Randomized Subexponential Algorithms for Infinite Games

Henrik Björklund, Sven Sandberg, and Sergei Vorobyov

Department of Information Technology
Uppsala University
Box 337, SE-751 05 Uppsala, Sweden

Technical report 2004-011
April 2004
ISSN 1404-3203

Randomized Subexponential Algorithms for Infinite Games

Henrik Björklund, Sven Sandberg, and Sergei Vorobyov *

Information Technology Department, Uppsala University, Box 337, SE-751 05 Uppsala, Sweden

Abstract

The complexity of solving infinite games, including parity, mean payoff, and simple stochastic games, is an important open problem in verification, automata theory, and complexity theory. In this paper we develop an abstract setting for studying and solving such games, as well as related problems, based on function optimization over certain discrete structures. We introduce new classes of *completely local-global (CLG)* and *recursively local-global (RLG)* functions, and show that strategy evaluation functions for parity and simple stochastic games belong to these classes. We also establish a relation to the previously well-studied *completely unimodal (CU)* and *local-global* functions. A number of nice properties of CLG-functions are proved.

In this setting, we survey several randomized optimization algorithms appropriate for CU-, CLG-, and RLG-functions. We show that the subexponential algorithms for linear programming by Kalai and Matoušek, Sharir, and Welzl, can be adapted to optimizing the functions we study, with preserved subexponential expected running time. We examine the relations to two other abstract frameworks for subexponential optimization, the LP-type problems of Matoušek, Sharir, Welzl, and the abstract optimization problems of Gärtner. The applicability of our abstract optimization approach to parity games builds upon a discrete strategy evaluation measure.

We also consider local search type algorithms, and settle two nontrivial, but still exponential, upper bounds. As applications we address some complexity-theoretic issues including non-PLS-completeness of the problems studied.

⋆ Supported by Swedish Research Council grants "*Infinite Games: Algorithms and Complexity*", "*Interior-Point Methods for Infinite Games*", and by a grant from the Swedish Foundation for International Cooperation in Research and Higher Education.

* Corresponding author. Phone: +46 18 471 10 55. Fax: +46 18 55 02 25
Email addresses: henrikbj@it.uu.se (Henrik Björklund), svens@it.uu.se (Sven Sandberg), vorobyov@csd.uu.se (Sergei Vorobyov).

Contents

1 Introduction

The theory of infinite two-person adversary full information games is a well established framework for modeling interaction between a system and its environment. A correct system can be interpreted as a player possessing a winning strategy against any strategy of the malicious environment. Conversely, a verification process can be considered as a proof that a system possesses such a strategy. If the system loses, then a winning strategy for the environment encompasses possible system improvements. During the last decades substantial progress has been achieved both in fitting diverse approaches to computer-aided verification into the game-theoretic paradigm and in developing efficient algorithms for solving games, i.e., determining the winner and the winning strategy [19,11,26,34,46,5]. A natural approach to solving games [30,13,42,46] is to take an initial strategy of the system and gradually "improving" it, since a non-optimal strategy is outperformed by some counterstrategy of the environment. This allows us to improve either the strategy, the system, or both. In this paper we address the complexity of such approaches in an abstract model based on optimizing functions on sets of game strategies.

Game theory suggests, for numerous types of games, a nice characterization of the optimal solutions for behaviors of rational players, the so-called *Nash equilibria*. One of the major remaining problem left widely open is: "*What is the exact computational complexity of finding Nash equilibria (optimal strategies) in two-person games with finitely many strategies? Could such games be solved in polynomial time?*" This is a fundamental problem, the solution to which determines whether or not, and to what extent game theory will be applicable to solving and optimizing real large-scale games emerging from practical verification of complex reactive systems [40].

Our primary motivation consist in creating a unified theory of abstract optimization appropriate for solving different classes of infinite games, including parity, mean payoff, and simple stochastic games. It turns out that despite differences, these games possess common structural properties that allow us to abstract away irrelevant features when studying iterative improvement algorithms. This leads to a simple, general, and powerful optimization theory.

1. Parity games are infinite games played on finite directed leafless graphs, with vertices colored by integers. Two players move a pebble along edges of the graph, thus forming an infinite path. Vertices are partitioned between the players, and the owner of the vertex currently visited by the pebble decides where to move it next. The goal of Player 0 is to ensure that the biggest color visited by the pebble infinitely often is even, whereas Player 1 tries to make it odd. Deciding the winner in parity games is equivalent to the Rabin chain tree automata non-emptiness, as well as to the μ-calculus model checking [20,19].

The μ-calculus is one of the most expressive temporal logics of programs [19]. The problem is therefore significant in verification and automata theory, and also from a complexity-theoretic point of view, since it belongs to the complexity class NP∩coNP, but is not known to belong to P.

2. Mean payoff games (MPGs) [17,27,51] are another important motivation for studying iterative improvement. They are similar to parity games, but instead of colored vertices they have weighted edges, and the players try to maximize, respectively minimize, the average edge weight in the limit. Parity games reduce to mean payoff games, and like for parity games, the decision problem for mean payoff games belongs to NP ∩ coNP. Zwick and Paterson [51] show how MPGs can be used to design and analyze algorithms for job scheduling, finite-window online string matching, and selection with limited storage.

3. Simple stochastic games (SSGs) are relevant for verification of probabilistic systems. They were introduced by Shapley and have been well-studied since the 1950's; see, e.g., [30,16,12,13,36,51]. SSGs are important generalizations of Markov decision processes. Parity and mean payoff games both reduce to simple stochastic games. The decision problem also belongs to NP ∩ coNP.

Our theory resulted in the first strongly subexponential randomized algorithms for parity [6] and mean payoff games [8], as well as in a randomized subexponential algorithm for simple stochastic games of arbitrary outdegree [7]. [1]

The basic concept underlying all algorithms discussed in this paper is to assign values to strategies of one of the players, and then search the strategy space guided by these values. This approach relies on the fundamental fact that the games studied are determined and solvable in *positional* strategies. This was first proved for mean payoff games by Ehrenfeucht and Mycielski [17,18], and for parity games by Emerson and Jutla [20]. Simple stochastic games have the same property; see, e.g., [31,12,21]. It means that each vertex of every such game has a value, and that both players have positional (memoryless) strategies that secure this value, whenever the game starts in the vertex. Solving the game amounts to finding these values and strategies. We can thus limit our search to the *finite* set of all positional strategies.

A positional strategy of one player maps each of the player's vertices to one of its successors. Using the positional strategy means to always select this successor, independently of the history of the play. The set of all positional strategies of a player can thus be seen as a Cartesian product of finite sets (for each vertex, the set of its successors is a factor in the product). Given an appropriate way of assigning values to strategies, we can reduce the problem of solving games with positional determinacy to function optimization over

[1] The algorithm of [36] is only subexponential for SSGs of *bounded* outdegree.

Cartesian products.

In particular, if the player has at most two choices in each vertex, the space of all positional strategies is isomorphic to a Boolean hypercube. The resulting functions resemble those studied in the field of pseudo-Boolean optimization [10,29], in which local search [1,45] is one of the dominating methods. In general, pseudo-Boolean functions cannot be efficiently optimized [45], but for some subclasses, in particular the *completely unimodal* (CU) functions [28,49,48,45], the situation is far more promising. They can be optimized in subexponential time using an adaptation of Ludwig's algorithm for simple stochastic games [4], based on ideas of Kalai [35] and Matoušek, Sharir, and Welzl [43,39,38]. A long-standing conjecture [49] claims that they can be optimized in polynomial time.

There is a well-known function for assigning values to strategies in simple stochastic games [30,13,36], with properties close to the CU-functions [41]. There is also a reduction from parity games via mean payoff games to simple stochastic games [42,51]. Thus parity and mean payoff games can be solved by reduction followed by optimization of the simple stochastic game value function. This approach has a downside, however. Each function evaluation involves solving a linear program with high precision. This is costly, both with regard to running time and development (typical off-the-shelf linear programming modules do not provide for the necessary precision). The randomized algorithm for simple stochastic games by Ludwig [36], is subexponential in the number of vertices of the game, assuming that the outdegree of vertices is *at most two*. Unfortunately, reduction of general parity games to binary ones may increase the number of vertices, making the algorithm exponential.

Vöge and Jurdziński developed an algorithm for parity games [46], in which the major invention was a scheme for assigning discrete values to strategies. This scheme, as the authors noted in [47], lends itself to use together with a variety of algorithms besides the one originally suggested. It also fits nicely into the setting of function optimization, and avoids linear programming procedures and high-precision floating point calculations. In [5] we suggested a modification of the evaluation function from [46]. It is also presented in this survey, and provides for better complexity analysis of some algorithms.

Until recently, no convenient (discrete) strategy evaluation function for mean payoff games was known. In this survey we present a new direct and discrete strategy evaluation for mean payoff games, first discovered in [8]. Together with the algorithms from LP-type optimization, this shows that mean payoff games can be solved in strongly subexponential expected time, without the need for high-precision arithmetic. The evaluation measure also provides, by reduction, a new and efficient strategy improvement scheme for parity games.

In this paper we generalize pseudo-Boolean functions and develop the abstract setting of *hyperstructure* optimization. The purpose is to facilitate the development and study of different algorithms for games. We describe some classes of functions and their relation to the strategy evaluation functions. Some interesting properties of these functions are presented and compared with previously studied classes of pseudo-Boolean functions.

In the abstract setting we show that the recursive, subexponential, randomized algorithms for linear programming due to Kalai [35] and Matoušek, Sharir, and Welzl [38,39] can be adapted to optimize a wide class of functions on hyperstructures, including the strategy evaluation functions for parity and simple stochastic games. We also study how these functions reduce to the LP-type problems of Matoušek, Sharir, and Welzl [38] and the Abstract Optimization Problems of Gärtner [22]. Local search-like algorithms are also considered, and we settle upper bounds for two randomized algorithms optimizing hypercubes, corresponding to binary games.

Johnson, Papadimitriou, and Yannakakis [32,50] introduced the complexity class PLS (Polynomial Time Local Search), lying somewhere between PTIME and NP, and identified a lot of well-known problems as being PLS-complete (including Max-Cut, Max2Sat, Max3Sat under the Flip neighborhood, as well as TSP under k-Opt and Kernighan-Lin's neighborhoods). As usual with the completeness theory, PLS-complete problems are either all polynomial or all superpolynomial. Yannakakis [50] raised a question whether simple stochastic games are PLS-complete. A similar question is appropriate for parity games. It is a simple consequence of our theory (Section 11.1) that Completely Local Global functions (and henceforth parity and simple stochastic games) are not PLS-complete, which gives new evidence in favor of the PTIME conjecture.

Outline. After preliminaries on games and pseudo-Boolean optimization in Section 2, we develop the framework of hyperstructures in Section 3. Section 4 presents three classes of functions on hyperstructures relevant for games, and some of their properties. Randomized subexponential algorithms for optimizing these functions are investigated in Section 5. Section 6 deals with more conventional local search-style algorithms. Section 7 presents the basics of iterative strategy improvement for games. Strategy evaluation functions for mean payoff and parity games are described in Sections 8 and 9, respectively. The functions for parity and simple stochastic games are proved to belong to one of the studied function classes in Section 10. Finally, Section 11 discusses some structural complexity-theoretic issues.

2 Preliminaries

Parity, mean payoff, and simple stochastic games are all full information, two-person, adversary games played on finite graphs. Parity and mean payoff games are infinite duration games, whereas simple stochastic games may be finitely terminating or not. We only consider finite explicitly represented graphs, because we are concerned with the complexity of decision algorithms.

2.1 Parity Games

We first define and discuss parity games in detail, and then proceed to mean payoff and simple stochastic games, explaining which properties of parity games hold for them too.

