

# A subexponential lower bound for the Random Facet algorithm for Parity Games

Oliver Friedmann\*

Thomas Dueholm Hansen<sup>†</sup>

Uri Zwick<sup>‡</sup>

## Abstract

*Parity Games* form an intriguing family of infinite duration games whose solution is equivalent to the solution of important problems in automatic verification and automata theory. They also form a very natural subclass of *Deterministic Mean Payoff Games*, which in turn is a very natural subclass of turn-based *Stochastic Mean Payoff Games*. It is a major open problem whether these game families can be solved in polynomial time.

The currently theoretically fastest algorithms for the solution of all these games are adaptations of the randomized algorithms of Kalai and of Matoušek, Sharir and Welzl for *LP-type* problems, an abstract generalization of linear programming. The expected running time of both algorithms is *subexponential* in the size of the game, i.e.,  $2^{O(\sqrt{n \log n})}$ , where  $n$  is the number of vertices in the game. We focus in this paper on the algorithm of Matoušek, Sharir and Welzl and refer to it as the *Random Facet* algorithm. Matoušek constructed a family of abstract optimization problems such that the expected running time of the Random Facet algorithm, when run on a random instance from this family, is close to the subexponential upper bound given above. This shows that in the abstract setting, the  $2^{O(\sqrt{n \log n})}$  upper bound on the complexity of the Random Facet algorithm is essentially tight.

It is not known, however, whether the abstract optimization problems constructed by Matoušek correspond to games of any of the families mentioned above. There was some hope, therefore, that the Random Facet algorithm, when applied to, say, parity games, may run in polynomial time. We show, that this, unfortunately, is not the case by constructing explicit parity games on which the expected running time of the Random Facet algorithm is close to the subexponential upper bound. The games we use mimic the behavior of a *randomized counter*. They are also the first *explicit LP-type* problems on which the Random Facet algorithm is not polynomial.

## 1 Introduction

Parity games are infinite-duration full-information two-player games played on finite graphs with integer priorities assigned to their vertices. The two players, called *player 0* and *player 1*, construct an infinite path in the game graph. Player 0 wins if the largest priority that appears an infinite number of times on the path is *even*.

Player 1 *wins* otherwise. For a more detailed and formal definition, see the next section.

The problem of *solving* a parity game, i.e., determining which of the two players has a *winning strategy*, is known to be equivalent to the problems of complementation of  $\omega$ -tree automata, and to the problem of  $\mu$ -calculus model checking. (See [EJ91],[EJS93],[Sti95],[GTW02].)

Parity games are also interesting as they form a very special subclass of *Deterministic Mean Payoff Games* (see [EM79], [GKK88], [ZP96]), which in turn form a very special subclass of turn-based *Stochastic Mean Payoff Games* (see [Con92],[AM09]). For the reduction from parity games to mean payoff games, see [Pur95]. While it is known that the decision problems corresponding to these games belong to  $\text{NP} \cap \text{co-NP}$ , and even to  $\text{UP} \cap \text{co-UP}$  ([Jur98]), it is a major open problem whether any of these game families can be solved in polynomial time.

There are various exponential time algorithms for solving parity games (see, e.g., [Zie98],[Jur00]). It was hoped, until recently, that the *policy iteration* algorithms of Vöge and Jurdziński [VJ00] or Schewe [Sch08], which behave extremely well in practice (see [FL09]), would solve parity games in polynomial time. Friedmann [Fri09] has recently shown, however, that this is not the case by exhibiting parity games on which these policy iteration algorithms take exponential time. The theoretically fastest deterministic algorithm currently known for solving parity games runs in  $2^{O(\sqrt{n \log n})}$  time, where  $n$  is the number of vertices in the game (see [JPZ08]). No such *deterministic* subexponential algorithms are known for the more general classes of deterministic or stochastic mean payoff games.

Slightly (theoretically) faster randomized algorithms for the solution of parity games can be obtained by adapting the algorithms of Kalai [Kal92, Kal97] and of Matoušek, Sharir and Welzl [MSW96] for solving *LP-type* problems, an abstract generalization of linear programming problems. The first to note the applicability of these abstract optimization algorithms to the families of games considered here was Ludwig [Lud95]. Petersson and Vorobyov [PV01] and Björklund *et al.* [BSV03],[BV05],[BV07] adapted these algorithms to work on parity games and mean payoff games. Halman [Hal07] gave a formal proof that the optimization

\*Department of Computer Science, University of Munich, Germany. E-mail: [Oliver.Friedmann@gmail.com](mailto:Oliver.Friedmann@gmail.com).

<sup>†</sup>Department of Computer Science, Aarhus University, Denmark. Supported by the Center for Algorithmic Game Theory, funded by the Carlsberg Foundation. E-mail: [tdh@cs.au.dk](mailto:tdh@cs.au.dk).

<sup>‡</sup>School of Computer Science, Tel Aviv University, Israel. Research supported by ISF grant no. 1306/08. E-mail: [zwick@tau.ac.il](mailto:zwick@tau.ac.il).

problems involved in solving all these type of games are of LP-type. The best known upper bound on the expected running time of these algorithms on  $n$ -vertex parity games is  $2^{O(\sqrt{n \log n})}$ . (The number of *vertices* in a game corresponds to the *combinatorial dimension* of the equivalent LP-type problem. The number of *edges* corresponds to the number of *constraints*.)

The randomized algorithms of Kalai [Kal92, Kal97] and of Matoušek, Sharir and Welzl [MSW96] are the fastest known *combinatorial* algorithms for solving linear programs. For a discussion of the relation between these algorithms, see Goldwasser [Gol95]. It is a major open problem whether their expected running times on linear programs is polynomial or not.

Matoušek [Mat94] constructed a family of abstract optimization problems such that the Random Facet algorithm, i.e., the algorithm of Matoušek, Sharir and Welzl [MSW96], when run on a *random* instance from the family, has an expected running time close to the subexponential upper bound given above. It is known, however, that the hard instances in this family do not correspond to linear programs (see Gärtner [Gär02]), and they are also unlikely to correspond, say, to parity game instances.

We construct explicit parity games such that the expected running time of the Random Facet algorithm on an  $n$ -vertex game is  $2^{\tilde{\Omega}(\sqrt{n})}$ , where  $\tilde{\Omega}(f) = \Omega(f / \log^c n)$ , for some constant  $c$ . This matches, up to a polylogarithmic factor in the exponent, the upper bound given in [MSW96]. Our games also provide the first *explicit* construction of LP-type problems on which the Random Facet algorithm is not polynomial.

Our lower bound is obtained by adapting and extending techniques used by Friedmann [Fri09] to show that the *deterministic* policy iteration algorithm for parity games may take exponential time. Friedmann's [Fri09] (and Fearnley's [Fea10]) delicate constructions are designed to fool a deterministic algorithm. Fooling a randomized algorithm like the Random Facet algorithm poses new challenges. Many difficulties, thus, need to be overcome to obtain a tight subexponential lower bound for the Random Facet algorithm. Instead of simulating a standard counter, runs of the Random Facet algorithm on our parity games essentially simulate the operation of a *randomized counter*. To ensure, with high probability, the desired behavior of the randomized counter, we *duplicate* critical gadgets used in the construction, thus achieving resilience against 'fortunate' choices of the Random Facet algorithm.

With hind sight, it could be said that the fact that the Random Facet algorithm for solving parity games is not polynomial is not that surprising. This should be contrasted, however, with the fact that some people are still hopeful that the Random Facet algorithm, when applied to linear programming problems, does run in

polynomial time (see, e.g., Gärtner [Gär02]). In any case, we believe that resolving the complexity of the Random Facet algorithm, the algorithm for solving parity games with the best known upper bound, is a significant contribution.

The rest of the paper is organized as follows. In the next section we define parity games and the other families of games considered in this paper. In Section 3 we give a general overview of the family of policy iteration algorithms to which the Random Facet algorithm belongs. In Section 4 we describe the adaptation of the Random Facet algorithm to parity games and other families of games. In Sections 5 and 6 we give a high level, and then a complete description, of the parity games used to obtain our lower bound. In Section 7 and 8 we analyze the expected running time of the Random Facet algorithm on these games and show that it is of the form  $2^{\tilde{\Omega}(\sqrt{n})}$ . We end in Section 9 with some concluding remarks and open problems. To enhance the readability of the paper, some the technical proofs are deferred to appendices.

## 2 Parity games and payoff games

A *parity game* is a tuple  $G = (V_0, V_1, E, \Omega)$ , where  $V_0$  is the set of vertices controlled by player 0,  $V_1$  is the set of vertices controlled by player 1,  $E \subseteq V \times V$ , where  $V = V_0 \cup V_1$ , is the set of edges, and  $\Omega : V \rightarrow \mathbb{N}$  is a function that assigns a *priority* to each vertex. We let  $E_i = E \cap (V_i \times V)$ , for  $i = 0, 1$ , denote the edges emanating from  $V_i$ . We assume that each vertex has at least one outgoing edge.

The two players, called 0 and 1, construct an infinite path in the game graph, called a *play*, as follows. They start at an initial vertex  $v_0 \in V$ . If the path constructed so far is  $v_0 v_1 \dots v_n$ , and  $v_n \in V_i$ , then player  $i$  selects a vertex  $w \in V$  such that  $(v_n, w) \in E$  and the play continues with  $v_0 \dots v_n w$ . Every play has a unique winner given by the *parity* of the largest priority that occurs infinitely often. Player 0 wins if this priority is even, and player 1 wins if it is odd.

A (positional) *strategy*  $\sigma$  for player  $i$  is a subset  $\sigma \subseteq E_i$  such that each vertex  $v \in V_i$  has exactly one outgoing edge in  $\sigma$ . A play  $v_0 v_1 \dots$  *conforms* to a strategy  $\sigma$  for player  $i$  if whenever  $v_j \in V_i$ , then  $(v_j, v_{j+1}) \in \sigma$ . A strategy  $\sigma$  for player  $i$  is a *winning strategy*, from  $v_0$ , if player  $i$  wins every play that begins at  $v_0$  and conforms to  $\sigma$ .

Parity games are known to be *determined*, i.e., from any vertex  $v$ , exactly one of the players has a winning positional strategy (see [EJ91]). We let  $W_i \subseteq V$ , for  $i = 0, 1$ , be a set of vertices from which player  $i$  has a winning strategy. Furthermore it is known that player  $i$  has a single strategy  $\sigma_i$  that is winning for  $i$  from all vertices of  $W_i$ . *Solving* a parity game means

determining the sets  $W_0$  and  $W_1$  and a pair of winning strategies  $\sigma_0$  and  $\sigma_1$  for both players.

A *Deterministic Mean Payoff Game* (DMPG) is a tuple  $G = (V_0, V_1, E, r)$ , where  $V_0$  and  $V_1$  are again the sets of vertices controlled by players 0 and 1 respectively,  $E \subseteq V \times V$ , where  $V = V_0 \cup V_1$ , is the set of edges, and  $r : E \rightarrow \mathbb{R}$  is an immediate *reward* function defined on the edges. The two players again construct an infinite path  $v_0 v_1 \dots$  in the game graph  $(V_0 \cup V_1, E)$ . The outcome of a play, paid by player 1 to player 0, is then  $\liminf_{n \rightarrow \infty} \frac{1}{n} \sum_{j=0}^{n-1} r(v_j, v_{j+1})$ . Player 0 tries to *maximize* this outcome, while player 1 tries to *minimize* it. The two players are thus trying to maximize or minimize the limiting *average reward per turn*.

Every vertex  $v$  in a DMPG has a *value*  $val(v)$ , such that player 0 has a strategy  $\sigma_0$  that ensures that the outcome of every play that starts at  $v$  and conforms to  $\sigma_0$  is at least  $val(v)$ , while player 1 has a strategy  $\sigma_1$  that ensures that the outcome of every play that starts at  $v$  and conforms to  $\sigma_1$  is at most  $val(v)$ . (In particular, the outcome of the unique play that conforms to both  $\sigma_0$  and  $\sigma_1$  is exactly  $val(v)$ .) Such strategies are said to be optimal strategies from  $v$ . Again, both players have optimal strategies that are optimal from all vertices.

