma',\tau'} > \mathrm{Val}_{\sigma,\tau}$ in $G[t+t']$. Note that whereas σ and σ' are respective stopping MAX-strategies of $G[t]$ and $G[t']$, they could be not stopping in $G[t+t']$. However it is not difficult to see that conclusion of Th. 9 would still apply. Hence it is sufficient to show in $G[t+t']$ that changing (σ,τ) into (σ',τ') makes a nondecreasing switch on every node, and is increasing in at least one node.

In the order t, let $I = [i, i_1, i_2, \cdots, i_\ell]$, with $\ell \geq 1$ be the increasing sequence of consecutive nodes sharing the same value as i for (σ,τ) (i.e. the value interval of i, starting from i). Since i is constrained, $\ell \geq 1$.

The pivot operation transforms this part of t into $[i_1, i_2, \cdots, i_\ell, i]$, hence the only differences between $G[t]$ and $G[t']$ are the ℓ arcs (i, i_c) for $1 \leq c \leq \ell$ that are inverted into (i_c, i). Hence, if $j \notin I$, it keeps the same position relatively to all other control nodes when we change the order t into t', hence $\textsc{For}[t](j) = \textsc{For}[t'](j)$. Hence, when we change (σ,τ) to (σ',τ'), either there is no switch in $\textsc{For}[t](j)$, or it is between nodes of the same values.

For nodes $x \in \textsc{For}[t](j)$ with $j \in I$, clearly $\sigma'(x)$ (resp. $\tau'(x)$) is in some $\textsc{For}[t](j')$ with $j' \in I$. Since all these nodes also share the same value (by definition of the value interval), these switches are also between nodes of same values.

Suppose that there is a decreasing switch on a control node, i.e. for a $j \in [1,k]$ we have $\mathrm{Val}[t](\tau'(j)) < \mathrm{Val}[t](j)$. In this case $\tau'(j)$ should be stricly before j in t since optimal values are increasing along t. So we could not have $(j, \sigma'(j)) \in t$ but should have $(j, \sigma'(j)) \in t'$. The only possibility is $\sigma'(j) = i$ and $j \in I$. Since these nodes are in the same value interval, once again this switch is unchanging, a contradiction.

We showed that no switch from (σ, τ) to (σ', τ') is decreasing. Now consider the case of i during the pivot operation. Since i is constrained, $\mathrm{Val}[t](i) < \mathrm{Val}[t](r_i)$. Since $\tau'(i)$ can either be equal to r_i or to some j which is stricly after the value interval of i in order t, hence has a greater value, we see that the switch at i must be increasing. ◀

4.3 Main algorithm

Algorithm 4 consists in iterating on orders $t \in \mathcal{T}(k)$, by picking randomly a pivotable element in t and updating t by a pivot on i, until we reach an optimal order.

Here is the **pivot selection rule**. First, prior to the execution of the algorithm, we choose randomly and uniformly an order Θ on the set of all $\frac{k(k-1)}{2}$ unordered pairs of control nodes $\{i, j\}$, with $i, j \in [1, k]$. Then, at each step of the algorithm, consider the game $G[t]$, and remove one by one the arcs between control nodes, following order Θ. During this process, choose as pivot the first constrained control node, if any, which is disconnected from the following nodes of its value interval. In more detail, for a given order t, compute $\mathrm{Val}[t]$ and then partition the control nodes into value intervals. Each constrained control node i has $d(i)$ arcs leading to other control nodes from the same value interval, where $d(i)$ is its distance in t to the last element of this interval. Enumerating Θ in ascending order, the pivot is the first constrained node i whose $d(i)$ arcs are encountered.

Example. Continued from the previous example with $k = 7$ and value intervals $[7, 2]$, $[4, 1, 3]$ and $[6, 5]$. Suppose that the order Θ starts $\{2, 5\}, \{7, 6\}, \{1, 4\}, \{2, 7\} \ldots$. The first element that is disconnected from its value interval is 7 which is the one we choose as a pivot leading to order $[2, 7, 4, 1, 3, 6, 5]$.

Algorithm 4: Iterative version for Bland's rule on random nodes.

input : G SSG, initial total order $t \in \mathcal{T}(k)$.
output : optimal MAX and MIN-strategies.