Definition 2.1 *A* parity game (PG) *is played between Player 0 and 1 on a vertex-colored leafless graph $G = (V = V_0 \cup V_1, E, k, c)$, where:*

- *(V, E) is a finite, directed graph with no sinks (every vertex has out-degree at least one),*
- *vertices in V_0 are controlled by Player 0 and vertices in V_1 by Player 1,*
- *$k \in \mathbb{N}$, and $c : V \to \{1, \ldots, k\}$ is the* coloring function. □

The two players take turns moving a token along outgoing edges. The token starts at some distinguished vertex, and the player who controls the current vertex selects a successor. Thus, they form an infinite sequence of vertices, called a *play*. Player 0 wins if the largest color occurring infinitely often in the play is even, and Player 1 wins if it is odd. Formally, one can define a *strategy* of Player 0 as a function $f : V^* \to V$, such that for any finite play $v_1 v_2 \ldots v_j \in V^*$ with $v_j \in V_0$, we have $(v_j, f(v_1, \ldots, v_j)) \in E$. A play is *consistent* with a strategy if $v_{i+1} = f(v_1, \ldots, v_i)$ for all i such that $v_i \in V_0$. A strategy is *winning* for Player 0 (Player 1) with respect to a given starting vertex, iff the highest color occurring infinitely often in any play consistent with it is even (odd). A player that has a winning strategy *wins the game* (with respect to a starting vertex). The problem we are concerned with is to compute the winner of the game from each vertex.

Convention 2.2 *Throughout the paper we systematically use the notation introduced in Definition 2.1. We also let $n_0 = |V_0|$, $n_1 = |V_1|$, and $n = |V|$. Sometimes we call the game $G = (V, E)$, when other components are clear from the context or not essential for the discussion.* □

Positional strategies are fundamental for parity games.

Definition 2.3 *In a parity game, a* positional strategy *for Player 0 is a function $\sigma : V_0 \to V$, such that $(v, \sigma(v)) \in E$ for all $v \in V_0$. Positional strategies for Player 1 are defined symmetrically.* □

A player *fixes* or *uses* a positional strategy by deterministically choosing the same successor $\sigma(v)$ each time the play comes to v, independently of the history of the play. Parity games are determined and solvable in positional strategies.

Theorem 2.4 ([20]) *In every parity game, the vertices can be partitioned into winning sets W_0 and W_1 of Player 0 and 1 such that both players have positional strategies that win any play starting from a vertex in their respective winning sets.* □

Therefore, the players can restrict themselves to positional strategies, and it is no disadvantage to reveal the strategy to the opponent in advance. Consequently, in the sequel by a "strategy" we will always mean a "positional strategy".

Given a strategy σ of Player 0, an optimal counterstrategy $\tau(\sigma)$ of Player 1 against σ can be found in polynomial time [46,6] (one such method is given by Algorithm 5 of Section 9.5). The counterstrategy $\tau(\sigma)$ is used to improve σ by making one or several local changes (*switches*) in σ; this process iterates until an optimal strategy is found. Note that improving the strategy of Player 0 (rather than of Player 1) is just a matter of convention. Indeed, parity games are *symmetric* in the sense that the roles of players can be reversed: if we add one to all colors and interchange V_{MIN} and V_{MAX}, then Player 0 wins the resulting game iff Player 1 won the original game. Consequently, throughout the paper we take the viewpoint of Player 0 and iteratively improve only strategies of Player 0. However, if Player 1 has fewer strategies to choose from, we can reverse the game before solving it, which reduces the search space.

All algorithms described in this paper are *iterative strategy improvement algorithms* on positional strategies, proceeding from a current trial positional strategy to another one that is "better", by making *single* or *multiple switches*, until an optimal strategy is reached.

Definition 2.5 (Single Switch) *If σ is a strategy of Player 0, $x \in V_0$ and $(x, y) \in E$, then the* (single) switch in x to y *changes σ to the new strategy $\sigma[x \mapsto y]$, defined as*

$$\sigma[x \mapsto y](v) \stackrel{def}{=} \begin{cases} y, & \text{if } v = x; \\ \sigma(v), & \text{otherwise.} \end{cases}$$ □

Definition 2.6 (Multiple Switch) *A* multiple switch *in a positional strategy σ of Player 0 is the successive application $\sigma[x_1 \mapsto y_1] \cdots [x_j \mapsto y_j]$ of several single switches in different vertices.* □

In Sections 5–6 we study algorithms that optimize certain classes of functions, mainly motivated by the application to parity games. Indeed, we show in Section 9 that there are appropriate evaluation functions on positional strategies in parity games that match the studied classes.

Some of our algorithms recurse on substructures corresponding to subgames.

Definition 2.7 (Subgames) *A* subgame *is obtained from a parity game by throwing away some edges, without creating sinks.*

Let G be a parity game and σ a positional strategy. The subgame G_σ is obtained from G by deleting all edges leaving vertices in V_0 not selected by σ. □

The special class of binary parity games is interesting because the spaces of possible strategies in such games are isomorphic to Boolean hypercubes.

Definition 2.8 (Binary Parity Game) *A* binary *parity game is a game where the vertex outdegree is at most two.* □

Some proofs use a finite variant of parity games, where the play stops as soon as some vertex is revisited and the maximal color on the resulting cycle determines the winner. Thus, for a play $v_1 v_2 \dots v_r v_{r+1} \dots v_s$ where $v_r = v_s$ is the first vertex repetition, the value is $\max_{r \le i \le s} c(i)$. The following is well-known [18,46,9].

Theorem 2.9 *The value of every vertex in the finite-duration version of parity games equals its value in the infinite-duration version.* □

The next corollary will be used implicitly throughout the paper.

Proposition 2.10 *A positional strategy σ of Player 0 wins in a vertex v, if and only if on every simple loop reachable from v in G_σ, the largest color is even.* □

2.2 Mean Payoff and Simple Stochastic Games

Mean payoff and simple stochastic games are similar and closely related to parity games. We will present algorithms for all three types of games. Most of

the discussion about parity games applies to mean payoff and simple stochastic games; see below for details.

Definition 2.11 *A* mean payoff game (MPG) *is played between players* Min *and* Max *on an edge-weighted leafless graph* $G = (V = V_{\text{Max}} \cup V_{\text{Min}}, E, w)$*, where*

- (V, E) *is a finite, directed graph with no sinks,*
- *vertices in* V_{Max} *are controlled by* Max *and vertices in* V_{Min} *by* Min*,*
- $w : E \to \mathbb{Z}$ *is the* weight function. □

Mean payoff games are played in the same way as parity games, but the goals of players are different. The play, here, is the resulting infinite sequence of *edges*, and the *value* of a play $e_1 e_2 e_3 \dots$ is defined as [2] $\lim_{i\to\infty} 1/i \cdot \sum_{j=1}^{i} w(e_j)$. The goal of Max is to maximize the value of the play, while Min tries to minimize it. A parity game can be transformed to an equivalent mean payoff game (Player 0 wins the parity game, iff Max can guarantee value > 0 in the mean payoff game), by labeling each outgoing edge from a vertex v by $(-n)^{c(v)}$.

Convention 2.12 *Throughout the paper, denote by* $W \stackrel{def}{=} \max_{e\in E} |w(e)|$ *the largest absolute value of a weight in the mean payoff game.*

Definition 2.13 *A* simple stochastic game (SSG) *is played between players* Min *and* Max *on a graph* $G = (V = V_{\text{Max}} \cup V_{\text{Min}} \cup V_{\text{Avg}} \cup \{0, 1\}, E, w)$*, where* (V, E) *is a directed graph with vertices partitioned into the following.*

- *Vertex sets* V_{Max} *and* V_{Min} *controlled by* Max *and* Min*, respectively;*
- *"random vertices"* V_{Avg}*;*
- *the 0-sink and the 1-sink.*

All vertices in $V_{\text{Max}} \cup V_{\text{Min}} \cup V_{\text{Avg}}$ *have at least one successor, while 0 and 1 have none. In addition, the function* $w : E \cap (V_{\text{Avg}} \times V) \to \mathbb{Q}$ *is a rational probability distribution over the successors of every random vertex. That is, for every* $v \in V_{\text{Avg}}$*, one has* $\sum_{(v,u)\in E} w(v, u) = 1$*.* □

Simple stochastic games are played similarly as parity and mean payoff games, except (1) the next move in random vertices is selected randomly according to w and (2) the game ends when one of the sinks is reached. Max wants to maximize the probability of reaching the 1-sink and Min wants to minimize it. To make the game zero-sum, we assume (without loss of generality [12])

[2] Formally, the limit may not be defined and one should replace "lim" by "lim inf". However, both players can restrict themselves to positional strategies, for which the limit is well-defined; see below.

that for any pair of positional strategies, the probability of reaching the 0-sink plus the probability of reaching the 1-sink is one. The value of a vertex under a pair of strategies of MAX and MIN is thus the probability that the 1-sink is reached. It can be shown that this value is well-defined.

Both general and positional strategies for these games are defined as for parity games. We have the following slightly sharper positional determinacy result.

Theorem 2.14 *In a mean payoff or simple stochastic game, every vertex has a value* $\mathrm{val}(v)$*, and the players possess positional strategies* σ, τ*, guaranteeing value* $\geq \mathrm{val}(v)$ *and* $\leq \mathrm{val}(v)$*, respectively, against any strategy of the opponent, when play starts in* v*.* □

Mean payoff and simple stochastic games are also symmetric; the roles of the players can be interchanged with a simple transformation. Single and multiple switches, subgames, G_σ, and binary games, are all defined similarly. Mean payoff games, *but not simple stochastic games* have finite-duration versions. Here, for a play $e_1 e_2 \dots e_r e_{r+1} \dots e_s$ where $e_r \dots e_s$ is a loop, the value is $\sum_{i=r}^{s} w(e_i)/(s-r+1)$. Ehrenfeucht and Mycielski [18] proved that the values in the finite and infinite versions coincide.

If we are given a strategy of one player in a mean payoff or simple stochastic game, the optimal counterstrategy of the other player can be computed in polynomial time. This is equivalent to solving a one-player game (where either V_{MAX} or V_{MIN} is empty); see [51,36].

2.3 Algorithmic Problems for MPGs and SSGs

We will address several computational problems for mean payoff and simple stochastic games.

The Decision Problem. Given a distinguished start vertex and a threshold value p, can MAX guarantee value $> p$?

p-Threshold Partition. Given p, partition the vertices of G into subsets $G_{\leq p}$ and $G_{>p}$ such that MAX can guarantee a value $> p$ starting from every vertex in $G_{>p}$, and MIN can secure a value $\leq p$ starting from every vertex in $G_{\leq p}$.

Ergodic Partition. Compute the value of each vertex of the game. This gives a partition of the vertices into subsets with the same value. Such a partition is called *ergodic* [27].

Strategy improvement algorithms for simple stochastic games solve the most general ergodic partition problem and also compute the optimal strategies. This contrasts with mean payoff games, where only the *0-threshold partition*

problem is solved directly. This subsumes the the p-threshold partition and hence also the decision problem. Indeed, subtracting p from the weight of every edge makes the mean value of all loops (in particular, of optimal loops) smaller by p, and the problem reduces to 0-threshold partitioning. The complexity remains the same for integer thresholds p, and changes slightly for rational ones; see Section 8.6, which also explains how to extend the basic algorithm to solve the ergodic partition problem. Another problem to which our algorithm is easily extended is finding optimal strategies in MPGs.