A *Deterministic Discounted Payoff Game* (DDPG) is similar to a DMPG. The only difference is that the outcome of a play is now defined to be  $(1 - \lambda) \sum_{j=0}^{\infty} \lambda^j r(v_j, v_{j+1})$ , where  $0 < \lambda < 1$  is a *discount factor*. Discount factors have interesting economical interpretations. Discounted games are slightly simpler to work with, as the discount factor ensures the convergence of the sum  $\sum_{j=0}^{\infty} \lambda^j r(v_j, v_{j+1})$ , no matter what the players do. When  $\lambda \rightarrow 1$ , the value of the discounted game tends to the value of the mean payoff game.

A parity game  $G = (V_0, V_1, E, \Omega)$  can be easily reduced into a DMPG  $G' = (V_0, V_1, E, r)$ , where for every  $(u, v) \in E$  we have  $r(u, v) = (-n)^{\Omega(u)}$ , where  $n = |V|$ . Player 0 has a winning strategy from vertex  $v$  in  $G$  if and only if she has a strategy that guarantees a *positive* outcome starting from  $v$  in  $G'$ . (The correctness of the reduction follows from the fact that a cycle in the game graph has a positive mean value if and only if the largest priority on one of its vertices is even.) Parity games thus form a very well-behaved subfamily of DMPGs.

Turn-based *Stochastic Mean Payoff Game* (SMPG) form an even more general family of games. A SMPG is a tuple  $G = (V_0, V_1, V_R, E, p, r)$ , where  $V_0$  and  $V_1$  are the sets of vertices controlled by players 0 and 1,  $V_R$  is a set of randomization vertices, controlled by nature,  $p : E_R \rightarrow [0, 1]$  is a function that assigns a probability to each edge of  $E_R = E \cap (V_R \times V)$ , and  $r : E \rightarrow \mathbb{R}$  is a function that assigns an immediate reward to each edge of the graph. For every  $u \in V_R$  we have  $\sum_{(u,v) \in E} p(u, v) = 1$ . The two players again construct an infinite path in the game graph. When the last

vertex reached is in  $V_i$ , where  $i \in \{0, 1\}$ , player  $i$  chooses the next edge. When the last vertex reached is in  $V_R$ , the next edge is chosen according to the probabilities specified by  $p$ . The two players try to maximize, respectively minimize, the long term *expected* average reward per turn. Turn-based *Stochastic Discounted Payoff Game* (SDPG) are defined similarly.

It is a tantalizing open problem whether parity games and the other families of games defined here can be solved in polynomial time. The Random Facet algorithm was considered to be a strong candidate for solving these games in polynomial time. We show that this, unfortunately, is not the case, even for parity games, the easiest of these families.

### 3 Policy iteration algorithms

In this section we describe a family of algorithms that can be used to solve the various games defined in the previous section. The RANDOMFACET algorithm, described in the next section, is a member of this family. We start by explaining how these algorithms work on DDPGs and then explain how to adapt them for the other families of games.

Policy iteration (also known as *strategy improvement*) algorithms were first developed by Howard [How60] who used them to solve infinite-horizon *Markov Decision Processes* (MDPs), a one-player variant of the stochastic two-player games defined in the previous section. They were adapted for the solution of two-player games by many researchers. (See, e.g., [HK66],[Con92],[Pur95].)

Let  $G = (V_0, V_1, E, r)$  be a DDPG game with discount factor  $\lambda$ . Let  $\sigma$  and  $\tau$  be policies for players 0 and 1. For every starting vertex  $v_0$ , there is a single play, i.e., infinite path,  $v_0 v_1 \dots$  that conforms to both  $\sigma$  and  $\tau$ . The value  $(1 - \lambda) \sum_{j=0}^{\infty} \lambda^j r(v_j, v_{j+1})$  of this play is defined to be the *valuation*  $val_{\sigma, \tau}(v_0)$  of these pair of strategies on  $v_0$ .

For a given game  $G$  and a strategy  $\sigma$  of player 0, we let  $\tau_{G, \sigma}$  be an *optimal counter-strategy* for player 1, i.e., a strategy  $\tau$  that minimizes  $val_{\sigma, \tau}(v)$ , for every  $v \in V$ . Such optimal counter-strategies are known to exist and they can be found in polynomial time (see, e.g. [MTZ10]). We let  $val_{\sigma}(v_0) = val_{\sigma, \tau_{G, \sigma}}(v_0)$ , for every  $v_0 \in V$ , be the valuation of  $\sigma$  on  $v_0$ , and let  $val_{\sigma} = (val_{\sigma}(v_1), val_{\sigma}(v_2), \dots, val_{\sigma}(v_n))$  be the *valuation vector* of  $\sigma$ , where  $v_1, v_2, \dots, v_n$  is some fixed ordering of the vertices of  $V_0$ .

Let  $x, y \in \mathbb{R}^n$ . We say that  $x \trianglelefteq y$  if and only if  $x_j \leq y_j$ , for every  $1 \leq j \leq n$ . We say that  $x \triangleleft y$  if and only if  $x \trianglelefteq y$  and  $x \neq y$ . Let  $\sigma$  and  $\sigma'$  be two strategies of player 0. In general, the valuation vectors  $val_{\sigma}$  and  $val_{\sigma'}$  may be incomparable. If  $val_{\sigma} \triangleleft val_{\sigma'}$  we say that  $\sigma'$  is better than  $\sigma$ .

If  $\sigma$  is a strategy for player 0 and  $e = (u, v) \notin \sigma$ , we

let  $\sigma[e]$  be the strategy obtained by replacing the edge emanating in  $\sigma$  from  $u$  by  $e$ . (I.e.,  $\sigma[e] = \sigma \cup \{e\} \setminus \{e'\}$ , where  $e' = (u, v') \in \sigma$ .) It can be shown that the valuations  $val_\sigma$  and  $val_{\sigma[e]}$  are always comparable, i.e., either  $val_{\sigma[e]} \preceq val_\sigma$  or  $val_\sigma \preceq val_{\sigma[e]}$ . If  $val_\sigma \triangleleft val_{\sigma[e]}$ , we say that  $e$  is an *improving switch* with respect to  $\sigma$ .

If  $e_1, e_2, \dots, e_k \notin \sigma$  are edges emanating from different vertices, we let  $\sigma[e_1, \dots, e_k] = \sigma[e_1] \dots [e_k]$  be the strategy obtained by performing the *multi-switch*  $\{e_1, e_2, \dots, e_k\}$ , i.e., by switching each one of  $e_1, e_2, \dots, e_k$ . We say that  $\{e_1, e_2, \dots, e_k\}$  is an improving multi-switch if and only if  $val_\sigma \triangleleft val_{\sigma[e_1, \dots, e_k]}$ . Policy iteration algorithms are based on the following fundamental theorem. (See, e.g., [How60],[Con92].)

**THEOREM 3.1.** (i) *If  $\sigma$  is not an optimal strategy, then there exists an improving switch  $e \notin \sigma$ .*

(ii) *If  $e_1, e_2, \dots, e_k \notin \sigma$  are improving switches, then  $\{e_1, e_2, \dots, e_k\}$  is an improving multi-switch.*

A policy iteration algorithm is an algorithm that starts with some initial strategy  $\sigma_0$  of player 0 and constructs a sequence of strategies  $\sigma_0, \sigma_1, \dots, \sigma_\ell$ , such that  $val_{\sigma_0} \triangleleft val_{\sigma_1} \triangleleft \dots \triangleleft val_{\sigma_\ell}$  and such that  $\sigma_\ell$  is an optimal strategy for player 0. Strategy  $\sigma_{i+1}$  is obtained from  $\sigma_i$  by performing an improving (multi-)switch. Different policy iteration algorithms differ in the way they choose the improving switches performed in each iteration.

An appealing feature of policy iteration algorithms is that determining whether  $e$  constitutes an improving switch with respect to  $\sigma$  can be done *without* evaluating  $\sigma[e]$ . (Evaluating  $\sigma[e]$  is a non-trivial operation as it involves finding an optimal counter-strategy.) An edge  $e = (u, v) \notin \sigma$  of a DDPG is an improving switch if and only if  $(1 - \lambda)r(u, v) + \lambda val_\sigma(v) > val_\sigma(u)$ .

Let  $e_0 = (u, v_0), e_1 = (u, v_1), \dots, e_d = (u, v_d)$  be the edges emanating from  $u \in V_0$  and suppose that  $e_0 \in \sigma$ . Note that several of  $e_1, e_2, \dots, e_d$  may be improving switches. The *standard* policy iteration algorithm selects for each vertex  $u$  the edge  $e_i$  that maximizes  $(1 - \lambda)r(u, v_i) + \lambda val_\sigma(v_i)$ . It then performs the corresponding multi-switch. The standard policy iteration algorithm behaves very well in practice. It was recently shown, however, that its worst-case running time is exponential for parity games (Friedmann [Fri09]) and for MDPs (Fearnley [Fea10]).

For a given set of player 0 edges  $F \subseteq E_0$  such that for every node  $v \in V_0$  there is at least one edge originating from  $v$  in  $F$ , let  $G_F = (V_0, V_1, F \cup E_1, \Omega)$  be the subgame of  $G$  in which only the edges of  $F$  are available for player 0. Let  $\sigma_{G_F} \subseteq F$  denote an (arbitrary but fixed) optimal strategy in the game  $G_F$ ; In particular, let  $\sigma_G$  denote an optimal strategy in the game  $G$ .

We next move from discounted payoff games to mean payoff games. Let  $\sigma$  and  $\tau$  be strategies of players 0

and 1 in a DMPG. Let  $v_0 v_1 \dots$  be the infinite path that conforms to  $\sigma$  and  $\tau$ . We define the *value*  $val_{\sigma, \tau}(v_0)$  of this play to be  $\lim_{n \rightarrow \infty} \frac{1}{n} \sum_{j=0}^{n-1} r(v_j, v_{j+1})$ . (Note the difference between the *value*, and the *valuation* defined below.) The infinite path  $v_0 v_1 \dots$  is composed of finite path  $P$  leading to a cycle  $C = u_0 u_1 \dots u_k$ , where  $u_k = u_0$ , which is repeated an infinite number of times, and  $val_{\sigma, \tau}(v_0)$  is simply the average reward  $\frac{1}{k} \sum_{j=0}^{k-1} r(u_j, u_{j+1})$  of the cycle  $C$ .

We can define improving switches with respect to the values defined above. Unfortunately, in the non-discounted case, it is not the case that if  $\sigma$  is not optimal then there exist at least one switch that *strictly* improves the value. To remedy the situation, we define *potentials* as follows. Let  $u = u(C)$  be a fixed vertex on the cycle  $C$ , e.g., the vertex with the smallest index. Let  $v_0 v_1 \dots v_\ell = u$  be the finite prefix of the infinite path  $v_0 v_1 \dots$  that ends with the first visit to  $u$ . We define the *potential*  $POT_{\sigma, \tau}(v_0)$  to be  $\sum_{j=0}^{\ell-1} (r(v_j, v_{j+1}) - val_{\sigma, \tau}(v_0))$ , i.e., the total reward on the path leading to  $u$ , when  $val_{\sigma, \tau}(v_0)$  is subtracted from the reward of each edge. Finally, we define the *valuation*  $val_{\sigma, \tau}(v_0)$  to be the pair  $(val_{\sigma, \tau}(v_0), POT_{\sigma, \tau}(v_0))$ . We compare valuations *lexicographically*, i.e.,  $val_{\sigma, \tau}(v_0) \prec val_{\sigma', \tau'}(v_0)$  if and only if  $val_{\sigma, \tau}(v_0) < val_{\sigma', \tau'}(v_0)$ , or  $val_{\sigma, \tau}(v_0) = val_{\sigma', \tau'}(v_0)$  and  $POT_{\sigma, \tau}(v_0) < POT_{\sigma', \tau'}(v_0)$ . We again define  $val_\sigma = (val_\sigma(v_1), val_\sigma(v_2), \dots, val_\sigma(v_n))$ . With these slightly more complicated valuations, Theorem 3.1 becomes valid again, and policy iteration algorithms can therefore be used to solve DMPGs.