· pick randomly and uniformly a total order Θ on all pairs of control nodes $\{i, j\}$
· compute values in $G[t]$ (poly. time, see Lemma 17)
while *t is not optimal (Lemma 19)* **do**
  · choose a pivot i by the pivot selection rule
  · update t by pivoting on i as in Def. 20
  · compute values in $G[t]$ (poly. time, see Lemma 17)
return *a couple of forcing strategies for order t in G.*

By Th. 21, no order $t \in \mathcal{T}(k)$ is repeated during the execution of Algorithm 4; since $\mathcal{T}(k)$ is finite, the algorithm reaches in a finite number of steps an order $t^* \in \mathcal{T}(k)$ which has no constrained node, i.e. which is optimal by Lemma 19. Hence, Algorithm 4 computes optimal strategies for G in at most $k!$ steps. However, we claim the following result, which will be proved in the next section.

▶ **Theorem 22.** *Alg. 4 computes optimal strategies for G in at most $e^{\sqrt{2 \cdot k}}$ expected steps.*

Note that for k large enough we have $e^{\sqrt{2 \cdot k}} < k!$, whose growth is roughly equivalent to $2^{k \log k}$. Moreover, the algorithm uses $O(k^2 \log k)$ random bits to choose the order Θ on pairs.

5 Analysis of Algorithm 4

In this section we prove Theorem 22. To do this we shall reformulate Algorithm 4 as a recursive algorithm, but we need additional notions for this. The recursive formulation also

reveals the nature of the algorithm: it computes an optimal order on control nodes by finding the right order between each pair of these nodes using dichotomy. This allows the same analysis as for Ludwig's Algorithm and its variants.

5.1 Modified game $G[p]$ for a pretotal order p

If $p \in \mathcal{P}(k)$ is a pretotal order, we define $G[p]$ exactly as was defined $G[t]$ for a total order $t \in \mathcal{T}(k)$ in Section 4.1. The only difference is that, since p is not total, a control node $i \in [1,k]$ only has arcs to those $j \neq i \in [1,k]$ such that $(i,j) \in p$.

To simplify notation, for any node x in $G[p]$, define $\mathrm{Val}_*[p](x) := \mathrm{Val}^{G[p]}_{*,*}(x)$ as the optimal value of x in $G[p]$. We can now directly extend some of the observations of Lemma 17 to pretotal orders.

▶ **Lemma 23.** *If $p \in \mathcal{P}(k)$, then optimal values of control nodes $i \in [1,k]$ in $G[p]$ are nondecreasing in order p, i.e. if $(i,j) \in p$ then $Val_*[p](i) \leq Val_*[p](j)$.*

In order to solve $G[p]$, the algorithm will recursively compute an optimal total ordering of control nodes $i \in [1,k]$ extending p. Thus, for all total orders $t \in \mathcal{T}(k)$ extending $p \in \mathcal{P}(k)$, we need to assign a value in $G[p]$, which we denote $\mathrm{Val}[p](t)$. Here is how we define it.

▶ **Definition 24.** *Let $t \in \mathcal{T}(k)$ extending $p \in \mathcal{P}(k)$. The values $Val[p](t)$ associated to t in $G[p]$ are the values Val_{σ_t,τ_t} where σ_t and τ_t satisfy:*

(i) *σ_t and τ_t are forcing strategies for $G[t]$;*

(ii) *τ_t statisfies the* MIN*-optimality conditions (Thm. 5) on every control node $i \in [1,k]$.*

As a summary, $\mathrm{Val}_*[p]$ is the vector of optimal values of game $G[p]$ while $\mathrm{Val}[p](t)$ is the vector of optimal values of $G[p]$ when the strategies in V_{MIN} and V_{MAX} are forcing strategies. It follows that $\mathrm{Val}[p](t) \leq \mathrm{Val}_*[p]$. Recall that $\mathrm{Val}[t]$ is the vector of optimal values of $G[t]$. Then we have $\mathrm{Val}[t] = \mathrm{Val}_*[t] = \mathrm{Val}[t](t)$.

▶ **Definition 25** (Optimal order)**.** *Let $p \in \mathcal{P}(k)$ and $t \in \mathcal{T}(k)$ extending p ($p \subset t$). We say that t is an optimal total order for p if $Val_*[p] = Val[p](t)$.*

The next lemma proves the existence and gives a characterization of optimal orders.

▶ **Lemma 26.** *Suppose G is in CF and let $p \in \mathcal{P}(k), t \in \mathcal{T}(k), p \subset t$. Then the following conditions are equivalent:*

(i) *t is an optimal order for p;*

(ii) *t is a nondecreasing ordering of the values $Val_*[p](i)$ for $i \in [1,k]$;*

(iii) *$Val_*[p] = Val[t]$.*

Proof of Lemma 26. First, note that, by definition, optimality conditions are satisfied at control nodes in the definition of $\mathrm{Val}[p](t)$, so it is always true that values $\mathrm{Val}[p](t)(i)$ are nondecreasing along p.