2.4 Optimization on Boolean Hypercubes

A function from the Boolean hypercube $\mathcal{H} = \{0,1\}^d$ to $\mathbb{R}$ is called *pseudo-Boolean*. Optimizing such a function is a problem of finding the corner of the hypercube with the maximal (or minimal) function value. This problem has been extensively studied; see, e.g., [28,48,44,49,45,10,4]. In the general case, exponential time in the dimension of $\mathcal{H}$ is needed [45], and the research typically focuses on special subclasses of functions, such as the *completely unimodal* (CU) functions. The dominating approach to pseudo-Boolean optimization is local search.

Recall that a *standard local search improvement algorithm* starts in an arbitrary point v_0 of the hypercube $\mathcal{H}$ and iteratively improves by selecting a next iterate with a better function value from a *polynomial* neighborhood of the current iterate. Specific instances of the standard algorithm are obtained when one fixes the neighborhood structure on $\mathcal{H}$, and the disciplines of selecting the initial point and the next iterate. Two major local search improvement algorithms, the *Greedy Single Switch Algorithm (GSSA)* and the *Random Single Switch Algorithm (RSSA)* have previously been extensively investigated and used for optimizing pseudo-Boolean function [45]. We consider those and also *multiple* switch algorithms in subsequent sections.

3 Hyperstructure Optimization

The set of all positional strategies of Player 0 in a parity game is isomorphic to a product of finite sets, each representing the player's choices in a vertex of V_0. The number of vertices of Player 0 corresponds to the dimension. Evaluating strategies in games motivates our study of functions on the *hyperstructures* defined below. They are generalizations of the Boolean hypercube $\{0,1\}^d$, corresponding to the case when the player's choice in every vertex is binary.

Definition 3.1 (Hyperstructure) *Let* $\mathcal{P}_j = \{e_{j,1}, \ldots, e_{j,\delta_j}\}$ *(for* $j \in \{1, \ldots,$

$d\}$) be a collection of finite nonempty disjoint sets. Call $\mathcal{P} = \prod_{j=1}^{d} \mathcal{P}_j$ a d-dimensional hyperstructure*, or* structure *for short.* □

(For the subsequent development it is useful to keep in mind that in this definition $e_{j,i}$ correspond to outgoing edges from vertex j, i.e., choices available to the player.)

A hyperstructure is a *hypercube* if $|\mathcal{P}_j| = 2$ for all j. A *substructure* of $\mathcal{P}$ is a product $\mathcal{P}' = \prod_{j=1}^{d} \mathcal{P}'_j$, where $\emptyset \neq \mathcal{P}'_j \subseteq \mathcal{P}_j$ for all j. In particular, $\mathcal{P}$ is a substructure of itself. A substructure for which $|\mathcal{P}'_j| \leq 2$ for all j is called a *subcube*. A *facet* of $\mathcal{P}$ is a substructure obtained by fixing the choice in exactly one coordinate. Thus $\mathcal{P}'$ is a facet of $\mathcal{P}$ if there is a $j \in \{1, \ldots, d\}$ such that $|\mathcal{P}'_j| = 1$ and $\mathcal{P}'_k = \mathcal{P}_k$ for all $k \neq j$. We will sometimes identify a facet with its defining element $e_{j,i} \in \mathcal{P}'_j$. We will consistently use the letter d for the dimension and $m := \sum_{j=1}^{d} |\mathcal{P}_j|$ for the number of facets. If $|\mathcal{P}_j| = 1$ for some j, this coordinate can be disregarded since it is constant, and we will consider such structures as having smaller dimension.

An element of $\mathcal{P}$ is called a *vertex* [3] or *corner*. Two vertices of $\mathcal{P}$ are *neighbors* if they differ in only one coordinate. The neighbor relation induces a graph with elements of $\mathcal{P}$ as nodes, and allows us to talk about paths and (Hamming) distances in $\mathcal{P}$. The subcube *spanned* by a pair of vertices is the smallest subcube containing the pair.

In terms of games and strategies, a *substructure* of the strategy hyperstructure $\mathcal{P}$ corresponds to the Player 0 strategies in a subgame where some edges leaving vertices in V_0 have been removed, without creating sinks. A *subcube* of $\mathcal{P}$ corresponds to the possible strategies in a *binary* subgame. If the choice in exactly one vertex from $v \in V_0$ is fixed, by removing all but one outgoing edge, the possible Player 0 strategies correspond to a *facet* of $\mathcal{P}$. The remaining outgoing edge from v is the defining element of this facet. Moving from a corner of $\mathcal{P}$ to a neighbor is the same as making a single switch in the underlying game.

For a set $E \subseteq \bigcup_{j=1}^{d} \mathcal{P}_j$, define $\mathit{struct}(E)$ to be the substructure $\prod_{j=1}^{d} (\mathcal{P}_j \cap E)$. Thus, if E does not have elements from each $\mathcal{P}_j$, then $\mathit{struct}(E) = \emptyset$. For a set $\mathcal{F}$ of facets, we use $\mathit{struct}(\mathcal{F})$ for $\mathit{struct}(E)$ where E is the set of defining elements of the facets in $\mathcal{F}$.

Proviso. Throughout this paper we consider functions $f : \mathcal{P} \to \mathcal{D}$ mapping a hyperstructure $\mathcal{P}$ to some *partially ordered set* $\mathcal{D}$. We always assume that any two neighbors on a hyperstructure have *comparable* f-values.

[3] Corresponds to a positional strategy, not to be confused with game vertices!

These assumptions are meaningful and non-restrictive for parity and simple stochastic games, as we will see in Sections 9 and 7.1.

A *local maximum* of a function $f : \mathcal{P} \to \mathcal{D}$ is a vertex v such that $f(v) \geq f(u)$ for all neighbors u of v. If $f(v) \geq f(u)$ for all vertices $u \in \mathcal{P}$, then v is a *global maximum* of f on $\mathcal{P}$. Local and global *minima* are defined symmetrically. If $\mathcal{P}'$ is a substructure of $\mathcal{P}$ then $f|_{\mathcal{P}'}$ denotes the restriction of f to $\mathcal{P}'$. Note that a vertex may be a local maximum of $f|_{\mathcal{P}'}$ without being a local maximum of f.

Given a function $f : \mathcal{P} \to \mathcal{D}$, and a substructure $\mathcal{P}'$ of $\mathcal{P}$, denote by $w_f(\mathcal{P}')$ the maximum value of f on $\mathcal{P}'$.

While optimizing functions on hyperstructures by local improvements, one needs to know which neighbors of the current vertex have better values. This is captured by the notion of *vector of improving directions* (VIDs).

Definition 3.2 (VID-function) *Let $f : \mathcal{P} \to \mathcal{D}$ be a function on $\mathcal{P} = \prod_{j=1}^{d} \mathcal{P}_j$ with comparable neighbors. Its corresponding* VID-function *VID_f is a function from $\mathcal{P}$ to $\prod_{j=1}^{d} 2^{\mathcal{P}_j}$, defined in the following way. Let $v \in \mathcal{P}$ and let u be the vertex reached from v by switching to $e_{j,i}$ in coordinate j. The value $VID_f(v)$ contains $e_{j,i}$ in coordinate j iff $f(v) < f(u)$.* □

Sometimes one just needs to know the number of better neighbors of a vertex in each coordinate. This information is given by *signatures*.

Definition 3.3 (Signature-function) *Let $f : \mathcal{P} \to \mathcal{D}$ be a function on $\mathcal{P} = \prod_{j=1}^{d} \mathcal{P}_j$ with comparable neighbors. Its corresponding* signature-function *SIG_f is a function from $\mathcal{P}$ to $\prod_{j=1}^{d} \{0, \ldots |\mathcal{P}_j| - 1\}$, defined in the following way. For a vertex $v \in \mathcal{P}$, if $VID_f(v) = (A_1, A_2, \ldots, A_d)$, then $SIG_f(v) = (|A_1|, |A_2|, \ldots, |A_d|)$. The value $SIG_f(v)$ is called the* signature *of v with respect to f.* □

Our interest in hyperstructure optimization is primarily motivated by the applications to solving games. It turns out that the assignments of values to strategies in parity games proposed by Vöge and Jurdziński [46], and our modification from [5] (developed in detail in Section 9), possess some very useful properties, allowing for a variety of optimization algorithms. The same holds true for the well-known strategy evaluation function for simple stochastic games. In the next section, we present three classes of functions, all interesting in themselves, and relevant in particular to the parity and simple stochastic games problems.

4 Classes of Functions

Arbitrary functions on $\mathcal{P}$ cannot be efficiently optimized. There are meaningful and non-degenerate classes of functions, like LG-functions (see Tovey [44,45] and below), which require provably exponential time for finding an optimum (see, e.g., Corollary 19 of [45]), but the prospects are better for some non-trivial and very interesting subclasses. The class of *completely unimodal* (CU) functions on hypercubes $\mathcal{H} = \{0,1\}^d$ has been studied in, e.g, [28,49,48,4], and can be optimized in subexponential time [4]. We generalize the definition of complete unimodality to hyperstructures. In [7] we defined the class of *completely local-global* (CLG) functions, a generalization of CU-functions, defined on hyperstructures and capturing the properties of value-functions obtained from parity and simple stochastic games. To demonstrate the full range of applicability of the algorithms in later sections, we also define an even wider class of *recursively local-global* (RLG) functions, the widest, to our knowledge, admitting subexponential optimization. The chain of inclusions

$$\mathrm{CU} \subset \mathrm{CLG} \subset \mathrm{RLG} \subset \mathrm{LG}$$

will follow from the definitions. The strictness of these inclusions is easy to establish.

4.1 Recursively Local-Global Functions

We begin with the definition of the most general class of functions we study.

Definition 4.1 (Recursively Local-Global) *A function $f : \mathcal{P} \to \mathcal{D}$ for which all neighbors have comparable values is called* recursively local-global *(RLG) if for every substructure $\mathcal{P}'$ of $\mathcal{P}$, all local maxima of $f|_{\mathcal{P}'}$ are also global.* □

In Section 10 we show that strategy evaluation functions for simple stochastic and parity games are RLG. The related larger class of LG (Local-Global) functions was introduced and studied by Aldous [2] and Tovey [44]. For an LG-function it is only required that every local maximum is also global, i.e., recursiveness of the LG-property on substructures is not stipulated, in contrast to RLG. Aldous settles tight exponential $\Theta(2^{d/2})$ lower and upper bounds for LG-function optimization. In contrast, we show that RLG-, CLG-, and CU-functions admit *subexponential*, $2^{O(\sqrt{d})}$ optimization. Tovey [44] proved that for natural distributions of LG-functions on hypercubes, standard variants of the greedy and random local searches are low-degree polynomial on the average.

Remark. It is important to note that Definition 4.1, together with the definitions of local and global maxima, imply that every vertex that is not a global maximum has at least one strictly better neighbor. It also follows that every restriction of an RLG-function to a substructure of $\mathcal{P}$ is RLG. □

In Section 5, we discuss algorithms for maximizing RLG-functions. Such algorithms can also be used to maximize CLG-functions and CU-functions. We employ *RLG-structure* to denote an RLG-function together with its underlying hyperstructure, and we use *RLG-cube* if this hyperstructure is a hypercube. An RLG-structure can be thought of as a hyperstructure with its vertices labeled by function values.

4.2 Completely Local-Global Functions

The class of Completely Local-Global (CLG) functions is a restriction of the RLG-functions. We demand that on every substructure, both local minima and maxima of the corresponding function restriction are global. Furthermore, all such optima should be connected by paths of optima.