Finally, we consider parity games. Let  $\sigma$  and  $\tau$  be strategies of players 0 and 1 in a DMPG. Let  $v_0 v_1 \dots$  be the infinite path that conforms to  $\sigma$  and  $\tau$ . Let  $C = u_0 u_1 \dots u_k$ , where  $u_k = u_0$  be the cycle repeated an infinite number of times, and let  $u = u(C)$  be a fixed vertex on the cycle  $C$ , e.g., the vertex with the smallest index. Let  $v_0 v_1 \dots v_\ell = u$  be the prefix leading to  $u$ . Following Vöge and Jurziński [VJ00] (see also [Vög00]), we define the valuation  $val_{\sigma, \tau}(v_0)$  to be a triplet  $(p, A, \ell)$ , where  $p$  is the largest priority on  $C$ ,  $A$  is the multiset of priorities greater than  $p$  appearing on the path  $v_0 v_1 \dots v_\ell = u$ , and  $\ell$  is the length of this path. We define a total ordering on valuations as follows:  $(p_1, A_1, \ell_1) \prec (p_2, A_2, \ell_2)$  if and only if one of the following conditions hold:

- (i)  $(-1)^{p_1} p_1 < (-1)^{p_2} p_2$ .
- (ii)  $p_1 = p_2$ ,  $A_1 \neq A_2$  and the largest element in  $A_1 \oplus A_2$ , the symmetric difference, is even and belongs to  $A_2$ , or is odd and belongs to  $A_1$ .
- (iii)  $p_1 = p_2$ ,  $A_1 = A_2$  and  $p_1$  is even and  $\ell_1 > \ell_2$ , or  $p_1$  is odd and  $\ell_1 < \ell_2$ .

With these valuations, policy iteration algorithm can be applied directly on parity games.

Let  $G = (V_0, V_1, E, \Omega)$  be a parity game. We say that  $G$  is a *1-sink parity game* iff there is a node  $v \in V$  such that  $\Omega(v) = 1$ ,  $(v, v) \in E$ ,  $\Omega(w) > 1$  for every other node  $w \in V$ ,  $v$  is the only cycle in  $G$  that is won by player 1, and player 1 has a winning strategy for the whole game.

**THEOREM 3.2.** ([FRI10]) *Let  $G$  be a 1-sink parity game. Discrete policy iteration requires the same number of iterations to solve  $G$  as the policy iteration for the induced payoff games as well as turn-based stochastic games to solve the respective game  $G'$  induced by applying the standard reduction from  $G$  to the respective game class, assuming that the improvement policy solely depends on the ordering of the improving edges.*

#### 4 The Random Facet algorithm

The RANDOMFACET algorithm of Matoušek, Sharir and Welzl [MSW96] is a very simple randomized algorithm for solving LP-type problems. As parity games, and the other type of games considered in Section 2, are LP-type problems (Halman [Hal07]), the algorithm can be used to solve these games, as was done by Ludwig [Lud95], Petersson and Vorobyov [PV01], and Björklund *et al.* [BSV03],[BV05],[BV07].

For concreteness, we describe the operation of the RANDOMFACET algorithm on parity games. Let  $G = (V_0, V_1, E, \Omega)$  be a parity game. Recall that  $E_0 = E \cap (V_0 \times V)$ . Let  $\sigma$  be an initial strategy for player 0. If  $\sigma = E_0$ , then  $\sigma$  is the only possible strategy for player 0, and is thus also an optimal strategy. Otherwise, the algorithm chooses, uniformly at random, an edge  $e \in E_0 \setminus \sigma$  and applies the algorithm recursively on the subgame  $G \setminus \{e\} = (V_0, V_1, E \setminus \{e\}, \Omega)$ , the game obtained by removing  $e$  from  $G$ , with the initial strategy  $\sigma$ . The recursive call returns a strategy  $\sigma'$  which is an optimal strategy for player 0 in  $G \setminus \{e\}$ . If  $e$  is not an improving switch for  $\sigma'$ , then  $\sigma'$  is also an optimal strategy in  $G$ . Otherwise, the algorithm performs the switch  $\sigma'[e]$ , and recursively calls the algorithm on  $G$ , with initial strategy  $\sigma'[e]$ . For pseudo-code, see Figure 1. It follows from the analysis of [MSW96] that the *expected number of switches* performed by the RANDOMFACET algorithm on any parity game is  $2^{O(\sqrt{n \log n})}$ , where  $n = |V_0|$  is the number of vertices controlled by player 0 in  $G$ .

We also consider a variant RANDOMFACET\* of the RANDOMFACET algorithm, see Figure 2. This variant receives, as a third argument, an *index function*  $\text{ind} : E_0 \rightarrow \mathbb{N}$  that assigns each edge of  $E_0$  a *distinct* natural number. Let  $\mathcal{I}(G)$  be the set of all index functions w.r.t.  $G$  with range  $\{1, 2, \dots, |E_0|\}$ . Instead of choosing a *random* edge  $e$  from  $E_0 \setminus \sigma$ , the algorithm now chooses the edge of  $E_0 \setminus \sigma$  with the *smallest index*. We show below that the expected running time of this

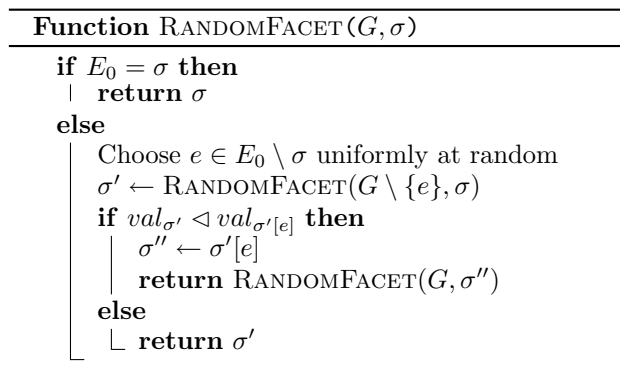


Figure 1: The RANDOMFACET algorithm

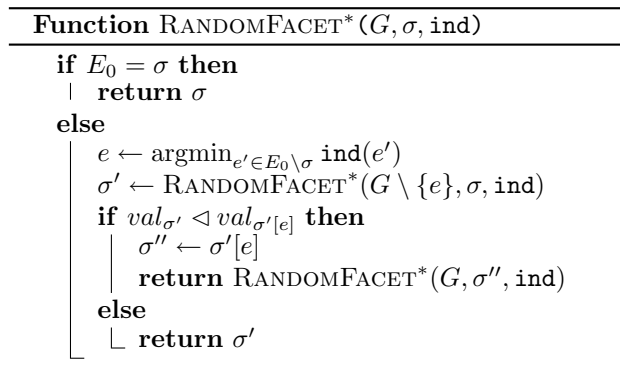


Figure 2: Variant of the RANDOMFACET algorithm

modified algorithm, when  $\text{ind}$  is taken to be a *random permutation* of  $E_0$ , is exactly equal to the expected running time of the original algorithm. We find it more convenient to work with the modified algorithm. The fact that the ordering of the edges is selected in advance simplifies the analysis. All our results apply, of course, also to the original version.

Let  $G = (V_0, V_1, E, \Omega)$  be a parity game and let  $F \subseteq E_0$ . Recall that  $G_F = (V_0, V_1, F \cup E_1, \Omega)$  is the subgame of  $G$  in which only the edges of  $F$  are available for player 0. Let  $\sigma \subseteq F$  be a strategy of player 0 in  $G_F$ . We let  $\mathbb{E}(G_F, \sigma)$  be the expected number of iterations performed by the call RANDOMFACET( $G_F, \sigma$ ). Also, we let  $\mathbb{E}^*(G_F, \sigma)$  be the expected number of iterations performed by the call RANDOMFACET\*( $G_F, \sigma, \text{ind}$ ) when  $\text{ind}$  is taken to be a random permutation of  $E_0$ .

We now have the following lemma by the linearity of the expectation; the random choices made by the two recursive calls made by RANDOMFACET are allowed to be dependent.

**LEMMA 4.1.** *Let  $G$  be a game,  $F \subseteq E_0$  and  $\sigma \subseteq F$  be a player 0 strategy. Then  $\mathbb{E}(G_F, \sigma) = \mathbb{E}^*(G_F, \sigma)$ .*

The simple proof of Lemma 4.1 can be found in Appendix A.

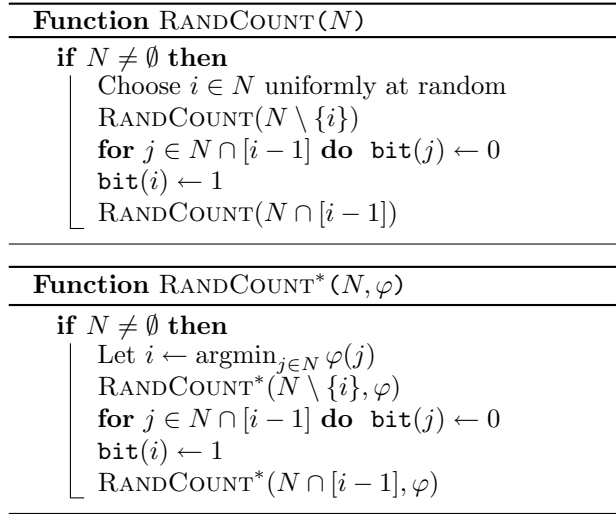


Figure 3: Counting with a randomized bit-counter (top), and counting with a modified, randomized bit-counter (bottom).

### 5 High-level description of the construction

The parity games that we construct on which the RANDOMFACET algorithm performs an expected subexponential number of iterations simulate a *randomized counter*. Such a counter is composed of  $n$  bits, all initially set to 0. We say that the  $i$ 'th bit is *set* if  $\mathbf{bit}(i) = 1$  and *reset* if  $\mathbf{bit}(i) = 0$ . The randomized counter works in a recursive manner, focusing each time on a subset  $N \subseteq [n] := \{1, \dots, n\}$  of the bits, such that for all  $j \in N$ ,  $\mathbf{bit}(j) = 0$ . Initially  $N = [n]$ . If  $N = \emptyset$ , then nothing is done. Otherwise, the counter chooses a random index  $i \in N$  and recursively performs a randomized count on  $N \setminus \{i\}$ . When this recursive call is done, we have  $\mathbf{bit}(j) = 1$ , for every  $j \in N \setminus \{i\}$ , while  $\mathbf{bit}(i) = 0$ . Next, all bits  $j \in N \cap [i - 1]$  are reset and the  $i$ 'th bit is set. Although it is more natural to increment the counter and then reset, resetting first corresponds to the behaviour for the lower bound construction. Finally, a recursive randomized count is performed on  $N \cap [i - 1]$ . A function RANDCOUNT that implements a randomized counter is given in Figure 3.

Let  $g(n)$  be the expected number of steps (recursive calls) performed by an  $n$ -bit randomized counter. It is easy to see that  $g(0) = 1$  and that

$$g(n) = 1 + g(n - 1) + \frac{1}{n} \sum_{i=0}^{n-1} g(i) \quad , \quad \text{for } n > 0.$$

The asymptotic behavior of  $g(n)$  is known quite precisely:

LEMMA 5.1. ([FS09], P. 596-597) *It holds that:*

$$g(n) \longrightarrow \frac{e^{2 \cdot \sqrt{n} - \frac{1}{2}}}{\sqrt{\pi} \cdot n^{\frac{1}{4}}} \quad \text{for } n \rightarrow \infty.$$

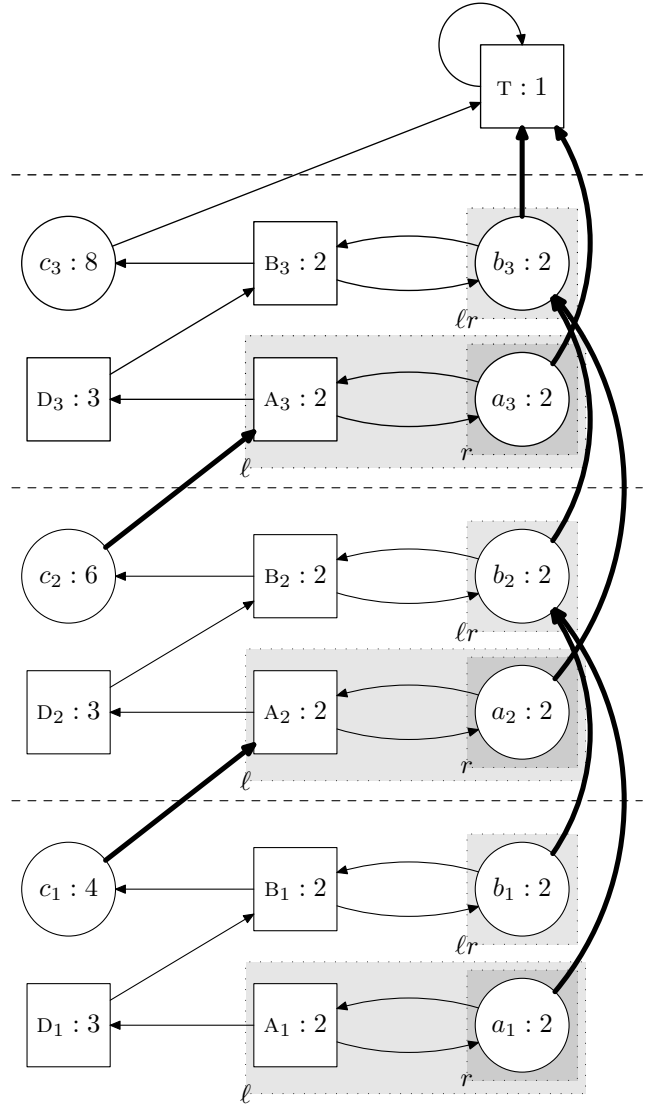


Figure 4: Lower bound construction for the RANDOMFACET algorithm.