Suppose now that t is optimal for p, i.e. $\mathrm{Val}_*[p] = \mathrm{Val}[p](t)$, and suppose that t is not a nondecreasing ordering of the values $\mathrm{Val}_*[p](i)$ for $i \in [1,k]$. Then there must be two consecutive i, j in order t such that $\mathrm{Val}_*[p](i) > \mathrm{Val}_*[p](j)$, and we must have $(i,j) \in t \setminus p$. Consider the order t' that we obtain from t by inverting j and i. Clearly, this order also extends p since the only inversion between t and t' is (i,j). By an argument similar to the proof of Theorem 21, it is easy to obtain that $\mathrm{Val}[p](t') > \mathrm{Val}[p](t)$, which contradicts optimality. Hence we proved $(i) \Rightarrow (ii)$.

Now suppose (ii). Let (σ^*, τ^*) be an optimal strategy of $G[p]$; we show that optimality conditions are met in $G[t]$. First note that (σ^*, τ^*) has the same values in $G[p]$ and $G[t]$. For the nodes in $V_{\text{MAX}} \cup V_{\text{MIN}}$, no new arcs are added so the optimality conditions are still

satisfied. Now, consider control nodes. An arc $(i,j) \in t \backslash p$ cannot lead to a lower value for i by assumption (ii). Hence the optimality conditions are still satisfied on control nodes, strategies (σ^*, τ^*) are optimal for $G[t]$, so finally $\mathrm{Val}_*[p] = \mathrm{Val}[t]$ and we proved $(ii) \Rightarrow (iii)$.

Since t involves more arcs than p between the control MIN-nodes, we have $\mathrm{Val}[t] \leq \mathrm{Val}[p](t) \leq \mathrm{Val}_*[p]$. Assume (iii), then $\mathrm{Val}_*[p] = \mathrm{Val}[p](t)$. Hence we proved $(iii) \Rightarrow (i)$. ◀

5.2 Recursive formulation

We now give Algorithm 5, a recursive formulation of Algorithm 4. We will prove that these two algorithms compute exactly the same sequence of total orders, and use the recursive formulation to derive a bound.

Algorithm 5: Recursive version for Bland's rule on random nodes.

input : G SSG, order Θ on all pairs $\{i,j\}$ with $i \neq j \in [1,k]$, initial total order t_0 in $\mathcal{T}(k)$ extending a pretotal order $p_0 \in \mathcal{P}(k)$.

output : optimal total order of $G[p_0]$

if p_0 *is total (i.e.* $p_0 = t_0$*)* **then return** t_0

else

- · select according to Θ the last pair $\{i,j\}$ s.t. $(i,j) \in t_0 \setminus p_0$
- · let $p_1 = p_0 + (i,j)$ and $p_2 = p_0 + (j,i)$
- · recursively solve $G[p_1]$ with initial order t_0, giving optimal total order t_1 for $G[p_1]$
- **if** t_1 *is optimal for* $G[p_0]$ *(apply criterium in Lemma 26)* **then return** t_1
- **else**
  - · let t_2 be the total order obtained by pivoting in t_1 along i
  - · recursively solve $G[p_2]$ with initial order t_2, giving optimal order t^* for $G[p_1]$
  - **return** t^*.

▶ **Definition 27.** *Let $p_0 \in \mathcal{P}(k)$ and $\{i,j\} \notin p_0$ such that $p_1 = p_0 + (i,j)$ is still a pretotal order. We say that the addition of (i,j) to p_0 is constraining, or that (i,j) is constrained, if $Val_*[p_1] < Val_*[p_0]$.*

When an arc is constrained, it is essential to the MIN-optimal strategy in $G[p_1]$; in other words removing this arc would increase optimal values.

▶ **Lemma 28.** *Suppose G is in CF and let $p \in \mathcal{P}(k)$, $t \in \mathcal{T}(k)$, $p \subset t$. Then the following conditions are equivalent:*

(i) *t is an optimal ordering for p;*

(ii) *the addition of every arc $(i,j) \in t \setminus p$ to p is not constraining;*

Proof of Lemma 28. Let $p_1 = p_0 + (i,j)$. Since $p \subset p_1 \subset t$, we have $\mathrm{Val}[t] \leq \mathrm{Val}_*[p_1] \leq \mathrm{Val}_*[p]$.