Definition 4.2 (Completely Local-Global) *Let $f : \mathcal{P} \to \mathcal{D}$ be a function such that all neighbors have comparable values. Say that f is* completely local-global *(CLG) if the following properties hold for every substructure $\mathcal{P}'$ of $\mathcal{P}$.*

(1) Every local maximum of $f|_{\mathcal{P}'}$ is also global;
(2) every local minimum of $f|_{\mathcal{P}'}$ is also global;
(3) every two local maxima of $f|_{\mathcal{P}'}$ are connected by a path of local maxima on $\mathcal{P}'$;
(4) every two local minima of $f|_{\mathcal{P}'}$ are connected by a path of local minima on $\mathcal{P}'$. □

This is the class of functions most closely related to the strategy evaluation functions of parity and simple stochastic games. We show in Section 10 that such functions are in fact CLG.

It turns out that checking conditions of Definition 4.2 can be conducted locally, on every 2-dimensional subcube, which considerably simplifies many subsequent proofs. This property was first proved for CU-functions on hypercubes by Williamson Hoke [49].

Theorem 4.3 *Let $f : \mathcal{P} \to \mathcal{D}$ be a function with comparable neighbors. If the restriction of f to any 2-dimensional* subcube *satisfies conditions 1, 2, and either 3 or 4 of Definition 4.2, then f is CLG.*

Proof. We first note that on a 2-dimensional subcube, a function that satisfies conditions 1, 2, and either 3 or 4 satisfies all the conditions. In fact, a 2-face has two local maxima not connected by a path of local maxima, iff it has two local minima with no path of local minima between them.

Now we prove that the restriction of f to every *subcube* $\mathcal{H}$ of $\mathcal{P}$ is CLG by induction on the dimension $d_{\mathcal{H}}$ of $\mathcal{H}$. If $d_{\mathcal{H}} \leq 2$ the theorem is immediate from the definition. Assume it holds for all cubes of dimension $< d$ and let $d_{\mathcal{H}} = d$. Suppose first that $f|_{\mathcal{H}}$ has either: 1) a local maximum that is not global, or 2) two local maxima not connected by a path. In the first case, the local maximum a is global on every $(d-1)$-dimensional face of $\mathcal{H}$ it belongs to by induction hypothesis. It follows that it has value greater or equal to all vertices on $\mathcal{H}$ except the one on the long diagonal (with Hamming distance d), which we denote by b. Note that $f(b)$ is strictly greater than the values of all neighbors of b: since they all have values smaller than or equal to $f(a)$, and the order on $\mathcal{D}$ is transitive, a would otherwise be a global maximum of $f|_{\mathcal{H}}$.

In the case of two local maxima not connected by a path, the situation is identical: two vertices a and b with Hamming distance n.

In both cases, consider any neighbor c of b on $\mathcal{H}$ and the $(d-1)$-face G of $\mathcal{H}$ spanned by c and a.

Suppose there is a local maximum $e \neq a$ of $f|_G$. This vertex must have the same value as a, and there would be a path of vertices with equal values from e to a on G. Consider the vertex closest to a on this path. It cannot be a local maximum on $\mathcal{H}$, since then there would, by inductive assumption also be a path of local maxima from b to it. Thus its neighbor on $\mathcal{H} \setminus G$ has a bigger value. This, however, contradicts the fact that a is a global maximum of the restriction of f to any 2-dimensional subcube of $\mathcal{H}$ it belongs to. Thus a is the only local maximum of $f|_G$.

Since a is the only local maximum of $f|_G$, there must be a strictly increasing path from c to a on $\mathcal{H}$. Since c was arbitrary, this holds for all neighbors of b on $\mathcal{H}$. Symmetrically, there must be an increasing path from every neighbor of a to b. This is a contradiction and we conclude that the restriction of f to any subcube of $\mathcal{P}$ is CLG.

Now consider any substructure $\mathcal{P}'$ of $\mathcal{P}$, and any two local maxima a and b of $f|_{\mathcal{P}'}$. There is a unique minimal hypercube $\mathcal{H}$ that contains a and b and is a substructure of $\mathcal{P}'$. Since $f|_{\mathcal{H}}$ is CLG, a and b have the same value and are connected by a path of local maxima.

The reasoning for local minima is symmetrical. □

The next theorem was originally proved in [48] for CU-functions on hypercubes. We extend it to CLG-functions. The theorem shows that CLG-functions satisfy the Hirsch conjecture.

Theorem 4.4 *Let $f : \mathcal{P} \to \mathcal{D}$ be a CLG-function, $v \in \mathcal{P}$, and d the dimension of $\mathcal{P}$. Then there is a path of vertices with strictly increasing (decreasing) f-values from v to a global maximum (minimum) of length at most d.*

Proof. We will prove a slightly stronger claim, namely that there is a path to the *nearest* maximum v^*, in the sense of the Hamming distance $h = hd(v, v^*)$, and that its length is exactly h.

If v is not a global maximum, then it has a neighbor u on the subcube spanned by v and v^* with a better function value. If it had not, it would be a local maximum on this subcube, and thus have the same value as v^*. Thus for a non-maximal vertex v, at least one of its better neighbors u has shorter Hamming distance to the nearest maximum v^*. The theorem is now proved by induction on the Hamming distance from v to v^*. Suppose it holds for all shorter distances. Then there is a path from u to v^* with length $hd(u, v^*)$. Since the step from v to u is improving there is an improving path from v to v^* with length h. The proof for minima is symmetrical. □

The argument above works for RLG-functions as well. We thus have the following

Corollary 4.5 *Let $f : \mathcal{P} \to \mathcal{D}$ be an RLG-function, $v \in \mathcal{P}$, and d the dimension of $\mathcal{P}$. Then there is a path of vertices with strictly increasing f-values from v to a global maximum of length at most d.*

4.3 Completely Unimodal Functions

Completely unimodal functions on hypercubes have been extensively studied [28,49,48,4] (also referred to as abstract objective functions (AOF) [23,24]). Here we generalize the definition to include functions on hyperstructures, and discuss appropriate optimization algorithms in Sections 5 and 6. Generalized CU-functions are interesting and representative: we show that every CLG-function can be efficiently reduced to a CU-function [7] (see Theorem 4.10), and thus many properties of the better studied and understood CU-functions can be transferred to the CLG-functions.

Definition 4.6 (Completely Unimodal) *A function $f : \mathcal{P} \to \mathcal{D}$ for which all neighbors have comparable values is called* completely unimodal *(CU) if for every substructure $\mathcal{P}'$ of $\mathcal{P}$, $f|_{\mathcal{P}'}$ has a* unique *local maximum and a* unique *local minimum.* □

This definition implies that no pair of neighbors can have the same value, since this would mean that the 1-dimensional substructure the two vertices define has two local maxima.

A remarkable theorem by Williamson Hoke [49] states that any CU-function on a hypercube has an injective signature-function. We here extend this theorem to CU-functions on any hyperstructures. For hypercubes, the converse is known to hold: if the signature-function of f is injective, then f is CU.

Theorem 4.7 (Injectivity of Signature Function) *Let $f : \mathcal{P} \to \mathcal{D}$ be a CU-function. Then SIG_f is injective.*

Proof. Assume the contrary and let $x, y \in \mathcal{P}$ be two vertices with identical signatures. We may assume that the Hamming distance between x and y is d; otherwise, fix all coordinates where x and y coincide and consider the corresponding substructure. By renaming coordinate values, assume without loss of generality that $x = (0, \ldots, 0)$ and $y = (1, \ldots, 1)$. Let $(X_1, \ldots, X_d) = VID_f(x)$ and $(Y_1, \ldots, Y_d) = VID_f(y)$. Thus by assumption $|X_j| = |Y_j|$ for all j.

It is feasible that $X_j = Y_j$ for some j and $X_j \neq Y_j$ for other j. By reordering coordinates, we may without loss of generality assume that $X_j \neq Y_j$ in the first k coordinates and $X_j = Y_j$ in the remaining $d - k$ coordinates, where $k \in \{0, \ldots, d\}$. For all $j \in \{1, \ldots, k\}$, take $x_j \in X_j$ such that $x_j \notin Y_j$ and take $y_j \in Y_j$ such that $y_j \notin X_j$. Both x_j and y_j exist because $|X_j| = |Y_j|$ and $X_j \neq Y_j$. Note that neither 1 nor 0 are elements in X_j or Y_j for $j > k$, since $0 \notin X_j, 1 \notin Y_j$ and $X_j = Y_j$.

Now consider the vertex $z = (y_1, \ldots, y_k, 1, \ldots, 1)$. Note that $f(z) \leq f(x)$ because x is a maximum in the hypercube spanned by x and z (all directions are non-improving). Moreover, $f(y) \leq f(z)$ since y is a minimum in the cube spanned by y and z (all directions are improving). The equalities $f(z) = f(x)$ and $f(z) = f(y)$ hold only in the degenerate cases $z = x$ and $z = y$, respectively. Thus, since $x \neq y$, we have $f(y) < f(x)$. With a similar argument we show that the vertex $w = (x_1, \ldots, x_k, 0, \ldots, 0)$ satisfies $f(y) \geq f(w) \geq f(x)$ and $f(y) > f(x)$. This contradiction finishes the proof. □

Williamson Hoke also proved that any function on a hypercube is CU if its restriction to any 2-dimensional subcube is CU [49]. This theorem extends to any hyperstructure.

Theorem 4.8 *Let $f : \mathcal{P} \to \mathcal{D}$ be a function such that all neighbors have comparable and different values. The following statements are equivalent.*

(1) For every 2-dimensional subcube $\mathcal{H}$ of $\mathcal{P}$, $f|_{\mathcal{H}}$ has a unique local minimum.

(2) For every 2-dimensional subcube $\mathcal{H}$ of $\mathcal{P}$, $f|_{\mathcal{H}}$ has a unique local maximum.
(3) For every substructure $\mathcal{P}'$ of $\mathcal{P}$, $f|_{\mathcal{P}'}$ has a unique local minimum.
(4) For every substructure $\mathcal{P}'$ of $\mathcal{P}$, $f|_{\mathcal{P}'}$ has a unique local maximum.

Proof. The equivalence of (1) and (2) follows from the theorem on hypercubes from [49], and the implications from (3) to (1) and from (4) to (2) are immediate. Now we assume (1) and prove (3). Let x be a local minimum of f on some substructure $\mathcal{P}'$ of $\mathcal{P}$. By the theorem on hypercubes, x is a global minimum on every subcube of $\mathcal{P}'$ containing x. Since every other vertex in $\mathcal{P}'$ shares some subcube with x, it follows that x is a global minimum on $\mathcal{P}'$. A similar argument shows that (2) implies (4). This finishes the proof. □

We now show that optimizing a CLG-function can be efficiently reduced to optimizing a CU-function on the same hyperstructure. By definition, every CU-function is also CLG. Conversely, it is possible to turn every CLG-function into a CU-function by inducing an artificial order on neighbors with equal values, as follows.

Definition 4.9 *Let $\mathcal{P} = \prod_{j-1}^{d} \mathcal{P}_j$ and introduce a linear order on every $\mathcal{P}_j$ so that $e_{ji} < e_{jk}$ whenever $i < k$. Let $f : \mathcal{P} \to \mathcal{D}$ be a CLG-function. Define a partial order on $\mathcal{D}_{CU} = \mathcal{D} \times \mathcal{P}$ as follows. For any $(\alpha, x), (\alpha', x') \in \mathcal{D}_{CU}$, say that $(\alpha, x) < (\alpha', x')$ if:*

(1) $\alpha < \alpha'$, or
(2) $\alpha = \alpha'$ and $x < x'$ (lexicographic comparison). □

It is easily verified that this is a partial order.