Note that  $g(n)$  is, thus, just of the right subexponential form. The challenge, of course, is to construct parity games on which the behavior of the RANDOMFACET algorithm mimics the behavior of RANDCOUNT. In order to explain the idea for doing so, we examine a simplified version of our construction.

Consider the parity game shown in Figure 4, but ignore the shaded rectangles in the background and the fact that some arrows are bold. Round vertices are controlled by player 0 and square vertices by player 1. Vertices are labelled with an identifier and a priority.

Any strategy  $\sigma$  for player 0 encodes a counter state in the following way:  $\mathbf{bit}(i) = 1$  iff  $(b_i, B_i), (a_i, A_i) \in \sigma$ . Note that player 1 can always reach T by staying in the left side, and that T has odd priority. Thus, if  $\mathbf{bit}(i) = 1$  player 1, who plays a best reply to  $\sigma$ , will use the edge  $(B_i, c_i)$ , and  $c_i$  has a large even priority. Similarly, if

$\text{bit}(i) = 1$ , player 1 uses the edge  $(A_i, D_i)$ .

The importance of player 1's choice at  $A_i$  is that it determines whether lower set bits have access to  $c_i$ , which has the large even priority. If  $(b_i, B_i) \in \sigma$  and  $(a_i, A_i) \notin \sigma$ , we say that the  $i$ 'th bit is *resetting*, since it allows the RANDOMFACET algorithm to remove  $(a_i, A_i)$ , so that player 1 can block the access to  $c_i$  for lower set bits, which initiates the resetting behaviour of the counter.

Now, suppose the RANDOMFACET algorithm removes the edge  $(b_i, B_i)$  and solves the game recursively (by counting with the remaining bits). During the recursion we say that the  $i$ 'th bit is *disabled*. The resulting strategy  $\sigma'$  by player 0 will correspond to the state of the counter where  $\text{bit}(j) = 1$ , for  $j \neq i$ , and  $(b_i, B_i), (a_i, A_i) \notin \sigma$ . Using  $(b_i, B_i)$  will be an improving switch for player 0, so we get a new strategy  $\sigma'' = \sigma'[(b_i, B_i)]$  for which the  $i$ 'th bit is resetting.

Next, suppose that  $(a_i, A_i)$  is removed and the game solved recursively. Note that this is only possible because the  $i$ 'th bit is resetting and  $(a_i, A_i) \notin \sigma$ . In particular, a corresponding edge could not be picked for a set bit. Since player 1 controls the left half of the graph, and player 0 is unable to use the edge  $(a_i, A_i)$ , player 1 can avoid the large even priority at  $c_i$  by staying to the left until moving from  $A_i$  to  $a_i$ . Thus, player 0 can only reach  $c_i$  from vertices  $a_j, b_j$ , with  $j < i$ , by staying within the right half of the graph. It follows that all counter bits with index  $j < i$  are reset. Using the edge  $(a_i, A_i)$  will now be an improving switch for player 0, so the strategy is updated such that  $\text{bit}(i) = 1$ , and we are ready for a second round of counting with the lower bits. During this recursive call the higher set bits are said to be *inactive*.

Note that in order to ensure the correct behavior it was crucial that  $(b_i, B_i)$  was picked by the RANDOMFACET algorithm before  $(a_i, A_i)$ . It was also crucial that other edges from  $b_i$  and  $a_i$ , i.e.  $(b_i, b_{i+1})$  and  $(a_i, b_{i+1})$ , were not removed. To increase the probability of that happening we make use of *duplication*. A shaded rectangles with  $\ell$  or  $r$  in the bottom-left corner indicates that the corresponding subgraph is copied  $\ell$  or  $r$  times, respectively. Also, a bold edge indicates that the corresponding edge is copied  $r$  times. We show that it suffices for  $\ell$  and  $r$  to be logarithmic in  $n$  to get a good bound on the probability.

As was the case with RANDOMFACET, we can obtain a modified version of RANDCOUNT in which all random decisions are made in advance. In the case of RANDCOUNT, this corresponds to choosing a random permutation on  $[n]$ . Let  $\mathcal{S}(n)$  be the set of permutations on  $[n]$ . A function RANDCOUNT\* that implements a modified, randomized counter is given in Figure 3.  $\varphi \in \mathcal{S}(n)$  is a permutation.

We end this section with a lemma showing that the

expected number of steps performed by the original counter, RANDCOUNT, is equal to the expected number of steps performed by the modified counter, RANDCOUNT\*, when given a random permutation. More precisely, let  $f_n(N, \varphi)$  be the number of steps performed by a call RANDCOUNT\*( $N, \varphi$ ), for some permutation  $\varphi \in \mathcal{S}(n)$ . Let  $f_n(N)$  be the expected value of  $f_n(N, \varphi)$  when  $\varphi \in \mathcal{S}(n)$  is picked uniformly at random.

LEMMA 5.2. *Let  $n \in \mathbb{N}$ . Then  $f_n([n]) = g(n)$ .*

The simple proof of Lemma 5.2, which is similar to the proof of Lemma 4.1, can be found in Appendix B.

## 6 Complete description of lower bound games

In this section we define a family of *lower bound games*  $G_{n,\ell,r} = (V_0, V_1, E, \Omega)$  that the RANDOMFACET algorithm requires many iterations to solve.  $n$  denotes the number of bits in the simulated randomized bit-counter, and  $\ell \geq 1$  and  $r \geq 1$  are parameters for the analysis in Section 8. We use multi edges for convenience. A similar graph without multi edges can easily be defined by introducing additional vertices.  $G_{n,\ell,r}$  is defined as follows.

$$\begin{aligned} V_0 &:= \{a_{i,j,k}\}_{[n] \times [\ell] \times [r]} \cup \{b_{i,j}\}_{[n] \times [\ell r]} \cup \{c_i\}_{[n]} \\ V_1 &:= \{A_{i,j}\}_{[n] \times [\ell]} \cup \{B_i\}_{[n]} \cup \{D_i\}_{[n]} \cup \{T\} \end{aligned}$$

where  $\{a_{i,j,k}\}_{[n] \times [\ell] \times [r]}$  is a shorthand for  $\{a_{i,j,k} \mid i \in [n], j \in [\ell], k \in [r]\}$ , etc. Figure 5 defines the edge set  $E$  and the priority assignment function  $\Omega$ , where  $b_{i,*}$  is a shorthand for the set  $\{b_{i,j} \mid j \in [\ell r]\}$ , and  $(T)_r$  indicates that the edge has multiplicity  $r$ .

Node $V$	Successors in $E$	Priority $\Omega$
$a_{i,j,k}$	$A_{i,j}, (b_{i+1,1})_r$	2
$a_{n,j,k}$	$A_{n,j}, (T)_r$	2
$b_{i,j}$	$B_i, (b_{i+1,1})_r$	2
$b_{n,j}$	$B_n, (T)_r$	2
$c_i$	$(A_{i+1,*})_r$	$2i + 2$
$c_n$	$T$	$2n + 2$
$A_{i,j}$	$D_i, a_{i,j,*}$	2
$B_i$	$c_i, b_{i,*}$	2
$D_i$	$B_i$	3
$T$	$T$	1

Figure 5: Edges and Priorities of  $G_{n,\ell,r}$

An example of a lower bound game with three bits, i.e.,  $n = 3$ , is shown in Figure 4. Round vertices are controlled by player 0 and square vertices by player 1. A shaded rectangle with label  $\ell$  indicates that the corresponding subgraph has been copied  $\ell$  times. Bold arrows are multi edges with multiplicity  $r$ . Note that bold incoming edges for  $b_i$ , in fact, only go to the vertex  $b_{i,1}$ .

The initial strategy  $\sigma$  given as input to the RANDOMFACET algorithm is described by:

$$\begin{aligned} \forall i \in [n] \forall j \in [\ell r] : \sigma(b_{i,j}) \neq B_i \\ \forall i \in [n] \forall j \in [\ell] \forall k \in [r] : \sigma(a_{i,j,k}) \neq A_{i,j} \end{aligned}$$

The choice at vertex  $c_i$ , for  $1 \leq i \leq n$ , is arbitrary.

## 7 Optimal strategies of subgames

The RANDOMFACET algorithm operates with a subset of the edges controlled by player 0,  $F \subseteq E_0$ , such that the corresponding subgame  $G_F$  is a parity game. We next introduce notation to concisely describe  $F$ . We say that  $F$  is *complete* if it contains at least one instance of every multi edge. We define the set of multi edges without multiplicities as:

$$\begin{aligned} M = & \{(a_{i,j,k}, b_{i+1,1}) \mid i \in [n-1], j \in [\ell], k \in [r]\} \cup \\ & \{(a_{n,j,k}, \top) \mid j \in [\ell], k \in [r]\} \cup \\ & \{(b_{i,j}, b_{i+1,1}) \mid i \in [n-1], j \in [\ell r]\} \cup \\ & \{(b_{n,j}, \top) \mid j \in [\ell r]\} \cup \\ & \{(c_i, A_{i+1,j}) \mid i \in [n-1], j \in [\ell]\}. \end{aligned}$$

Furthermore, for  $F \subseteq E_0$  define:

$$\begin{aligned} b_i(F) &= \begin{cases} 1 & \text{if } \forall j \in [\ell r] : (b_{i,j}, B_i) \in F \\ 0 & \text{otherwise} \end{cases} \\ a_{i,j}(F) &= \begin{cases} 1 & \text{if } \forall k \in [r] : (a_{i,j,k}, A_{i,j}) \in F \\ 0 & \text{otherwise} \end{cases} \\ a_i(F) &= \begin{cases} 1 & \text{if } \exists j \in [\ell] : a_{i,j}(F) = 1 \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

That is,  $b_i(F) = 1$  if and only if  $F$  contains every edge leading to  $B_i$ , and  $a_{i,j}(F) = 1$  if and only if  $F$  contains every edge leading to  $A_{i,j}$ .

Finally, we define:

$$\text{reset}(F) = \max(\{0\} \cup \{i \in [n] \mid b_i(F) = 1 \wedge a_i(F) = 0\})$$

to be the maximum index  $i$  for which  $b_i(F) = 1$  and  $a_i(F) = 0$ . Intuitively, all bits with index lower than  $i$  will be reset when computing the optimal strategy in the subgame  $G_F$ .

Let  $F \subseteq E_0$  be a complete set. We say that a strategy  $\sigma \subseteq F$  is *well-behaved* if for every  $i \in [n]$ , all copies  $a_{i,j,k}$ , for  $j \in [\ell]$  and  $k \in [r]$ , and all copies  $b_{i,j}$ , for  $j \in [\ell r]$ , adopt corresponding choices, whenever possible. More formally, for every  $i, j_1, j_2, k_1, k_2$ , if  $(a_{i,j_1,k_1}, A_{i,j_1}), (a_{i,j_2,k_2}, A_{i,j_2}) \in F$ , then  $\sigma(a_{i,j_1,k_1}) = A_{i,j_1}$  if and only if  $\sigma(a_{i,j_2,k_2}) = A_{i,j_2}$ , and similarly, for every  $i, j_1, j_2$ , if  $(b_{i,j_1}, B_i), (b_{i,j_2}, B_i) \in F$ , then  $\sigma(b_{i,j_1}) = B_i$  if and only if  $\sigma(b_{i,j_2}) = B_i$ . We show below that for every complete set  $F \subseteq E_0$ , the optimal strategy of player 0 in  $G_F$  is well-behaved.