Assume that t is an optimal order for p. If the addition of (i,j) to p is constraining, then $\mathrm{Val}_*[t] \leq \mathrm{Val}_*[p_1] < \mathrm{Val}_*[p]$ which contradicts $\mathrm{Val}[t] = \mathrm{Val}_*[p]$ by Lemma 26. Hence we proved $(i) \Rightarrow (ii)$.

Assume that no arc in $t \backslash p$ is constraining. Then, add sequentially arcs in $t \backslash p$ to $G[p]$ until we get $G[t]$, hence forming a sequence of games $G[p] = G[p_0], G[p_1], \ldots, G[t]$. If none of these arcs is used in $G[t]$ then $\mathrm{Val}[t] = \mathrm{Val}_*[p]$ and (i) is proved. Otherwise consider the first

arc (i,j) such that $\mathrm{Val}_*[p_\ell + (i,j)] < \mathrm{Val}_*[p_\ell]$. This implies that $\mathrm{Val}_*[p_\ell](j) < \mathrm{Val}_*[p_\ell](i)$. But $\mathrm{Val}_*[p] = \mathrm{Val}_*[p_\ell]$ since no constraining arc has been added until step ℓ. Hence $\mathrm{Val}_*[p](j) < \mathrm{Val}_*[p](i)$, and finally (i,j) is constraining for p, a contradiction. ◀

Consider a run of the recursive algorithm and let t be a total order at any step of the run. Let us inspect the first time where t is modified. Order t will be optimal for a sequence of pretotal orders that are obtained from t by removing one by one pairs in order Θ as long as they are not constrained. This in fact amounts to ascending the recursive call tree. Let (i,j) be the first constrained pair and p_0 the pretotal order obtained once (i,j) is removed. Then t is turned into t' by pivoting on node i as we did in iterative Alg. 4. We show now that control node i is same as the pivot selected by the pivot selection rule.

Note first that, during the process of removing the pairs one by one in order Θ, the value intervals of $G[t]$ are kept unchanged until the pretotal order $p_0 + (i,j)$ is reached. Since removing (i,j) implies an increase of the optimal values, it means that i and j were in the same value interval and that i had no other neighbour in that interval. Note here that, as a consequence, $p_0 + (j,i)$ is then guaranteed to be a pretotal order. Clearly, node i is the first control node in that situation. So the choice of node i exactly obeys the pivot selection rule.

Finally, the following lemma enables us to analyze the complexity of Algorithm 5.

▶ **Lemma 29.** *Let $p_0 \in \mathcal{P}(k)$ and $\{i,j\} \notin p_0$ such that $p_1 = p_0 + (i,j)$ and $p_2 = p_0 + (j,i)$ are pretotal orders and where the addition of (i,j) to p_0 is constraining. Let t_1 be an optimal total order for p_1, t_2 obtained from t_1 by pivoting in i, and let t_2^* be an optimal total order for p_2.*

Let (i_1,j_1) such that $Val_*[p_0 + (i,j)] \not\leq Val_*[p_0 + (i_1,j_1)]$. Then for any total order t obtained by Algorithm 5 between t_1 and t_2^* (including those), one has $(j_1,i_1) \in t$.

Proof. Suppose that $(i_1,j_1) \in t$. Then $t \supset p_0 + (i_1,j_1)$ hence $\mathrm{Val}[t] \leq \mathrm{Val}_*[p_0 + (i_1,j_1)]$.

On the other hand, since the pivot operation is increasing values, we have $\mathrm{Val}_*[p_0+(i,j)] = \mathrm{Val}[t_1] < \mathrm{Val}[t_2] \leq \mathrm{Val}[t]$, so $\mathrm{Val}_*[p_0 + (i,j)] \leq \mathrm{Val}_*[p_0 + (i_1,j_1)]$, a contradiction. ◀

Using this result, the proof for the complexity bound is the same as the proof of Theorem 10 using the recursive formulation. Let $f^\Theta(G, p_0, t_0)$ be the total number of pivots performed by Algorithm 5 on input G, p_0, t_0 for an order Θ on pairs.

Now define $\Phi(m) = \sup_{G,p_0,t_0} \mathbb{E}^\Theta\left[f^\Theta(G,p_0,t_0]\right)$ where the supremum is taken over all games G, pretotal orders p_0 and total orders t_0 extending p_0 such that $t_0 \setminus p_0$ is of size at most m. The expectation is taken over all possible uniform choices for Θ.