Theorem 4.10 (Reducing CLG-functions to CU-functions) *If $f : \mathcal{P} \to \mathcal{D}$ is a CLG-function then $f_{CU} : \mathcal{P} \to \mathcal{D}_{CU}$, defined as $f_{CU}(x) = (f(x), x)$, is completely unimodal. If $x \in \mathcal{P}$ is the maximum (minimum) of f_{CU} then x is a maximum (minimum) of f.*

Proof. Clearly any two neighbors have comparable and different f_{CU}-values. By Theorem 4.8, it suffices to prove that f_{CU} restricted to any two-dimensional subcube has a unique local maximum. Consider any two-dimensional subcube F and assume $f_{CU}|_F$ has more than one local maximum. These cannot be neighbors, since neighbors have comparable and different values. Thus they are on the diagonal. Maxima of $f_{CU}|_F$ must be maxima of $f|_F$, since $f_{CU}(x) \geq f_{CU}(y) \implies f(x) \geq f(y)$ for any neighbors x, y. By Definition 4.2, all values of $f|_F$ are equal, so Case 2 of Definition 4.9 applies whenever the value of two neighbors x, y on F are compared. Hence the coordinate where x and y are different determines the order, and it follows that any local maximum has both coordinates set to their maximum on F. But there is only one such

corner, contradicting our assumption that there are more than one maxima. This completes the proof. □

Note that the reduction is efficient: f_{CU} can be computed in a single vertex of $\mathcal{P}$ using only one call to f.

5 Subexponential Recursive Algorithms

In [5] we showed how Kalai's algorithm for linear programming [35] can be adapted to solving parity games in expected subexponential time. In this section we show in the more general setting of RLG-function optimization, that several different algorithms building on the same ideas as Kalai's algorithm, are applicable. We begin with the conceptually easier case of hypercubes, and show how to adapt Ludwig's algorithm for simple stochastic games [36] to optimizing RLG-functions on $\mathcal{H}$. We then continue with adaptations of the linear programming algorithms by Matoušek, Sharir, and Welzl [38,39] and Kalai [35] to RLG-optimization on the general hyperstructures. We also show that RLG-optimization can be reduced to both the LP-type problems defined by Matoušek, Sharir, and Welzl [38], and to the abstract optimization problems of Gärtner [22].

5.1 Ludwig-Style Algorithm for Hypercubes

The first subexponential randomized algorithm for solving simple stochastic games was presented in [36]. It applies the ideas of the linear programming algorithms in [38,39] and [35], which we will discuss in later sections. Unfortunately, the algorithm only applies to binary games (with vertex outdegree at most two), and reduction to such games may increase the number of vertices, undermining the subexponential analysis.

Nevertheless, as an introduction to the ideas underlying the algorithms presented later, and their adaptations to RLG-functions, we first describe the Ludwig-style algorithm (presented as Algorithm 1 below; see also [24]) maximizing any RLG-function $f : \mathcal{H} \to \mathcal{D}$, on a hypercube $\mathcal{H} = \{0, 1\}^d$.

The key observation in the analysis of this algorithm is that, depending on the choice in line (3), the actual ("hidden") dimension of the remaining problem solved on line 6 may decrease with more than one. There are d equally likely choices; call them $F_1, F_2, \ldots, F_d$. Assume the set $\{w_f(F_1), \ldots, w_f(F_d)\}$ is linearly ordered so that $w_f(F_1) \leq \cdots \leq w_f(F_d)$.

If F_i is chosen for the first recursive call, none of the facets $F_1, \ldots, F_i$ will ever

Algorithm 1: Ludwig-Style Optimization Algorithm
LUDWIG(RLG-cube $\mathcal{H}$, initial vertex $v_0 \in \mathcal{H}$)
(1) **if** $\dim(\mathcal{H}) = 0$
(2) **return** v_0
(3) choose a random facet F of $\mathcal{H}$ containing v_0
(4) $v^* \leftarrow$ Ludwig(F, v_0)
(5) **if** neighbor u of v^* on $\mathcal{H} \setminus F$ is better than v^*
(6) **return** Ludwig($\mathcal{H} \setminus F$, u)
(7) **else**
(8) **return** v^*

be visited by the algorithm again, since the value of the current vertex v^* is the biggest value on any of them. Let $T(d)$ be the expected number of times that the second recursive call will be made. Then $T(0) = 0$ and

$$T(d) \leq T(d-1) + 1 + \frac{1}{d} \sum_{j=0}^{d-1} T(j).$$

Solving this recurrence [36,22] gives $T(d) = 2^{O(\sqrt{d})}$. What if the set $\{w_f(F_1), \ldots, w_f(F_d)\}$ is not linearly ordered? This is no disadvantage to the algorithm, since it only visits vertices with *comparable and strictly bigger* values than the current vertex. Therefore, linearly ordered maximal values on the facets is the worst case.

5.2 Matoušek–Sharir–Welzl-Style Algorithm

In this section, we show how the LP-algorithm by Matoušek, Sharir, and Welzl [38,39] can be modified for RLG-functions, yielding a simple subexponential randomized algorithm for the problem (and thus eventually for parity, mean payoff and simple stochastic games; see [7] and Sections 7-9). It has expected running time at most $2^{O(\sqrt{d \log(m/\sqrt{d})} + \log m)}$, where $m = \sum_{j=1}^{d} |\mathcal{P}_j|$ is the number of facets of $\mathcal{P}$. Note that this algorithm generalizes Ludwig's algorithm, discussed in Section 5.1. If $\mathcal{P}$ is a hypercube, the two algorithms are the same.

Algorithm 2 always terminates since the recursive calls in lines (4) and (6) are made on strictly smaller substructures. It is correct since any vertex without better neighbors in an RLG-structure is globally optimal.

It remains to show that the bound from [39] holds in this setting. This is done by translating into the concepts of [39], thus showing that the same recurrence for the running time holds. The following definition will be useful.

Definition 5.1 *A facet of an RLG-structure is* extreme *if it contains all local*

Algorithm 2: MSW-Style Optimization Algorithm
MSW(RLG-structure $\mathcal{P}$, initial vertex v_0)
(1) **if** $\dim(\mathcal{P}) = 0$
(2) **return** v_0
(3) choose a random facet F of $\mathcal{P}$, not containing v_0
(4) $v^* \leftarrow$ MSW$(\mathcal{P} \setminus F, v_0)$
(5) **if** neighbor u of v^* on F is better than v^*
(6) **return** MSW(F, u)
(7) **else**
(8) **return** v^*

maxima on $\mathcal{P}$. □

At most one facet in each coordinate can contain all maxima, and thus there are at most d extreme facets. The second recursive call on line (6) of the algorithm will be performed only when the chosen facet F is extreme on $\mathcal{P}$. This means that the second call will be made with probability at most $d/(m-d)$. If the current vertex belongs to $d-k$ extreme facets, this probability is at most $k/(m-d)$, hence, at most $\min(k, m-d)/(m-d)$.

Definition 5.2 (Hidden Dimension.) *Given an RLG-structure* $\mathcal{P}$ *and a vertex* $v \in \mathcal{P}$*, the* hidden dimension *of the pair* $(\mathcal{P}, v)$ *is* d *minus the number of facets of* $\mathcal{P}$ *containing* v *and every* $u \in \mathcal{P}$ *with* $f(u) > f(v)$. □

Intuitively, if $(\mathcal{P}, v)$ has hidden dimension k, then v and all vertices with better values than v belong to some k-dimensional substructure $\mathcal{P}'$ of $\mathcal{P}$. In particular, if the hidden dimension is 0, then v is a maximum on $\mathcal{P}$. The algorithm will only visit vertices that belong to $\mathcal{P}'$. There are $d-k$ facets, $F_1, F_2, \ldots, F_{d-k}$, that contain $\mathcal{P}'$; these are extreme. In the worst case there are k more extreme facets. For each such facet, consider the best value that does not belong to it. Assume first that these values are totally ordered. Enumerate the facets so that

$$w_f(\mathcal{P} \setminus F_{d-k+1}) \leq \cdots \leq w_f(\mathcal{P} \setminus F_{d-k+i}) \leq \cdots \leq w_f(\mathcal{P} \setminus F_d).$$

Suppose the algorithm chooses facet F_{d-k+i}. Then $f(v^*) = w_f(\mathcal{P} \setminus F_{d-k+i}) < f(u)$. Every vertex with a better value than v^* that belongs to F_{d-k+i} must also belong to F_{d-k+j} for all $1 \leq j < i$. Thus u belongs to $(d-k+i)$ facets that contain all vertices with better values, and the hidden dimension of the pair (F_{d-k+i}, u) for the second call is at most $(k-i)$.

If the best values outside the remaining extreme facets are not totally ordered, this only benefits the algorithm. The values are partially ordered, and if facet F_{d-k+i} is chosen, u will belong to all facets that do not have a strictly bigger

value for $w_f(\mathcal{P} \setminus F_j)$, and they will all contain all vertices with better values. This is because if $w_f(\mathcal{P} \setminus F_{d-k+i})$ and $w_f(\mathcal{P} \setminus F_j)$ are incomparable, then any value better than $w_f(\mathcal{P} \setminus F_{d-k+i})$ will be better than or incomparable to $w_f(\mathcal{P} \setminus F_j)$, since the order on $\mathcal{D}$ is transitive.

This discussion is enough to get the same recurrences for the numbers of tests on line (5) and jumps to u on line (6) as [39] gets for the numbers of violation tests and basis computations, respectively, in the linear programming setting. They are both bounded by the recurrence $t_k(d) = 0$ (where $0 \leq k \leq d$) and

$$t_k(m) \leq t_k(m-1) + 1 + \frac{1}{m-d} \sum_{i=1}^{\min(k,m-d)} t_{k-i}(m), \qquad \text{for } m > d.$$

This recurrence is solved in [39], and from the solution it is easy to infer that the expected running time of Algorithm 2 on RLG-structures is at most $2^{O(\sqrt{d \log(m/\sqrt{d})} + \log m)}$, as long as function evaluation and comparison of function values can be performed in polynomial time. As soon as m can be assumed to be polynomially bounded by d, this bound is subexponential in d. The reduction from parity and simple stochastic games with d vertices to RLG-functions gives $m = O(d^2)$, so we can solve these games in expected time $2^{O(\sqrt{d \log d})}$.

5.3 Kalai-Style Algorithm

We now describe another algorithm for optimizing RLG-structures, originally invented for linear programming by Kalai [35,25] and later adapted by us to parity games [5] (it can also be adjusted for mean payoff and simple stochastic games). It is also subexponential with complexity similar to the MSW-style algorithm of the previous section (but it is slightly more complicated).

Let $\mathcal{P}(d, m)$ denote the class of RLG-structures with dimension d and $m = \sum_{j=1}^{d} |\mathcal{P}_j|$. If v is a vertex in $\mathcal{P}$ then a facet F is *v-improving* if some *witness vertex* $v' \in F$ has a better value than v.

5.3.1 The Algorithm

Algorithm 3 takes an RLG-structure $\mathcal{P} \in \mathcal{P}(d, m)$ and an initial vertex v_0 as inputs and returns an optimal vertex on $\mathcal{P}$. It uses the subroutine Collect-Improving-Facets, described later, that collects a set of pairs (F, v) of v_0-improving facets F and corresponding witness vertices v.