The essential behavior of a well-behaved policy  $\sigma$  is characterized by two Boolean vectors  $\alpha(\sigma) = (\alpha_1, \dots, \alpha_n)$  and  $\beta(\sigma) = (\beta_1, \dots, \beta_n)$  that are defined as follows:

$$\begin{aligned} \alpha_i(\sigma) &= \begin{cases} 1 & \text{if } \forall j \in [\ell] \forall k \in [r] : \\ & (a_{i,j,k}, A_{i,j}) \in F \Rightarrow \sigma(a_{i,j,k}) = A_{i,j} \\ 0 & \text{if } \forall j \in [\ell] \forall k \in [r] : \sigma(a_{i,j,k}) \neq A_{i,j} \end{cases} \\ \beta_i(\sigma) &= \begin{cases} 1 & \text{if } \forall j \in [\ell r] : \\ & (b_{i,j}, B_i) \in F \Rightarrow \sigma(b_{i,j}) = B_i \\ 0 & \text{if } \forall j \in [\ell r] : \sigma(b_{i,j}) \neq B_i \end{cases} \end{aligned}$$

Similarly, given two Boolean vectors  $\alpha = (\alpha_1, \dots, \alpha_n)$  and  $\beta = (\beta_1, \dots, \beta_n)$  we let  $\sigma = \sigma(\alpha, \beta)$  be a well-behaved strategy such that  $\alpha(\sigma) = \alpha$  and  $\beta(\sigma) = \beta$ . Note that  $\sigma(\alpha, \beta)$  is not uniquely determined, as when  $\alpha_i = 0$  or  $\beta_i = 0$  we do not specify which copy of a multi-edge is chosen. This choice, however, is irrelevant.

The  $i$ 'th bit of the randomized bit-counter is interpreted as being *set*, for some complete set  $F$  and a well-behaved strategy  $\sigma$ , if  $a_i(F) = b_i(F) = 1$  and  $\alpha_i(\sigma) = \beta_i(\sigma) = 1$ .

Let  $\sigma_F^*$  and  $\tau_F^*$  be optimal strategies for player 0 and 1, respectively, in the subgame  $G_F = (V_0, V_1, F \cup E_1, \Omega)$  defined by edges of  $F$ . We show below that  $\sigma_F^*$  is always well-behaved, and we let  $\alpha^*(F) = \alpha(\sigma_F^*)$  and  $\beta^*(F) = \beta(\tau_F^*)$ . The following lemma then describes the key parts of optimal strategies in the construction.

**LEMMA 7.1.** *Let  $F \subseteq E_0$  be complete. Then  $\sigma_F^*$  is well-behaved and  $\beta_i^*(F) = 1$  if and only if  $i \geq \text{reset}(F)$ , and  $\alpha_i^*(F) = 1$  if and only if  $b_i(F) = 1$  and  $i \geq \text{reset}(F)$ .*

The proof of Lemma 7.1 relies on a technical lemma stated below. Note that, when  $F$  is complete, both players can force the play to reach  $\top$  by simply not moving into the cycles of length two. In particular,  $G_F$  is a 1-sink game. For all  $v \in V$ , let  $\text{val}_{\sigma_F^*, \tau_F^*}(v) = (p_F(v), A_F(v), \ell_F(v))$ . In our case,  $p_F(v) = 1$ , for every  $v \in V$ , and  $A_F(v)$  is the multi-set of priorities greater than  $p_F(v)$  encountered on the way to  $\top$ . Also note that except for the small cycles of length at most two,  $G_F$  is acyclic. We prove Lemma 7.1 by constructing  $\sigma_F^*$  and  $\tau_F^*$  by backwards induction from  $\top$ . The following lemma allows us to handle cycles while doing so.

Let

$$A_F^k(v) = \{x \in A_F(v) \mid x \geq k\}.$$

The total ordering  $\prec$  is defined for sets of priorities as it was for valuations.

LEMMA 7.2. Let  $F \subseteq E_0$  be complete, then:

$$\beta_i^*(F) = \begin{cases} 1 & \text{if } A_F^3(c_i) \succ A_F^3(b_{i+1,1}) \\ 0 & \text{if } A_F^3(c_i) \prec A_F^3(b_{i+1,1}) \end{cases}$$

$$A_F^3(B_i) = \begin{cases} A_F^3(c_i) & \text{if } \beta_i^*(F) = 0 \text{ or } b_i(F) = 1 \\ A_F^3(b_{i+1,1}) & \text{otherwise} \end{cases}$$

$$A_F^3(b_{i,j}) = \begin{cases} A_F^3(c_i) & \text{if } \beta_i^*(F) = 1 \text{ and } b_i(F) = 1 \\ A_F^3(b_{i+1,1}) & \text{otherwise} \end{cases}$$

and similarly:

$$\alpha_i^*(F) = \begin{cases} 1 & \text{if } A_F^3(D_i) \succ A_F^3(b_{i+1,1}) \\ 0 & \text{if } A_F^3(D_i) \prec A_F^3(b_{i+1,1}) \end{cases}$$

$$A_F^3(A_{i,j}) = \begin{cases} A_F^3(D_i) & \text{if } \alpha_i^*(F) = 0 \text{ or } a_{i,j}(F) = 1 \\ A_F^3(b_{i+1,1}) & \text{otherwise} \end{cases}$$

$$A_F^3(a_{i,j,k}) = \begin{cases} A_F^3(D_i) & \text{if } \alpha_i^*(F) = 1 \text{ and } a_{i,j}(F) = 1 \\ A_F^3(b_{i+1,1}) & \text{otherwise} \end{cases}$$

The technical proofs of Lemma 7.2 and 7.1, can be found in appendices C and D, respectively.

## 8 Lower bound proof

In this section we consider the expected number of iterations performed by the modified RANDOMFACET algorithm when applied to our family of lower bound games. We show that for appropriate parameters  $\ell$  and  $r$  the number of iterations is, with high probability, at least as large as the expected number of steps performed by the corresponding modified randomized bit-counter.

For brevity, we say that the RANDOMFACET algorithm takes a set  $F \subseteq E_0$  as argument rather than the game  $G_F$ . Let  $F \subseteq E_0$ ,  $\sigma$  and  $\text{ind}$  be arguments to the RANDOMFACET algorithm, and let  $e = \text{argmin}_{e' \in F \setminus \sigma} \text{ind}(e')$ . We distinguish between three types of iterations; *count-iterations*, *reset-iterations* and *irrelevant iterations*. We say that an iteration is a count-iteration if it satisfies the following:

$$\text{reset}(F) = 0 \quad \text{and} \\ \exists i \in [n] : b_i(F) = 1 \wedge b_i(F \setminus \{e\}) = 0$$

Similarly, we say that an iteration is a reset-iteration if it satisfies:

$$\text{reset}(F) = 0 \quad \text{and} \quad \text{reset}(F \setminus \{e\}) > 0$$

An iteration that is neither a count-iteration nor a reset-iteration is said to be irrelevant.

In order to correctly simulate a modified randomized bit-counter it must be the case that between any two count-iterations there is a reset-iteration. Furthermore,

$F$  must be complete in every count-iteration, as well as in every reset-iteration. To handle these requirements we introduce the notion of a *good* index function. Recall that  $M$  is the set of multi edges without multiplicities. We write  $e_t$  to refer to the  $t$ 'th copy of some edge  $e \in M$ . We say that an index function  $\text{ind}$  is good if it satisfies the following two requirements:

1.  $\forall i \in [n] \exists j \in [\ell] \exists t \in [\ell r] \forall k \in [r] :$ 

$$\text{ind}(b_{i,t}, B_i) < \text{ind}(a_{i,j,k}, A_{i,j})$$
2.  $\forall e \in M \forall i \in [n] \forall j \in [\ell] \exists k \in [r] \exists t \in [r] :$ 

$$\text{ind}(a_{i,j,k}, A_{i,j}) < \text{ind}(e_t)$$

The first requirement says that for every  $i$ , the first edge, according to  $\text{ind}$ , going to  $B_i$  is before the first edge going to  $A_{i,j}$ , for some  $j$ . The second requirement says that for all  $i$  and  $j$ , the first edge going to  $A_{i,j}$  is before some copy of every edge from  $M$ . Note that when both requirements are combined we also get that for all  $i$ , the first edge going to  $B_i$  is before some copy of every edge from  $M$ .

For  $F \subseteq E_0$  and a well-behaved strategy  $\sigma \subseteq F$ , define for  $1 \leq i \leq n$  where  $b_i(F) = 1$  and  $a_i(F) = 1$ :

$$\text{bit}_{F,\sigma}(i) = \begin{cases} 1 & \text{if } \alpha_i(\sigma) = \beta_i(\sigma) = 1, \\ 0 & \text{if } \alpha_i(\sigma) = \beta_i(\sigma) = 0, \\ \perp & \text{otherwise.} \end{cases}$$

The state  $\text{bit}_{F,\sigma}(i) = \perp$  is an intermediate state in which the bit is switching from set to unset, or vice versa.

We say that the  $i$ 'th bit is *disabled* if  $b_i(F) = 0$ . Furthermore, we say that the  $i$ 'th bit is *inactive* if  $\text{bit}_{F,\sigma}(i) = 1$ , and for all  $i < j \leq n$ , the  $j$ 'th bit is either disabled or inactive. We also say that the  $i$ 'th bit is *resetting* if  $b_i(F) = 1$ ,  $a_i(F) = 1$ ,  $\beta_i(\sigma) = 1$  but  $\alpha_i(\sigma) = 0$ . We define the set of active bits  $N_{F,\sigma} \subseteq [n]$  such that  $i \in N_{F,\sigma}$  if and only if the  $i$ 'th bit is not disabled, inactive or resetting. Finally, we define the permutation  $\phi_{\text{ind}}$  of  $[n]$  such that for all  $i, j \in [n]$ :

$$\phi_{\text{ind}}(i) < \phi_{\text{ind}}(j) \iff \\ \exists k \in [\ell r] \forall t \in [\ell r] : \text{ind}(b_{i,k}, B_i) < \text{ind}(b_{j,t}, B_j)$$

Note that these concepts correspond exactly to the concepts utilized in a randomized bit-counter.

Let  $f_{\text{ind}}(F, \sigma)$  be the number of iterations (recursive calls) performed by the call  $\text{RANDOMFACET}^*(G_F, \sigma, \text{ind})$ . We denote the expected value of  $f_{\text{ind}}(F, \sigma)$ , when  $\text{ind}$  is a random good index function picked from the set of permutations of  $E_0$ , by  $\mathbb{E}_{G_{n,\ell,r}}^*(F, \sigma | \text{ind is good})$ .

LEMMA 8.1. *Let  $G_{n,\ell,r}$  be a lower bound game with initial strategy  $\sigma$  for player 0, then*

$$\mathbb{E}_{G_{n,\ell,r}}^*(E_0, \sigma | \mathbf{ind} \text{ is good}) \geq g(n).$$

*Proof.* Using forward induction, we first show that  $N_{F,\sigma}$  and  $\mathbf{bit}_{F,\sigma}(i)$ , for  $1 \leq i \leq n$ , are updated in the same way as in a modified randomized bit-counter with  $n$  bits, and we provide bounds for  $f_{\mathbf{ind}}(F, \sigma)$ . We later use backward induction to combine these observations and complete the proof.

*Forward induction:* For the first step we use the following induction hypothesis:

- If the call  $\text{RANDOMFACET}^*(G_F, \sigma, \mathbf{ind})$  performs either a count-iteration or a reset-iteration then  $F$  is complete.
- If there is no resetting bit then, for all  $i \in N_{F,\sigma}$ ,  $\mathbf{bit}_{F,\sigma}(i) = 0$ .
- If the  $i$ 'th bit is resetting then it is the only resetting bit, and for all  $i \in N_{F,\sigma}$ ,  $\mathbf{bit}_{F,\sigma}(i) = 1$ .

This is clearly true for the first iteration, since  $E_0$  is complete and for the initial input we have  $N_{E_0,\sigma} = [n]$ , and for all  $1 \leq i \leq n$ ,  $\mathbf{bit}_{E_0,\sigma}(i) = 0$ .

Let  $F$  and  $\sigma$  be given, and let  $e = \text{argmin}_{e' \in F \setminus \sigma} \mathbf{ind}(e')$ . Any iteration is either a count-iteration, a reset-iteration or irrelevant. We consider the three cases separately.