Then by Lemma 29, $\Phi(m)$ will satisfy Lemma 12, hence the claimed bound of Th. 22 by Lemma 13 since the depth of the recursive tree is at most $\frac{k(k-1)}{2}$.

References

1. Daniel Andersson, Kristoffer Arnsfelt Hansen, Peter Bro Miltersen, and Troels Bjerre Sørensen. Deterministic graphical games revisited. In *Conference on Computability in Europe*, pages 1–10. Springer, 2008.
2. Daniel Andersson and Peter Bro Miltersen. The complexity of solving stochastic games on graphs. In *International Symposium on Algorithms and Computation*, pages 112–121. Springer, 2009.
3. David Auger, Pierre Coucheney, and Yann Strozecki. Finding optimal strategies of almost acyclic simple stochastic games. In *International Conference on Theory and Applications of Models of Computation*, pages 67–85. Springer, 2014.
4. Robert G Bland. New finite pivoting rules for the simplex method. *Mathematics of operations Research*, 2(2):103–107, 1977.

5 Cristian S Calude, Sanjay Jain, Bakhadyr Khoussainov, Wei Li, and Frank Stephan. Deciding parity games in quasipolynomial time. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 252–263. ACM, 2017.

6 Krishnendu Chatterjee, Luca de Alfaro, and Thomas A Henzinger. Termination criteria for solving concurrent safety and reachability games. In *Proceedings of the twentieth annual ACM-SIAM symposium on Discrete algorithms*, pages 197–206. SIAM, 2009.

7 Krishnendu Chatterjee and Nathanaël Fijalkow. A reduction from parity games to simple stochastic games. In *GandALF*, pages 74–86, 2011.

8 Taolue Chen, Vojtěch Forejt, Marta Kwiatkowska, David Parker, and Aistis Simaitis. Automatic verification of competitive stochastic systems. *Formal Methods in System Design*, 43(1):61–92, 2013.

9 Taolue Chen, Marta Kwiatkowska, Aistis Simaitis, and Clemens Wiltsche. Synthesis for multi-objective stochastic games: An application to autonomous urban driving. In *International Conference on Quantitative Evaluation of Systems*, pages 322–337. Springer, 2013.

10 Anne Condon. On Algorithms for Simple Stochastic Games. In *Advances in computational complexity theory*, pages 51–72, 1990.

11 Anne Condon. The complexity of stochastic games. *Information and Computation*, 96(2):203–224, 1992.

12 Decheng Dai and Rong Ge. New results on simple stochastic games. In *International Symposium on Algorithms and Computation*, pages 1014–1023. Springer, 2009.

13 Kousha Etessami and Mihalis Yannakakis. On the complexity of Nash equilibria and other fixed points. *SIAM Journal on Computing*, 39(6):2531–2597, 2010.

14 Oliver Friedmann. An exponential lower bound for the parity game strategy improvement algorithm as we know it. In *Logic In Computer Science, 2009. LICS'09. 24th Annual IEEE Symposium on*, pages 145–156. IEEE, 2009.

15 Hugo Gimbert and Florian Horn. Simple stochastic games with few random vertices are easy to solve. In *Foundations of Software Science and Computational Structures*, pages 5–19. Springer, 2008.

16 Nir Halman. Simple stochastic games, parity games, mean payoff games and discounted payoff games are all LP-type problems. *Algorithmica*, 49(1):37–50, 2007.

17 Thomas Dueholm Hansen and Uri Zwick. An improved version of the Random-Facet pivoting rule for the simplex algorithm. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 209–218. ACM, 2015.

18 Alan J Hoffman and Richard M Karp. On nonterminating stochastic games. *Management Science*, 12(5):359–370, 1966.

19 Rasmus Ibsen-Jensen and Peter Bro Miltersen. Solving simple stochastic games with few coin toss positions. In *European Symposium on Algorithms*, pages 636–647. Springer, 2012.

20 Gil Kalai. A subexponential randomized simplex algorithm. In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, pages 475–482. ACM, 1992.

21 Walter Ludwig. A subexponential randomized algorithm for the simple stochastic game problem. *Information and computation*, 117(1):151–155, 1995.

22 Lloyd S Shapley. Stochastic games. *Proceedings of the National Academy of Sciences of the United States of America*, 39(10):1095, 1953.

23 Colin Stirling. Bisimulation, modal logic and model checking games. *Logic Journal of IGPL*, 7(1):103–124, 1999.

24 Rahul Tripathi, Elena Valkanova, and VS Anil Kumar. On strategy improvement algorithms for simple stochastic games. *Journal of Discrete Algorithms*, 9(3):263–278, 2011.