Algorithm 3: Kalai-Style Optimization Algorithm
KALAI(RLG-structure $\mathcal{P}$, initial vertex v_0)
(1) **if** $\dim(\mathcal{P}) = 0$
(2) **return** v_0
(3) $M \leftarrow$ COLLECT-IMPROVING-FACETS($\mathcal{P}, v_0$)
(4) choose a random pair $(F, v_1) \in M$
(5) $v^* \leftarrow$ KALAI(F, v_1)
(6) **if** some neighbor u of v^* on $\mathcal{P} \setminus F$ is better than v^*
(7) **return** KALAI($\mathcal{P}, u$)
(8) **else**
(9) **return** v^*

5.3.2 How to Find Many Improving Facets

Now we describe the subroutine COLLECT-IMPROVING-FACETS. The goal is to find r facets that are v_0-improving, where r is a parameter (Kalai uses $r = \max(d, m/2)$ to get the best complexity analysis). To this end we construct a sequence $(\mathcal{P}^0, \mathcal{P}^1, \ldots, \mathcal{P}^{r-d})$ of substructures of $\mathcal{P}$, with $\mathcal{P}^i \in \mathcal{P}(d, d+i)$ and $\mathcal{P}^i \subset \mathcal{P}^{i+1}$. All the $d+i$ facets of $\mathcal{P}^i$ are v_0-improving; we simultaneously determine the corresponding witness vertices v^j optimal in $\mathcal{P}^j$. The subroutine returns r facets of $\mathcal{P}$, each one obtained by fixing one of the r choices in $\mathcal{P}^{r-d} \in \mathcal{P}(d, r)$. All these are v_0-improving by construction.

Let v^0 be a better neighbor of v_0 on $\mathcal{P}$. (If no better neighbor exists then v_0 is optimal in $\mathcal{P}$ and we are done.) Set $\mathcal{P}^0$ to the RLG-structure containing only v^0. Fixing any of the d coordinates of $\mathcal{P}$ as in v^0 defines a v_0-improving facet of $\mathcal{P}$ with v^0 as a witness.

To construct $\mathcal{P}^{i+1}$ from $\mathcal{P}^i$, let v' be a better neighbor of v^i. (Note that v^i is optimal in $\mathcal{P}^i$ but not necessarily in the full structure $\mathcal{P}$. If it is, we terminate.) Let $\mathcal{P}^{i+1}$ be the smallest substructure of $\mathcal{P}$ containing v' and all vertices in $\mathcal{P}^i$. Recursively apply the algorithm to find the optimal v^{i+1} in $\mathcal{P}^{i+1}$. Note that fixing a coordinate in $\mathcal{P}$ to any of the $d+i$ choices in $\mathcal{P}^i$ defines a v_0-improving facet. Therefore, the final $\mathcal{P}^{r-d}$ has r facets that are v_0-improving.

5.3.3 Analysis

First note that we can pick the random number before line (3), pass it as a parameter to COLLECT-IMPROVING-FACETS, and only find that many facets. Now each solved subproblem starts from a strictly better vertex, and it is clear that the algorithm terminates. It is correct because it can only terminate by returning an optimal vertex.

This algorithm yields a recurrence similar to those in previous sections. Kalai solves it and gets the subexponential running time $2^{O((\log m)\sqrt{d/\log d})}$.

It is interesting to note the similarities of Ludwig's, Matoušek–Sharir–Welzl's, and Kalai's subexponential algorithms. They all take a structure $\mathcal{P}$ and a vertex v_0 as parameters and recursively find the optimum v^* on a substructure $\mathcal{P}'$ of $\mathcal{P}$. The choice of $\mathcal{P}'$ is random, but it is guaranteed to contain vertices at least as good as v_0, and the recursive call starts from a witness of this fact. If the result is not an optimum on the entire structure, then we optimize the rest of the structure, starting from a better neighbor of v^*. Intuitively, the subexponential upper bound relies on the substructure being taken randomly from some set. Thus it is expected that the optima of many structures in the set are no better than v^*, and those structures will be ignored in the remainder of the algorithm.

5.4 LP-Type Problems

Local improvement algorithms are well-studied since the 1950's in the context of the simplex method for linear programming. The subexponential algorithms we present in this paper and in [5] are adaptations of algorithms originally suggested for linear programming [35,38,39]. The papers [43,38] present an abstract framework called *LP-type problems*, appropriate for several problems, including linear programming, MiniBall (given a set of points in d-space, determine the center and radius of the smallest enclosing ball), PolyDist (given two convex polyhedra in d-space, determine the distance between them) [22], and others [38]. We now show how optimizing an RLG-structure can be reduced to solving an LP-type problem. Together with the results of Section 10, this provides an interesting link between game theory and computational geometry, and allows us to transfer any possible new algorithms for LP-type problem to RLG-optimization and games. We start by briefly defining the abstract framework [43]; see [43,38] for details.

Definition 5.3 (LP-type problem) *Let H be a finite set, $\mathcal{W}$ a linearly ordered set, and $g : 2^H \to \mathcal{W}$ a function. The pair (H, g) is called an* LP-type problem *if it satisfies the following properties.*

$$F \subseteq G \subseteq H \implies g(F) \leq g(G) \tag{1}$$

$$\left.\begin{array}{c} F \subseteq G \subseteq H \\ g(F) = g(G) \\ h \in H \end{array}\right\} \implies \left\{\begin{array}{c} g(F \cup h) > g(F) \\ \iff \\ g(G \cup h) > g(G) \end{array}\right. \tag{2}$$

Intuitively, each element of H corresponds to an element of some $\mathcal{P}_i$, and the value of a subset of H will be the value of the corresponding substructure.

A *basis* is a subset B of H with $g(B') < g(B)$ for all $B' \subset B$; it corresponds to an element of the hyperstructure. A *basis for* $G \subseteq H$ is a basis $B \subseteq G$ with $g(B) = g(G)$; it corresponds to a vertex possessing the maximum value on the substructure corresponding to G. To "solve" an LP-type problem means to find a basis of H. Note that "basis" is not the same as "basis for H".

The reduction is described as follows.

Theorem 5.4 *An RLG-function $f : \mathcal{P} \to \mathcal{D}$, where $\mathcal{D}$ is linearly ordered, can be expressed as an LP-type problem (H, g) such that a basis for H defines a maximum of the RLG-function.*

Proof. Recall that $\mathcal{P} = \prod_{j=1}^{d} \mathcal{P}_j$ and let H be the disjoint union of all $\mathcal{P}_j$. As defined in Section 3, let $\mathit{struct}(G) = \prod_{j=1}^{d}(\mathcal{P}_j \cap G)$, for $G \subseteq H$. Recall that $\mathit{struct}(G) = \emptyset$ whenever $P_j \cap G = \emptyset$ for some j. The value set of our LP-type problem will be $\mathcal{W} = \mathcal{D} \cup \mathbb{N}$, with the usual orders on $\mathcal{D}$ and on $\mathbb{N}$, and all elements of $\mathbb{N}$ smaller than those of $\mathcal{D}$. Now define $g : 2^H \to \mathcal{W}$ as

$$g(G) = \begin{cases} |G| \in \mathbb{N} & \text{if } \mathit{struct}(G) = \emptyset; \\ w_f(\mathit{struct}(G)) & \text{otherwise.} \end{cases}$$

In other words, if G contains at least one element from each $\mathcal{P}_j$ then $g(G)$ is the maximum of all elements in the associated substructure; otherwise $g(G) = |G|$.

Property (1) of Definition 5.3 is immediate: if $F \subseteq G$ and $g(F) \in \mathcal{D}$ then $\mathit{struct}(F)$ is a substructure of $\mathit{struct}(G)$, so the maximum in $\mathit{struct}(F)$ is no bigger than the maximum in $\mathit{struct}(G)$. If $g(F) \in \mathbb{N}$ then $g(G)$ is either a bigger natural number or an element of $\mathcal{D}$. Proving (2) needs some more work.

Let $F \subseteq G \subseteq H$ be such that $g(F) = g(G)$ and take $h \in H$. We need to prove that $g(F \cup h) > g(F) \iff g(G \cup h) > g(G)$. If $g(F) \in \mathbb{N}$, then $F = G$, since otherwise $g(G)$ would be greater than $g(F)$, and the equivalence follows. Assume $g(F) \in \mathcal{D}$. The left-to-right implication is clear: if $g(F) \in \mathcal{D}$ then $\mathit{struct}(F \cup h)$ is a substructure of $\mathit{struct}(G \cup h)$. Therefore $g(F \cup h) \leq g(G \cup h)$, and we get $g(G \cup h) \geq g(F \cup h) > g(F) = g(G)$.

Now suppose $g(G \cup h) > g(G)$. It remains to show that $g(F \cup h) > g(F)$. Assume the contrary and let $x \in \mathit{struct}(F)$ be a local maximum on $\mathit{struct}(F)$. Note that $g(G) = g(F) = g(F \cup h)$ by assumptions ($g(F)$ cannot be greater than or incomparable to $g(F \cup h)$ by Property (1) of Definition 5.3). Since any neighbor of x on $\mathit{struct}(G \cup h)$ is an element of either $\mathit{struct}(F \cup h)$ or $\mathit{struct}(G)$, they all have smaller or equal values, so x is a local maximum on $\mathit{struct}(G \cup h)$. This contradicts the assumption that $\mathit{struct}(G \cup h)$ has a greater local maximum, so (H, g) is an LP-type problem.

We now show that that an element $x \in \mathcal{P}$ is a local maximum of f if and only

if it is also a basis for H. If x is a maximum of f, then $g(x) = g(H)$. Also, since x has exactly one component for every $\mathcal{P}_j$, any subset of x has a value in $\mathbb{N}$. Thus x is a basis for H.

If, on the other hand x is a basis for H then it must have exactly one component from each $\mathcal{P}_j$: if $h \cap \mathcal{P}_j = \emptyset$ for some j, then x would not have the same value as H. Otherwise, if $|h \cap \mathcal{P}_j| > 1$ for some j, then there would be a subset of x with the same value. Thus x corresponds to an element of $\mathcal{P}$. Since $g(x) = g(H)$ it must be a global maximum of f on $\mathcal{P}$. □

Although the theorem requires a linear order, which is not guaranteed by the RLG definition, this is not a major drawback. First, many algorithms solving LP-type problems do not rely on the order being total. Second, the order on strategy values for simple stochastic games is total, and the one for parity games can easily be made total. Rather than comparing values of strategies componentwise in the tuple of vertex values, comparing them lexicographically extends the order to a total one.

5.5 Abstract Optimization Problems

The class of *abstract optimization problems* (AOPs) was introduced by Gärtner [22]. He uses this generalization to show how the ideas from [39,35] can be used to obtain a subexponential randomized algorithm for a wider class of optimization problems. In this section we show that optimizing an RLG-function with totally ordered codomain can be reduced to solving an AOP.

Definition 5.5 (AOP [22]) *An AOP is a triple $(A, <, \Phi)$, where A is a finite set, $<$ is a total order on 2^A, and $\Phi : \{(C, B) | C \subseteq B \subseteq A\} \to 2^A$ satisfies*

$$\Phi(C, B) = \begin{cases} C, & \text{iff } C = \max_<\{C' | C' \subseteq B\}; \\ C', \text{ for some } C' \text{ s.t. } C < C' \subseteq B, & \text{otherwise.} \end{cases}$$

For $B \subseteq A$, let $opt(B)$ denote $\max_<\{C | C \subseteq B\}$. Solving an AOP means finding $opt(A)$. □

Intuitively, solving an AOP corresponds to maximizing some function on a boolean hypercube. The function may be unstructured, but there is an oracle capable of finding some better vertex than the current one on any subcube containing the origin. The algorithm in [22] solves any AOP with $|A| = m$ using expected $e^{2\sqrt{m}+O(\sqrt[4]{m}\ln m)}$ calls to Φ. Maximizing an RLG-function f with m facets can be reduced to solving an AOP with $|A| = m$, if its codomain is totally ordered. This restriction can often be satisfied, as for, e.g., parity and simple stochastic games.