*Case 1:* Assume that the iteration is a count-iteration. Then  $e = (b_{i,j}, B_i)$  for some  $i$  and  $j$ , and in particular  $i \in N_{F,\sigma}$ , since  $e \notin \sigma$ . By the same argument  $\mathbf{bit}_{F,\sigma}(i) = 0$ , and, hence, there can be no resetting bit.

Since  $i \in N_{F,\sigma}$ , we have  $i = \text{argmin}_{j \in N_{F,\sigma}} \phi_{\mathbf{ind}}(j)$ . During the first recursive call the  $i$ 'th bit is disabled, and we get:

$$N_{F \setminus \{e\}, \sigma} = N_{F,\sigma} \setminus \{i\}.$$

Let  $\sigma' = \sigma_{F \setminus \{e\}}^*$ . Since  $F \setminus \{e\}$  is still complete, we can apply Lemma 7.1 to  $F \setminus \{e\}$  as well as  $F$ , and it follows that  $\sigma'$  is not optimal for  $F$ . Since  $\text{reset}(F \setminus \{e\}) = 0$  and  $b_i(F \setminus \{e\}) = 0$  we get for  $\sigma'' = \sigma'[e]$ :

$$\begin{aligned} \forall j \in N_{F,\sigma} : \sigma''(b_j) &= 1 \\ \forall j \in N_{F,\sigma} \setminus \{i\} : \sigma''(a_j) &= 1 \\ \sigma''(a_i) &= 0 \end{aligned}$$

Hence, for the second recursive call the  $i$ 'th bit is resetting, and  $N_{F,\sigma''} = N_{F,\sigma} \cap [i-1]$ , and the induction step is complete.

Also, note that:

$$f_{\mathbf{ind}}(F, \sigma) = 1 + f_{\mathbf{ind}}(F \setminus \{e\}, \sigma) + f_{\mathbf{ind}}(F, \sigma'').$$

*Case 2:* Assume that the iteration is a reset-iteration. Then  $e = (a_{i,j,k}, A_{i,j})$  for some  $i, j$  and  $k$ , and the

$i$ 'th bit can neither be disabled nor inactive. If there is a resetting bit then  $i \notin N_{F,\sigma}$ , since  $e \notin \sigma$  and  $\mathbf{bit}_{F,\sigma}(i') = 1$  for all  $i' \in N_{F,\sigma}$ . Hence, if there is a resetting bit then  $i$  must be resetting. On the other hand, if there is no resetting bit, then  $i \notin N_{F,\sigma}$  since the index function  $\mathbf{ind}$  is good, and the first requirement for a good index function would imply that we instead have  $e = (b_{i,j'}, B_i)$ , for some  $j'$ . Thus, the  $i$ 'th bit must be resetting, and we have  $\text{reset}(F \setminus \{e\}) = i$ .

Let  $\sigma' = \sigma_{F \setminus \{e\}}^*$ .  $F \setminus \{e\}$  remains complete, and we apply Lemma 7.1 to  $F \setminus \{e\}$  as well as  $F$ , and see that  $\sigma'$  is not optimal for  $F$ . For  $\sigma'' = \sigma'[e]$  we then get:

$$\begin{aligned} \forall j \in N_{F,\sigma} : \mathbf{bit}_{F,\sigma''}(j) &= 0 \\ \mathbf{bit}_{F,\sigma''}(i) &= 1 \end{aligned}$$

Thus,  $N_{F,\sigma''} = N_{F,\sigma}$  for the second recursive call, there is no resetting bit, and the induction step follows.

By ignoring contributions to the number of iterations from the first recursive call, as well as from the reset-iteration itself, we get  $f_{\mathbf{ind}}(F, \sigma) \geq f_{\mathbf{ind}}(F, \sigma'')$ . Also note that we do not need to argue that the induction hypothesis holds during the first recursive call, since in the end we are only interested in the resulting optimal strategy.

*Case 3:* Assume that the iteration is irrelevant. Then  $N_{F \setminus \{e\}, \sigma} = N_{F,\sigma}$ ,  $\mathbf{bit}_{F \setminus \{e\}, \sigma}(i) = \mathbf{bit}_{F,\sigma}(i)$ , for all  $i$ , and  $\text{reset}(F) = \text{reset}(F \setminus \{e\})$ . Note also that either  $F \setminus \{e\}$  is complete, or there are no following count-iterations or reset-iterations. This follows from the assumption that  $\mathbf{ind}$  is a good index function, and the second requirement for a good index function. If  $F \setminus \{e\}$  is not complete, then  $N_{F,\sigma} = \emptyset$  since the choice of  $e$  would otherwise be different. For the first recursive call  $\sigma$  remains the same, and the induction step follows.

We ignore contributions to the number of iterations from the second recursive call, and since we know the optimal strategy  $\sigma_F^*$  from Lemma 7.1, we do not need to argue that the invariants are satisfied during the second recursive call. Thus,  $f_{\mathbf{ind}}(F, \sigma) \geq f_{\mathbf{ind}}(F \setminus \{e\}, \sigma)$ .

*Backward induction:* Let  $N \subseteq [n]$ , and let  $\phi$  be a permutation of  $[n]$ . Recall that the function  $f_n$  is defined as  $f_n(\emptyset, \phi) = 1$ , and for  $N \neq \emptyset$ :

$$f_n(N, \phi) = f_n(N \setminus \{i\}, \phi) + f_n(N \cap [i-1], \phi)$$

where  $i = \text{argmin}_{j \in N} \phi(j)$ . Furthermore,  $f_n(N)$  is the expected value of  $f_n(N, \phi)$  when  $\phi$  is picked uniformly at random, and from Lemma 5.2 we have  $f_n([n]) = g(n)$ .

Let  $F \subseteq E_0$ ,  $\sigma$  and  $\mathbf{ind}$  be arguments for the  $\text{RANDOMFACET}$  algorithm for some iteration, and let  $e = \text{argmin}_{e' \in F \setminus \sigma} \mathbf{ind}(e')$  and  $i = \text{argmin}_{j \in N_{F,\sigma}} \phi_{\mathbf{ind}}(j)$ . We next show that:

$$f_{\mathbf{ind}}(F, \sigma) \geq f_n(N_{F,\sigma}, \phi_{\mathbf{ind}})$$

if the iteration does not appear during the first recursive call of a reset-iteration or the second recursive call of an irrelevant iteration, i.e., if the previous induction hypothesis and case analysis is valid. The inequality is proved by backward induction.

For the basis consider the case where  $F = \sigma$ . Since there is no resetting edge we have  $N_{F,\sigma} = \emptyset$ , and we get  $f_{\text{ind}}(F, \sigma) = f_n(N_{F,\sigma}, \phi_{\text{ind}}) = 1$ . For the induction step we observe that:

- If the iteration is a count-iteration, then:

$$\begin{aligned} f_{\text{ind}}(F, \sigma) &= \\ 1 + f_{\text{ind}}(F \setminus \{e\}, \sigma) + f_{\text{ind}}(F, \sigma'') &\geq \\ 1 + f_n(N_{F \setminus \{e\}, \sigma}, \phi_{\text{ind}}) + f_n(N_{F, \sigma''}, \phi_{\text{ind}}) &= \\ 1 + f_n(N_{F, \sigma} \setminus \{i\}, \phi_{\text{ind}}) + f_n(N_{F, \sigma} \cap [i-1], \phi_{\text{ind}}) &= \\ f_n(N_{F, \sigma}, \phi_{\text{ind}}). \end{aligned}$$

- If the iteration is a reset-iteration, then:

$$\begin{aligned} f_{\text{ind}}(F, \sigma) &\geq f_{\text{ind}}(F, \sigma'') \geq f_n(N_{F, \sigma''}, \phi_{\text{ind}}) = \\ &f_n(N_{F, \sigma}, \phi_{\text{ind}}). \end{aligned}$$

- If the iteration is irrelevant, then:

$$\begin{aligned} f_{\text{ind}}(F, \sigma) &\geq f_{\text{ind}}(F \setminus \{e\}, \sigma) \geq \\ f_n(N_{F \setminus \{e\}, \sigma}, \phi_{\text{ind}}) &= f_n(N_{F, \sigma}, \phi_{\text{ind}}). \end{aligned}$$

*Conclusion:* Note that  $\phi_{\text{ind}}$  is a random permutation of  $[n]$  when  $\text{ind}$  is picked uniformly at random from the set of good index functions that are permutations of  $E_0$ . Thus, since  $f_{\text{ind}}(F, \sigma) \geq f_n(N_{F, \sigma}, \phi_{\text{ind}})$  we get:

$$\mathbb{E}_{G_{n,\ell,r}}^*(F, \sigma | \text{ind is good}) \geq f_n(N_{F, \sigma})$$

and in particular, for the initial input:

$$\begin{aligned} \mathbb{E}_{G_{n,\ell,r}}^*(E_0, \sigma | \text{ind is good}) &\geq \\ f_n(N_{E_0, \sigma}) &= f_n([n]) = g(n), \end{aligned}$$

which concludes the proof.  $\square$

LEMMA 8.2. *Let  $G_{n,\ell,r}$  be a lower bound game, and let  $\text{ind}$  be chosen uniformly at random from the set of permutations of  $E_0$ . Then  $\text{ind}$  is good with probability  $p_{n,\ell,r}$ , where:*

$$p_{n,\ell,r} \geq 1 - n \frac{(\ell!)^2}{(2\ell)!} - n\ell(n(2\ell r + \ell) - \ell) \frac{(r!)^2}{(2r)!}$$

*Proof.* The main idea of the proof is to use the following observation. Let  $S = \{b_1, \dots, b_\ell, a_1, \dots, a_\ell\}$  and  $\phi$  be a uniformly random permutation of  $S$ . Then:

$$Pr[\forall i \in [\ell] \forall j \in [\ell] : \phi(b_i) > \phi(a_j)] = \frac{(\ell!)^2}{(2\ell)!}$$

Recall that the first requirement for a good index function  $\text{ind}$  was:

1.  $\forall i \in [n] \exists j \in [\ell] \exists t \in [\ell r] \forall k \in [r] :$

$$\text{ind}(b_{i,t}, B_i) < \text{ind}(a_{i,j,k}, A_{i,j})$$

We first consider the probability that a random index function  $\text{ind}$  does not satisfy the first requirement for being good.

Let  $1 \leq i \leq n$  be fixed. We identify each element of  $S^i = \{b_1^i, \dots, b_\ell^i, a_1^i, \dots, a_\ell^i\}$  with a set of  $r$  edges, such that for all  $j \in [\ell]$ :

$$\begin{aligned} b_j^i &:= \{(b_{i,r(j-1)+k}, B_i) \mid k \in [r]\} \\ a_j^i &:= \{(a_{i,j,k}, A_{i,j}) \mid k \in [r]\} \end{aligned}$$

Furthermore, we define the permutation  $\phi_{\text{ind}}^i$  of  $S^i$  such that for two distinct elements  $x, y \in S^i$ :

$$\begin{aligned} \phi_{\text{ind}}^i(x) < \phi_{\text{ind}}^i(y) &\iff \\ \exists e \in x \forall e' \in y : \text{ind}(e) &< \text{ind}(e') \end{aligned}$$

Note that when  $\text{ind}$  is a random permutation of  $E_0$ , then  $\phi_{\text{ind}}^i$  is a random permutation of  $S^i$ . Now  $\text{ind}$  satisfies the first requirement for being good if and only if:

$$\forall i \in [n] \exists j \in [\ell] \exists t \in [\ell] : \phi_{\text{ind}}^i(b_t^i) < \phi_{\text{ind}}^i(a_j^i)$$

It follows that the probability that the first requirement is not satisfied is at most  $n \frac{(\ell!)^2}{(2\ell)!}$ .