Let $\mathcal{P}$ be a d-dimensional hyperstructure with m facets, and let $f : \mathcal{P} \to \mathcal{D}$ be an RLG-function with totally ordered codomain. Define an AOP $(\mathcal{F}, <, \Phi)$ as follows. Let $-\infty$ be an artificial value, smaller than all values of vertices of $\mathcal{P}$. Let $\mathcal{F}$ be the set of all facets of p, and define $val : 2^{\mathcal{F}} \to \mathcal{D} \cup \{-\infty\}$ by

$$val(F) = \begin{cases} w_f(struct(F)), \text{ if } struct(F) \neq \emptyset; \\ -\infty, \qquad\qquad \text{otherwise.} \end{cases}$$

Take $\prec$ to be any total order on $2^{\mathcal{F}}$ such that $F \prec F'$ whenever $|F| > |F'|$. For $F, F' \in \mathcal{F}$ let $F < F'$ iff 1) $val(F) < val(F')$, or 2) $val(F) = val(F')$ and $F \prec F'$. Now define $\Phi(F, G)$ as follows.

- Suppose $val(F) \neq -\infty$ and $struct(F)$ contains only one vertex v. If v is a local maximum on $struct(G)$, then $\Phi(F, G) = F$. Otherwise $\Phi(F, G)$ is a set defining a 0-dimensional substructure containing only a better neighbor of v on $struct(G)$.
- If $val(F) \neq -\infty$ but $|F| > d$ then $\Phi(F, G) = F'$ where F' defines a substructure containing only one vertex with maximal value on $struct(F)$.
- If $val(F) = -\infty$ and $struct(G) = \emptyset$ then $\Phi(F, G) = \emptyset$.
- If $val(F) = -\infty$ and $struct(G) \neq \emptyset$ then $\Phi(F, G) = F'$ for some $F' \in struct(G)$.

Now $(\mathcal{F}, <, \Phi)$ defines an AOP, and applying Gärtner's algorithm will yield a solution from which a maximal vertex of the original RLG-structure can be recovered.

Unfortunately, the bound this gives for maximizing RLG-functions is not very good. The previously discussed algorithms give bounds subexponential in d as long as n is polynomial in d, which is not the case here. If for example, $n = \Theta(d^2)$, which is the worst case when reducing games to RLG-functions, we only get a $2^{O(d)}$ bound on the expected calls to Φ. The reason seems to be that the original dimension is lost in the reduction, and all facets are treated as independent.

Gärtner's algorithm, called with the appropriate parameters, will never call Φ with any set of size bigger than d as the first parameter. For such sets, Φ can be computed in polynomial time.

6 Local Search Algorithms

A *standard local-search improvement algorithm* starts in an arbitrary point v_0 of the hyperstructure $\mathcal{P}$ and iteratively improves by selecting a next iterate

with a better value from a *polynomial* neighborhood $N(v_i)$ of the current iterate. Specific instances of the standard algorithm are obtained by fixing (1) the neighborhood structure on $\mathcal{P}$, and (2) the disciplines of selecting the initial point and the next iterate.

Two major local-search improvement algorithms, the *Greedy Single Switch Algorithm (GSSA)* and the *Random Single Switch Algorithm (RSSA)* have previously been investigated and used for optimizing CU-functions on hypercubes [45]. We also describe and study the *All Profitable Switches Algorithm (APSA)* and the *Random Multiple Switches Algorithm (RMSA)*. Strictly speaking, neither APSA, nor RMSA is a local-search algorithm. The first one operates with neighborhood structures which vary depending on the function being optimized. The second chooses the next iterate from a non-polynomially bounded (in general) neighborhood of the current iterate.

6.1 Single Switch Algorithms

A single switch algorithm in each iteration proceeds to a better neighbor of the current iterate, by changing one coordinate at a time, until a local maximum is found. For RLG-functions (and thus also CLG- and CU-functions) local maxima are global, so any single switch algorithm correctly optimizes them.

6.1.1 Greedy Single Switch Algorithm (GSSA)

This is a local-search algorithm that at every iteration chooses the *highest-valued neighbor* (using the standard neighborhood) of the current vertex as the next iterate. Since it compares the values of the neighbors, this algorithm requires any values of vertices on Hamming distance two to be comparable and different. The GSSA needs less than $2^{d-(1-\varepsilon)\log d}$ iterations, for any $0 < \varepsilon < 1$, to optimize CU-functions on d-dimensional hypercubes [4,37]. Unfortunately, an $\Omega(3^{d/3})$ lower bound is also known for the same problem [48]. Still, its practical behavior on some randomly generated distributions is one of the best [3].

6.1.2 Random Single Switch Algorithm (RSSA)

This is a local-search algorithm that at every iteration chooses uniformly at random one of the *higher-valued neighbors* of the current vertex as the next iterate. It has the same $2^{d-(1-\varepsilon)\log d}$ (expected) upper bound as the GSSA for optimizing CU-functions on hypercubes [4], but for the RSSA no nontrivial lower bounds are known. Also, the RSSA has expected quadratic running time on any *fully absorbing* function on hypercubes [49]. A function $f : \mathcal{H} \to \mathcal{D}$

is fully absorbing if it has dimension 0 or there is a facet F of $\mathcal{H}$ such that every vertex on F has a better value than its neighbor on the opposite facet $\mathcal{H} \setminus F$, and $f|_F$ is fully absorbing. This result generalizes to give an upper bound that is quadratic in the number of facets for fully absorbing functions on hyperstructures.

6.2 Multiple Switch Algorithms

When optimizing CLG-functions, we can perform "more aggressive" improvement steps, by moving to a next iterate that differs from the current vertex in more than one coordinate. The following theorem validates the correctness of this approach for CLG-functions (this is not always possible for RLG-functions).

Theorem 6.1 (Multi-switching for CLG-functions) *Let $f : \mathcal{P} \to \mathcal{D}$ be a CLG-function, $x \in \mathcal{P}$, and $y \in \mathcal{P}$ be a vector such that $y_j \neq x_j$ implies $f(x) < f(x$ with coordinate j changed to $y_j)$. Then $f(x) < f(y)$.*

Proof. Consider the subcube $\mathcal{H}$ spanned by x and y. All neighbors of x on $\mathcal{H}$ have bigger values, so x must be the unique local minimum of $f|_{\mathcal{H}}$. Indeed, if there was another local minimum, there would be a path on $\mathcal{H}$ from it to x containing only local minima of $f|_{\mathcal{H}}$ by Definition 4.2. Thus one of the neighbors of x on F would have the same value as x, contradicting the fact that on $\mathcal{H}$ all directions from x are improving. □

In particular, the function value is improved by moving from a vertex $x = (x_1, \ldots, x_d)$, with $VID_f(x) = (X_1, \ldots, X_d)$ to any vertex in $\prod_{j=1}^{d}(X_j \cup \{x_j\}) \setminus \{x\}$.

6.2.1 All Profitable Switches Algorithm (APSA)

This is a multi-switching algorithm that changes the current iterate in all coordinates where there are better neighbors. Thus if $x = (x_1, \ldots, x_d)$ and $VID_f(x) = (X_1, \ldots, X_d)$, then the next iterate $y = (y_1, \ldots, y_d)$ satisfies $y_i = x_i$ for all i such that $X_i = \emptyset$, and $y_i \in X_i$ for all other i. If X_i has more than one element, then one of them is chosen in an arbitrary fashion, e.g., randomly.

On hypercubes, this algorithm is deterministic. For CU-functions on hypercubes, it has the same $2^{d-(1-\varepsilon)\log d}$ upper bound as the GSSA [4], and a quadratic lower bound [3].

The algorithm for parity games suggested by Vöge and Jurdziński [46] is basically the APSA together with their strategy improvement measure.

No nontrivial upper bounds for the APSA on hyperstructures are known, even though experiments seem to indicate that in many circumstances it works extremely well [3,47].

6.2.2 Random Multiple Switches Algorithm (RMSA)

The Random Multiple Switches Algorithm (RMSA) at every iteration jumps to a next iterate taken uniformly at random from $\prod_{j=1}^{d}(X_j \cup \{x_j\}) \setminus \{x\}$.

On the Boolean hypercube, this algorithm computes $s \stackrel{\text{def}}{=} SIG_f(x)$, and inverts bits in v corresponding to a nonempty subset s' of the nonzero bits in s, chosen uniformly at random (i.e., $v' \stackrel{\text{def}}{=} v$ `XOR` s'.)

RMSA is a stepwise improvement algorithm by Theorem 6.1. Note that RMSA selects at random from a neighborhood that may be *exponentially big* in the dimension. So, strictly speaking, this is not a polynomial local-search improvement algorithm.

6.2.3 An Upper Bound for RMSA on Hypercubes

We now proceed to proving an upper bound of $2^{0.773d}$ for RMSA *on hypercubes*, which is better than the best known upper bounds for GSSA, RSSA, and APSA. This bound was first established by Mansour and Singh for policy iteration on Markov decision processes [37], but rediscovered independently in [4] in a more general context of CU-optimization. A new better bound is settled in the next section where we couple RMSA with random sampling.

We start by showing that the improvement in each step is exponential in the number of improving directions.

Lemma 6.2 *Let $f : \mathcal{H} \to \mathcal{D}$ be a CU-function. When RMSA jumps from a vertex v_0, with $i > 0$ improving directions with respect to f, to the next vertex v_1, then the expected number of vertices u with $f(v_0) < f(u) \leq f(v_1)$ is at least 2^{i-1}.*

Proof. Since each improving direction is chosen with probability 1/2, the algorithm will jump, with equal probability, to any of the 2^i vertices in the i-dimensional subcube defined by the improving directions. Since all of these vertices have better function values than v, the claim follows. □

Thus when running the RMSA on hypercubes, we can guarantee that the expected number of skipped vertices in each step is relatively high, provided there are many improving directions. The (binomial) distribution of numbers of improving directions over the hypercube is given by Theorem 4.7. The proof

of Theorem 6.6 assumes the worst (and seemingly unrealizable) case that the algorithm is always unlucky, selecting a vertex with the *fewest* possible number of improving directions, and the cube generated by these directions is numbered by the *smallest* possible values bigger than the current value. This settles an upper bound on the expected number of RMSA iterations in the worst case.

The following notation will be useful.

Notation 6.3 *Let* $H(c) = -c\log_2 c - (1-c)\log_2(1-c)$ *denote the binary entropy.* □

We also need two technical lemmas.

Lemma 6.4 (Estimating binomial sum) *For* $0 < k < d/2$ *one has*

$$\sum_{i=0}^{k-1} \binom{d}{i} < \frac{k}{d-2k} \cdot \binom{d}{k}.$$

Proof. See [14, p. 122]. □

Lemma 6.5 (Estimating binomial coefficients) *There are polynomials* p *and* q *such that, for any fixed* $c < \frac{1}{2}$ *and big enough* d*,*

$$p(d) \cdot 2^{H(c)d} \leq \binom{d}{\lfloor cd \rfloor} \leq q(d) \cdot 2^{H(c)d}.$$

Proof. Note that $(x+1)^{x+1} \leq x^{O(1)}(x+1)^x = x^{O(1)}(1+1/x)^x x^x \leq x^{O(1)}x^x$. Now apply Stirling's approximation for factorials, $k! = \Theta(k^{O(1)}) \cdot (k/e)^k$, in $\binom{d}{\lfloor cd \rfloor} = d!/(\lfloor cd \rfloor!(d - \lfloor cd \rfloor)!)$ to get

$$\begin{aligned}
\binom{d}{\lfloor cd \rfloor} &\overset{\text{[Stirling]}}{\geq} d^{O(1)} \cdot \frac{d^d}{(d-\lfloor cd \rfloor)^{d-\lfloor cd \rfloor} \cdot \lfloor cd \rfloor^{\lfloor cd \rfloor}} \\
&\overset{\text{[if } d-1 \geq cd \geq 1\text{]}}{\geq} d^{O(1)} \cdot \frac{d^d}{(d-cd+1)^{d-cd+1}(cd)^{cd}} \\
&\overset{[(x+1)^{x+1} \leq x^{O(1)}x^x]}{\geq} d^{O(1)} \cdot \frac{d^d}{(d-cd)^{d-cd}(cd)^{cd}} \\
&= d^{O(1)} \cdot 2^{H(c)d}.
\end{aligned}$$

This proves the first inequality. The second inequality can be shown in a similar way. □

We are ready to state and prove the upper bound for the Randomized Multiple Switches Algorithm (RMSA) for CU-optimization.