Next, we consider the probability of  $\text{ind}$  not satisfying the second requirement. Recall that the second requirement was:

2.  $\forall e \in M \forall i \in [n] \forall j \in [\ell] \exists k \in [r] \exists t \in [r] :$

$$\text{ind}(a_{i,j,k}, A_{i,j}) < \text{ind}(e_t)$$

In fact, this case is simpler than the first. Define:

$$S^{e,i,j} = \{e_1, \dots, e_r, (a_{i,j,1}, A_{i,j}), \dots, (a_{i,j,r}, A_{i,j})\}$$

and define  $\phi_{\text{ind}}^{e,i,j}$  such that for two distinct elements  $x, y \in S^{e,i,j}$ :

$$\phi_{\text{ind}}^{e,i,j}(x) < \phi_{\text{ind}}^{e,i,j}(y) \iff \text{ind}(x) < \text{ind}(y)$$

Again we see that when  $\text{ind}$  is a random permutation of  $E_0$ , then  $\phi_{\text{ind}}^{e,i,j}$  is a random permutation of  $S^{e,i,j}$ . Since  $|M| = n(2\ell r + \ell) - \ell$ , it follows that the probability that the second requirement is not satisfied is at most  $n\ell(n(2\ell r + \ell) - \ell) \frac{(r!)^2}{(2r)!}$ , and the statement of the lemma follows.  $\square$

THEOREM 8.1. *The worst-case expected running time of the RANDOMFACET algorithm for  $n$ -state parity games is at least  $2^{\Omega(\sqrt{n}/\log n)}$ .*

*Proof.* Let  $G_{n,\ell,r}$  be a lower bound game, and let  $\sigma$  be the initial strategy. Note that unlike the statement of the lemma  $n$  refers to the number of bits in the corresponding randomized bit-counter. From Lemma 4.1, Lemma 8.1 and Lemma 5.1 we have:

$$\begin{aligned}\mathbb{E}_{G_{n,\ell,r}}(E_0, \sigma) &= \mathbb{E}_{G_{n,\ell,r}}^*(E_0, \sigma) \\ &\geq p_{n,\ell,r} \cdot \mathbb{E}_{G_{n,\ell,r}}^*(E_0, \sigma | \text{ind is good}) \\ &\geq p_{n,\ell,r} \cdot g(n) \\ &= p_{n,\ell,r} \cdot 2^{\Omega(\sqrt{n})}\end{aligned}$$

All that remains is, thus, to pick the parameters  $\ell$  and  $r$  such that  $p_{n,\ell,r}$  is constant. In the following we show that for  $\ell = r = 3 \log n$  and  $n$  sufficiently large, we get  $p_{n,\ell,r} \geq \frac{1}{2}$ .

It is easy to prove, by induction, that

$$\frac{(k!)^2}{(2k)!} \leq \frac{1}{2^k}.$$

For  $\ell = r = 3 \log n$  we then get from Lemma 8.2 that:

$$\begin{aligned}p_{n,\ell,r} &\geq 1 - n \frac{(\ell!)^2}{(2\ell)!} - n\ell(n(2\ell r + \ell) - \ell) \frac{(r!)^2}{(2r)!} \\ &\geq 1 - n \frac{1}{2^\ell} - n\ell(n(2\ell r + \ell) - \ell) \frac{1}{2^r} \\ &= 1 - \frac{1}{n^2} - \frac{\ell(n(2\ell r + \ell) - \ell)}{n^2} \\ &\geq 1 - \frac{1}{n^2} - \frac{81 \log n}{n} \geq \frac{1}{2}\end{aligned}$$

for  $n$  sufficiently large.

Since, for  $\ell = r = 3 \log n$  and  $n$  sufficiently large, the number of vertices of  $G_{n,\ell,r}$  is  $|V| = n(2\ell r + \ell + 3) + 1 \leq 27n \log^2 n$ , the number of bits expressed in terms of the number of vertices is  $n = \Omega(|V|/\log^2 |V|)$ , and we get:

$$\mathbb{E}_{G_{n,\ell,r}}(E_0, \sigma) = 2^{\Omega(\sqrt{|V|}/\log |V|)}.$$

This concludes the proof.  $\square$

By Theorem 3.2 this result extends to Mean Payoff Games, Discounted Payoff Games and Turn-based Stochastic Games, and we get the following corollary.

**COROLLARY 8.1.** *The worst-case expected running time of the RANDOMFACET algorithm for  $n$ -state Mean Payoff Games, Discounted Payoff Games and Turn-based Stochastic Games is at least  $2^{\Omega(\sqrt{n}/\log n)}$ .*

## 9 Concluding remarks

We constructed explicit parity games on  $n$  vertices on which the expected running time of the RANDOMFACET algorithm is  $2^{\tilde{\Omega}(\sqrt{n})}$ , almost matching the upper bound of Matoušek, Sharir and

Welzl [MSW96]. This dashes the hope that the RANDOMFACET algorithm could be used to solve parity games, or even deterministic or stochastic mean payoff games, in polynomial time.

Our parity games also supply the first *explicit* instances on which the expected running time of the RANDOMFACET algorithm is not polynomial. Matoušek [Mat94] previously constructed a family of instances such that the expected running time of the RANDOMFACET algorithm is not polynomial when run on a *random* instance from this family. One advantage of Matoušek's non-explicit construction is that his instances correspond to *Acyclic Unique Sink Orientations* (AUSOs) of  $n$ -cubes, while our explicit instances do not. (For information on AUSOs, see [SW01],[Gär02],[GS06].) It would be interesting to see whether an extension of the method used here could be used to construct an explicit AUSO of an  $n$ -cube on which RANDOMFACET takes more than polynomial time to find the sink.

It would also be interesting to resolve the complexity of the RANDOMFACET algorithm on MDPs and LPs. Fearnley [Fea10], extending the result of Friedmann's [Fri09] for parity games, showed that the deterministic policy iteration algorithm may require exponential time on MDPs. We believe that similar techniques could be used to extend our results to MDPs and LPs.

Another natural randomized algorithm for solving parity games and other games is the RANDOMEDGE algorithm in which a random improving switch is performed at each step. Matoušek and Szabó constructed abstract instances on which the expected running time of this algorithm is  $2^{\Omega(n^{1/3})}$ . Are there parity games on which the algorithm behaves as badly? Again, we believe that the answer is yes.

The most interesting open problem is, perhaps, whether there is a polynomial time algorithm for solving parity games.

## References

- [AM09] D. Andersson and P.B. Miltersen. The complexity of solving stochastic games on graphs. In *Proc. of 20th ISAAC*, pages 112–121, 2009.
- [BSV03] H. Björklund, S. Sandberg, and S. Vorobyov. A discrete subexponential algorithm for parity games. In *Proc. of 20th STACS*, pages 663–674, 2003.
- [BV05] H. Björklund and S. Vorobyov. Combinatorial structure and randomized subexponential algorithms for infinite games. *Theoretical Computer Science*, 349(3):347–360, 2005.
- [BV07] H. Björklund and S. Vorobyov. A combinatorial strongly subexponential strategy improvement algorithm for mean payoff games. *Discrete Applied Mathematics*, 155(2):210–229, 2007.

- [Con92] A. Condon. The complexity of stochastic games. *Information and Computation*, 96:203–224, 1992.
- [EJ91] E.A. Emerson and C. Jutla. Tree automata,  $\mu$ -calculus and determinacy. In *Proceedings of the 32nd FOCS*, pages 368–377. IEEE Computer Society Press, 1991.
- [EJS93] E. A. Emerson, C. S. Jutla, and A. P. Sistla. On model-checking for fragments of  $\mu$ -calculus. In *International Conference on Computer-Aided Verification, CAV'93*, volume 697 of *LNCS*, pages 385–396, 1993.
- [EM79] A. Ehrenfeucht and J. Mycielski. Positional strategies for mean payoff games. *International Journal of Game Theory*, 8:109–113, 1979.
- [Fea10] J. Fearnley. Exponential lower bounds for policy iteration. *CoRR*, abs/1003.3418, 2010.
- [FL09] O. Friedmann and M. Lange. Solving parity games in practice. In *Proceedings of the 7th ATVA*, pages 182–196, 2009.
- [Fri09] O. Friedmann. An exponential lower bound for the parity game strategy improvement algorithm as we know it. In *Proc. of 24th LICS*, pages 145–156, 2009.
- [Fri10] O. Friedmann. An exponential lower bound for the latest deterministic strategy iteration algorithms. *Selected Papers of the Conference "Logic in Computer Science 2009" (to appear)*, 2010. A preprint available from <http://www.tcs.ifi.lmu.de/~friedman>.
- [FS09] P. Flajolet and R. Sedgewick. *Analytic Combinatorics*. Cambridge University Press, 2009.
- [Gär02] B. Gärtner. The random-facet simplex algorithm on combinatorial cubes. *Random Structures and Algorithms*, 20(3):353–381, 2002.
- [GKK88] V.A. Gurvich, A.V. Karzanov, and L.G. Khachiyan. Cyclic games and an algorithm to find minimax cycle means in directed graphs. *USSR Computational Mathematics and Mathematical Physics*, 28:85–91, 1988.
- [Gol95] M. Goldwasser. A survey of linear programming in randomized subexponential time. *SIGACT News*, 26(2):96–104, 1995.
- [GS06] B. Gärtner and I. Schurr. Linear programming and unique sink orientations. In *Proc. 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 749–757, 2006.
- [GTW02] E. Grädel, W. Thomas, and T. Wilke, editors. *Automata, Logics, and Infinite Games. A Guide to Current Research*, volume 2500 of *LNCS*. Springer, 2002.
- [Hal07] N. Halman. Simple stochastic games, parity games, mean payoff games and discounted payoff games are all LP-type problems. *Algorithmica*, 49(1):37–50, 2007.
- [HK66] A. Hoffman and R. Karp. On nonterminating stochastic games. *Management Science*, 12:359–370, 1966.
- [How60] R.A. Howard. *Dynamic programming and Markov processes*. MIT Press, 1960.
- [JPZ08] M. Jurdziński, M. Paterson, and U. Zwick. A deterministic subexponential algorithm for solving parity games. *SIAM Journal on Computing*, 38(4):1519–1532, 2008.
- [Jur98] M. Jurdziński. Deciding the winner in parity games is in  $UP \cap co-UP$ . *Information Processing Letters*, 68(3):119–124, 1998.
- [Jur00] M. Jurdziński. Small progress measures for solving parity games. In *Proc. of 17th STACS*, pages 290–301, 2000.
- [Kal92] G. Kalai. A subexponential randomized simplex algorithm (extended abstract). In *Proc. of 24th STOC*, pages 475–482, 1992.
- [Kal97] G. Kalai. Linear programming, the simplex algorithm and simple polytopes. *Mathematical Programming*, 79:217–233, 1997.
- [Lud95] W. Ludwig. A subexponential randomized algorithm for the simple stochastic game problem. *Information and Computation*, 117(1):151–155, 1995.
- [Mat94] J. Matoušek. Lower bounds for a subexponential optimization algorithm. *Random Structures and Algorithms*, 5(4):591–608, 1994.
- [MS06] J. Matoušek and T. Szabó. RANDOM EDGE can be exponential on abstract cubes. *Advances in Mathematics*, 204(1):262–277, 2006.
- [MSW96] J. Matoušek, M. Sharir, and E. Welzl. A subexponential bound for linear programming. *Algorithmica*, 16(4-5):498–516, 1996.
- [MTZ10] O. Madani, M. Thorup, and U. Zwick. Discounted deterministic Markov decision processes and discounted all-pairs shortest paths. *ACM Transactions on Algorithms*, 6(2):1–25, 2010.
- [Pur95] A. Puri. *Theory of Hybrid Systems and Discrete Event Simulation*. PhD thesis, University of California, Berkeley, 1995.
- [PV01] V. Petersson and S.G. Vorobyov. A randomized subexponential algorithm for parity games. *Nord. J. Comput.*, 8(3):324–345, 2001.
- [Sch08] S. Schewe. An optimal strategy improvement algorithm for solving parity and payoff games. In *Proc. of 17th CSL*, pages 369–384, 2008.
- [Sti95] C. Stirling. Local model checking games. In *Proceedings of the 6th International Conference on Concurrency Theory (CONCUR'95)*, volume 962 of *LNCS*, pages 1–11. Springer, 1995.
- [SW01] T. Szabó and E. Welzl. Unique sink orientations of cubes. In *Proc. of 42th FOCS*, pages 547–555, 2001.
- [VJ00] J. Vöge and M. Jurdziński. A discrete strategy improvement algorithm for solving parity games (Extended abstract). In *International Conference on Computer-Aided Verification, CAV 2000*, volume 1855 of *LNCS*, pages 202–215, 2000.
- [Vög00] J. Vöge. *Strategiesynthese für Paritätsspiele auf endlichen Graphen*. PhD thesis, University of Aachen, 2000.
- [Zie98] W. Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theoretical Computer Science*, 200:135–183, 1998.
- [ZP96] U. Zwick and M.S. Paterson. The complexity of mean payoff games on graphs. *Theoretical Computer Science*, 158(1–2):343–359, 1996.

## A Proof of Lemma 4.1

*Proof.* By lexicographic induction on  $(|F|, \text{val}_\sigma)$ . Let  $H = G_F$ . For  $|F| = |\sigma|$ , we clearly have  $\mathbb{E}(H, \sigma) = \mathbb{E}^*(H, \sigma) = 1$ . For the induction step, let  $|F| > |\sigma|$ . It is easy to see that

$$\mathbb{E}(H, \sigma) = \frac{1}{|F \setminus \sigma|} \sum_{e \in F \setminus \sigma} \mathbb{E}(H, e, \sigma)$$

$$\mathbb{E}(H, e, \sigma) = \mathbb{E}(H \setminus \{e\}, \sigma) + \mathbb{1}_{e \in \sigma_H} \cdot \mathbb{E}(H, \sigma_{H \setminus \{e\}}[e])$$

where  $\mathbb{1}_{\text{pred}} \in \{0, 1\}$  denotes the *indicator function*, i.e.,  $\mathbb{1}_{\text{pred}} = 1$  iff *pred* holds.

For an index function  $i$ , let  $e_i = \text{argmin}_{e' \in F \setminus \sigma} i(e')$  and  $H_i = H \setminus \{e_i\}$ . Similarly we have the following.

$$\mathbb{E}^*(H, \sigma) = \frac{1}{|\mathcal{I}(H)|} \sum_{i \in \mathcal{I}(H)} \mathbb{E}^*(H, \sigma, i)$$

$$\mathbb{E}^*(H, \sigma, i) = \mathbb{E}^*(H_i, \sigma, i) + \mathbb{1}_{e_i \in \sigma_H} \cdot \mathbb{E}^*(H, \sigma_{H_i}[e_i], i)$$

Finally the following holds.

$$\begin{aligned} \mathbb{E}^*(H, \sigma) &= \frac{1}{|\mathcal{I}(H)|} \sum_{i \in \mathcal{I}(H)} \mathbb{E}^*(H_i, \sigma, i) + \\ &\quad \frac{1}{|\mathcal{I}(H)|} \sum_{i \in \mathcal{I}(H)} \mathbb{1}_{e_i \in \sigma_H} \cdot \mathbb{E}^*(H, \sigma_{H_i}[e_i], i) \\ &= \sum_{e \in F \setminus \sigma} \frac{1}{|F \setminus \sigma|} \sum_{i \in \mathcal{I}(H \setminus \{e\})} \frac{\mathbb{E}^*(H \setminus \{e\}, \sigma, i)}{|\mathcal{I}(H \setminus \{e\})|} + \\ &\quad \sum_{e \in F \setminus \sigma} \frac{\mathbb{1}_{e \in \sigma_H}}{|F \setminus \sigma|} \sum_{i \in \mathcal{I}(H)} \frac{\mathbb{E}^*(H, \sigma_{H \setminus \{e\}}[e], i)}{|\mathcal{I}(H)|} \\ &\stackrel{IH}{=} \sum_{e \in F \setminus \sigma} \frac{1}{|F \setminus \sigma|} \mathbb{E}(H \setminus \{e\}, \sigma) + \\ &\quad \sum_{e \in F \setminus \sigma} \frac{\mathbb{1}_{e \in \sigma_H}}{|F \setminus \sigma|} \mathbb{E}(H, \sigma_{H \setminus \{e\}}[e]) = \mathbb{E}(H, \sigma) \end{aligned}$$

## B Proof of Lemma 5.2

*Proof.* It is not hard to see that  $f_n(\emptyset, \varphi) = 1$  and  $f_n(N, \varphi) = 1 + f_n(N \setminus \{i\}, \varphi) + f_n(N \cap [i], \varphi)$  for  $N \neq \emptyset$  and  $i = \text{argmin}_{j \in N} \varphi(j)$ . We also have

$$f_n(N) = \frac{1}{|\mathcal{S}(n)|} \sum_{\varphi \in \mathcal{S}(n)} f_n(N, \varphi)$$

We are now ready to show  $f_n([n]) = g(n)$  by induction on  $n$ . The claim obviously holds true for  $n = 0$ . Let now  $n > 0$  and  $i = \varphi^{-1}(1)$ . The following holds.

$$\begin{aligned} f_n([n]) &= \frac{1}{|\mathcal{S}(n)|} \sum_{\varphi \in \mathcal{S}(n)} 1 + f_n([n] \setminus \{i\}, \varphi) + f_n([i], \varphi) \\ &= 1 + \frac{1}{|\mathcal{S}(n-1)|} \sum_{\varphi \in \mathcal{S}(n-1)} f_{n-1}([n-1], \varphi) + \\ &\quad \frac{1}{|\mathcal{S}(n)|} \sum_{\varphi \in \mathcal{S}(n)} f_i([i], \varphi|_{\varphi([i])}) \\ &= 1 + g(n-1) + \frac{1}{n} \sum_{i=0}^{n-1} \frac{1}{|\mathcal{S}(i)|} \sum_{\varphi \in \mathcal{S}(i)} f_i([i], \varphi) \\ &= 1 + g(n-1) + \frac{1}{n} \sum_{i=0}^{n-1} g(i) = g(n) \end{aligned}$$

□

## C Proof of Lemma 7.2

*Proof.* We only provide the proof for the first three relations. The proof of the other three relations is analogous.

We consider three cases. In each case we present the optimal choices and verify that they are, indeed, optimal. First, if  $A_F^3(c_i) \succ A_F^3(b_{i+1,1})$  and  $b_i(F) = 1$ , then  $\tau_F^*(B_i) = c_i$  and  $\beta_i^*(F) = 1$ . To verify this we observe that:

$$\begin{aligned} A_F(B_i) &= \{2\} \cup A_F(c_i) \\ A_F(b_{i,j}) &= \{2, 2\} \cup A_F(c_i) \quad \text{for all } j \in [lr] \end{aligned}$$

Hence,  $A_F(c_i) \prec A_F(b_{i,j})$  for all  $j \in [lr]$ , and  $A_F^3(b_{i+1,1}) \prec A_F^3(B_i)$ . It follows that no player can gain by changing strategy.

Next, consider the case where  $A_F^3(c_i) \succ A_F^3(b_{i+1,1})$  and  $\beta_i^*(F) = 0$ . There exists a  $j \in [lr]$  such that  $(b_{i,j}, B_i) \notin F$ . Then  $\tau_F^*(B_i) = b_{i,j}$ , and  $\beta_i^*(F) = 1$ . To verify this we observe that:

$$\begin{aligned} A_F(B_i) &= \{2, 2\} \cup A_F(b_{i+1,1}) \\ A_F(b_{i,j}) &= \begin{cases} \{2, 2, 2\} \cup A_F(b_{i+1,1}) & \text{if } (b_{i,j}, B_i) \in F \\ \{2\} \cup A_F(b_{i+1,1}) & \text{otherwise} \end{cases} \end{aligned}$$

It again follows that no player can gain by changing strategy.

If  $A_F^3(c_i) \prec A_F^3(b_{i+1,1})$ , then  $\tau_F^*(B_i) = c_i$  and  $\beta_i^*(F) = 0$ . It follows that:

$$\begin{aligned} A_F^3(B_i) &= A_F^3(c_i) \\ A_F^3(b_{i,j}) &= A_F^3(b_{i+1,1}) \quad \text{for all } j \in [lr] \end{aligned}$$

Hence, no player can gain by changing strategy. □

## D Proof of Lemma 7.1

*Proof.* We first consider the case where  $i \geq \text{reset}(F)$ . To prove the lemma we simply go through the vertices using induction, observing at each vertex the optimal choice and the obtained valuation. More precisely, we use backward induction on  $i$ , with induction hypothesis  $A_F^4(A_{i+1,j}) \preceq A_F^4(b_{i+1,1})$ , for all  $j \in [\ell]$ , with equality for some  $j$ . For the base case,  $\top$  takes the role as both  $b_{n+1}$  and  $A_{n+1,j}$ , for all  $j \in [\ell]$ , and the statement is clearly correct. We then observe the following:

1. Induction hypothesis:

$$\begin{aligned} \forall j \in [\ell] : A_F^4(A_{i+1,j}) &\preceq A_F^4(b_{i+1,1}) \\ \exists j' \in [\ell] : A_F^4(A_{i+1,j'}) &= A_F^4(b_{i+1,1}) \end{aligned}$$

2.  $c_i$  moves to  $A_{i+1,j'}$ , where  $j'$  is defined in 1.:

$$\begin{aligned} \sigma_F^*(c_i) &= A_{i+1,j'} \\ A_F^4(c_i) &= \{2i+2\} \cup A_F^4(b_{i+1,1}) \end{aligned}$$

3.  $\beta_i^*(F) = 1$ , by Lemma 7.2.

4. If  $b_i(F) = 1$ , then:

- a. By 2. and Lemma 7.2:

$$\begin{aligned} A_F^4(B_i) &= \{2i+2\} \cup A_F^4(b_{i+1,1}) \\ A_F^4(b_{i,1}) &= \{2i+2\} \cup A_F^4(b_{i+1,1}) \end{aligned}$$

- b.  $A_F^4(D_i) = \{2i+2\} \cup A_F^4(b_{i+1,1})$ .
- c.  $\alpha_i^*(F) = 1$ , by Lemma 7.2.
- d. If  $a_i(F) = 1$ , then by 4.a., 4.b. and Lemma 7.2:

$$\begin{aligned} \forall j \in [\ell] : A_F^4(A_{i,j}) &\preceq A_F^4(b_{i,1}) \\ \exists j' \in [\ell] : A_F^4(A_{i+1,j'}) &= A_F^4(b_{i,1}) \end{aligned}$$

- e. If  $a_i(F) = 0$ , then by Lemma 7.2:

$$\forall j \in [\ell] : A_F^4(A_{i,j}) = A_F^4(b_{i+1,1})$$

5. If  $b_i(F) = 0$ , then:

- a. By Lemma 7.2:

$$\begin{aligned} A_F^3(B_i) &= A_F^3(b_{i+1,1}) \\ A_F^3(b_{i,1}) &= A_F^3(b_{i+1,1}) \end{aligned}$$

- b.  $A_F^3(D_i) = \{3\} \cup A_F^3(b_{i+1,1})$ .
- c.  $\alpha_i^*(F) = 0$ , by Lemma 7.2.
- d.  $\forall j \in [\ell] : A_F^4(A_{i,j}) = A_F^4(b_{i+1,1})$ .

Note that the statement of the lemma follows from 3., 4.c. and 5.c.. For  $i > \text{reset}(F)$ , the induction step follows from 4.d. and 5.d..

For  $i = \text{reset}(F)$  we have  $A_F^4(A_{i,j}) = A_F^4(b_{i+1,1})$  and  $A_F^4(b_{i,1}) = \{2i+2\} \cup A_F^4(b_{i+1,1})$ , from 4.e. and 4.a., respectively. We use this as the basis for continuing using backward induction on  $i$ , for  $i < \text{reset}(F)$ . In the following let  $k = 2 \cdot \text{reset}(F) + 2$ . We use the induction hypothesis  $A_F^k(b_{i+1,1}) = \{k\} \cup A_F^k(A_{i+1,j})$ , for all  $j$ . We observe:

6. Induction hypothesis:

$$\forall j \in [\ell] : A_F^k(b_{i+1,1}) = \{k\} \cup A_F^k(A_{i+1,j})$$

7.  $A_F^k(c_i) = A_F^k(A_{i+1,j'})$ , for some  $j'$ .

8. By Lemma 7.2:

$$\begin{aligned} \beta_i^*(F) &= 0 \\ A_F^k(B_i) &= A_F^k(A_{i+1,j'}) \\ A_F^k(b_{i,1}) &= \{k\} \cup A_F^k(A_{i+1,j'}) \end{aligned}$$

9.  $A_F^k(D_i) = A_F^k(A_{i+1,j'})$ .

10. By Lemma 7.2:

$$\begin{aligned} \alpha_i^*(F) &= 0 \\ \forall j \in [\ell] : A_F^k(A_{i,j}) &= A_F^k(A_{i+1,j'}) \end{aligned}$$

Note that the statement of the lemma, as well as the induction step, follows from 8. and 10..  $\square$