Theorem 6.6 *The expected number of iterations made by the RMSA on an d-dimensional CU-cube is $O(2^{0.773d}) = O(1.71^d)$.*

Proof. By Lemma 6.2, the RMSA is expected to skip at least 2^{i-1} vertices when it jumps from a vertex with i improving directions. By Theorem 4.7, there are exactly $\binom{d}{i}$ vertices with i improving directions. We consider a bound on the worst case, when the algorithm makes the smallest possible steps in each iteration. That is, it visits only vertices with $\leq k$ improving directions, for some $k \leq d$. To provide for the fewest possible iterations, k should be as small as possible, but it has to be big enough that all 2^d vertices are skipped over. We claim that we can take $k \equiv \lfloor cd \rfloor$ for some $c < 1/2$. Hence the problem reduces to minimizing c subject to

$$\sum_{i=1}^{\lfloor cd \rfloor} \binom{d}{i} 2^{i-1} \geq 2^d, \tag{3}$$

to guarantee that all vertices are skipped over. The number of steps taken by the algorithm is then at most

$$\sum_{i=1}^{\lfloor cd \rfloor} \binom{d}{i}. \tag{4}$$

Since the sum in (3) is bigger than its last term, we get

$$\sum_{i=1}^{\lfloor cd \rfloor} \binom{d}{i} 2^{i-1} \quad \geq \quad \binom{d}{\lfloor cd \rfloor} 2^{\lfloor cd \rfloor - 1} \tag{5}$$

$$\overset{\text{[Lemma 6.5]}}{\geq} d^{O(1)} \cdot 2^{H(c)d} \cdot 2^{\lfloor cd \rfloor - 1} \geq d^{O(1)} \cdot 2^{(H(c)+c)d}. \tag{6}$$

Thus, to satisfy (3), it is sufficient to guarantee $H(c) + c > 1$. Numerical evaluation shows that this holds whenever $0.2271 \leq c < 1/2$. Now we estimate the number of steps taken for $c = 0.2271$, by bounding (4) above using our two lemmas.

$$\sum_{i=1}^{\lfloor cd \rfloor} \binom{d}{i} \quad = \quad \sum_{i=1}^{\lfloor cd \rfloor - 1} \binom{d}{i} + \binom{d}{\lfloor cd \rfloor} \tag{7}$$

$$\overset{\text{[Lemma 6.4]}}{<} \left(\frac{\lfloor cd \rfloor}{d - 2 \cdot \lfloor cd \rfloor} + 1 \right) \binom{d}{\lfloor cd \rfloor}$$

$$\overset{\text{[Lemma 6.5]}}{\leq} d^{O(1)} \cdot 2^{H(c)d} \tag{8}$$

$$\overset{[c=0.2271]}{<} 2^{0.773d} < 1.71^d.$$

This concludes the proof. □

6.2.4 A Better Upper Bound for RMSA with Random Sampling (RMSA-RS)

The behavior and analysis for RMSA on hypercubes can be improved by adding random sampling. We want to start RMSA from a "good" vertex, with a value close to the optimum. Thus we select a good initial vertex as follows. Compute the value of a number of random vertices, pick the best one, and run RMSA starting from this vertex. We call the modified algorithm the RMSA-RS and parameterize it by the number of randomly sampled vertices. For this modified algorithm, a better upper bound can be shown, when we choose the parameter optimally:

Theorem 6.7 *The RMSA-RS can be parameterized in such a way that its expected running time on a d-dimensional CU-cube is* $O(2^{0.453d}) = O(1.37^d)$.

Proof. Suppose that we first take uniformly at random 2^{xd} samples of vertices on the CU-function, for some $x \in (0, 1)$, a parameter to be fixed later. The expected number of vertices with better value than all samples is $2^{d(1-x)}$. Thus, with the same argument as in the proof of Theorem 6.6, after choosing x, we need to minimize c subject to

$$\sum_{i=1}^{\lfloor cd \rfloor} \binom{d}{i} 2^{i-1} \geq 2^{(1-x)d}. \tag{9}$$

For instance, if $x = 1/2$, the expected number of vertices better than the best probed value will be at most $2^{d/2}$, and *any* subsequent iterative improvement will take at most $2^{d/2}$ iterations. But we hope to select $x < 1/2$, to allow RMSA to converge faster. The optimal value of x should allow taking c so that the number of samples is close to the number of iterations taken by RMSA. That is,

$$\sum_{i=1}^{\lfloor cd \rfloor} \frac{d}{i} \approx 2^{xd}.$$

We use (7)–(8) to guess $x = H(c)$. By (5)–(6) and (9), we need to find c such that $H(c) + c \geq 1 - H(c)$. Numerical evaluation gives $0.095 \leq c < 1/2$. Thus, for $c = 0.095$ both the number of probes and, by (7)–(8), the number of iterations taken in the subsequent call to RMSA are bounded by

$$2^{xd} = 2^{H(c)d} \overset{[c=0.095]}{<} 2^{0.453d} < 1.37^d.$$

This finishes the proof. □

Note that RMSA-RS as stated has a lower bound of $\Omega(2^{0.452d})$, because it makes that many samples. We can avoid this lower bound by running the

sampling process in parallel with RMSA, as follows. Start at some vertex and run RMSA, but in each iteration also take a random sample. The next iterate is the best of the random sample and the vertex jumped to by RMSA. This algorithm has the same upper bound as given by Theorem 6.7, but no known nontrivial lower bound.

7 Strategy Improvement

The games studied in this paper are all positionally determined; see Section 2. Thus, before the game starts, both players can select *positional strategies*, without reducing their chances in the game. When solving games, positional determinacy allows one to restrict attention to the *finite* set of positional strategies for each player. One of the most important methods for finding the best such strategies is *iterative strategy improvement*. Originally developed for Markov decision processes, this approach has been extensively used to solve games; see, e.g., [13,36,42,46,6]. It assigns values to the positional strategies of one of the players. An initial strategy is selected and then iteratively improved by finding strategies with better and better values.

The way values are assigned to strategies is crucial for strategy improvement algorithms to work. When seen as functions from the hyperstructure corresponding to the space of positional strategies, all strategy evaluations we consider satisfy the property that local optima are global. Furthermore, every global optimum corresponds to a strategy that is "sufficiently good". In parity games, this means that it wins in all vertices of the player's winning set. For all evaluation functions, the value of a strategy is a vector of values for the individual vertices of the game. This allows us to use the concepts of *attractive* single switches, and *stable* strategies. A switch is *attractive* if it selects a successor with a better value with respect to *the current strategy*. A strategy is *stable* if it has no attractive switches. The strategy evaluation functions we consider have the following two properties.

Profitability of attractive switches. Let σ be a strategy and v a vertex. Roughly speaking, if the value under σ of the successor $\sigma(v)$ of v is worse than the value of one of v's other successors (the switch to this successor is *attractive*), then changing σ in v to this other successor results in a better strategy (the switch is *profitable*). This property (attractive implies profitable) ensures that the algorithms can move to better strategies without actually evaluating the neighbors of the current one.

Optimality of stable strategies. If a strategy is *stable*, i.e., has no attractive switches, then it is globally optimal. Consequently, the algorithms can terminate as soon as a stable strategy is found, without evaluating its neighbors.

Together, these two properties imply that we can let the iterative improvement be guided by attractiveness of switches. As long as there are attractive switches, we can make them, knowing that they are profitable. And if there are no attractive switches, we know that the current strategy is globally optimal. Relying on attractiveness, rather than investigating neighbors explicitly, does not change the behavior of algorithms, only makes them more efficient.

In what follows, we will consider three strategy evaluation functions, for simple stochastic, mean payoff, and parity games. The first one we present, for simple stochastic games, is the oldest and best known. It is used by, e.g., Condon [13], who attributes it to Hoffman and Karp. Simple stochastic games are also the most general of the games we consider. The drawback of this measure is that each function evaluation involves solving a linear program with high precision. The measures for mean payoff and parity games avoid this. The mean payoff game function is the newest, recently discovered in [8]. It is discrete, still fairly simple, and efficient to compute. For parity games, Vöge and Jurdziński developed the first discrete strategy evaluation function in 2000 [46]. We present a modification that allows for better complexity analysis in some cases, first developed in [6].

7.1 Strategy Improvement Measure for Simple Stochastic Games

There is a well-known strategy evaluation function for simple stochastic games [13,36]. Given a strategy σ of Max, the value of a vertex is the probability of reaching the 1-sink when Max uses σ and Min uses an optimal counterstrategy against σ. The whole value for σ can be taken to be either a vector containing all vertex values, or simply their sum. Keeping track of the individual vertex values allows algorithms to make use of attractive switches. The counterstrategy of Min and the vertex values can be found by solving a linear program.

8 A Strategy Evaluation Function for Mean Payoff Games

This section presents a strategy evaluation function for computing 0-threshold partitions in mean payoff games. It straightforwardly applies to the p-threshold partition problem, for integer thresholds p, by subtracting p from every edge weight. If p is not an integer, the measure still works, with only a small extra factor in the complexity. It can also be used for solving the ergodic partition problem by dichotomy on the range of possible vertex values. Any one of the algorithms presented in earlier sections can be used together with the evaluation function. The main advantage compared to reducing to simple stochastic

games is that the measure is easy to compute, without solving high-precision linear programs.

The evaluation function has the property that any stable strategy ensures a value larger than 0 for MAX in every vertex where this is possible.

8.1 The Longest-Shortest Paths Problem

The key step in computing 0-threshold partitions can be explained by using a "controlled" version of the well-known *single source (target) shortest paths problem* on directed graphs. Suppose in a given digraph some set of *controlled* vertices is distinguished, and we can select *exactly one edge* leaving every controlled vertex, deleting all other edges from these vertices. Such a selection is called a *positional strategy*. We want to find a positional strategy that makes the shortest paths from all vertices to the distinguished sink as long as possible (also avoiding negative cycles that make the sink unreachable and the distances $-\infty$).

THE LONGEST-SHORTEST PATHS PROBLEM.
Given:

(1) a directed weighted graph G with unique sink t,
(2) some distinguished *controlled* vertices U of G, with $t \notin U$.

Find:

- a positional strategy σ such that in the graph G_σ the shortest simple path from every vertex to t is as long as possible (over all positional strategies).

If a negative weight loop is reachable, the length of the shortest path is $-\infty$, which MAX does not want. If only positive loops are reachable, and t is not, then the value is $+\infty$.

For our purposes it suffices to consider a version of the problem above with the following additional input data.

Additionally Given:

- some strategy τ, which guarantees that in the graph G_τ there are no negative cycles.

This additionally supplied strategy τ guarantees that the longest shortest distance from every vertex to the sink t is not $-\infty$; it is not excluded that τ or the optimal strategy will make some distances equal $+\infty$. The strategy

evaluation to be described ensures that improvement algorithms never consider a strategy that allows for negative cycles. The simplifying additional input strategy is easy to provide in the context of MPGs, as we show below. [4]

For DAGs, the longest-shortest path problem can be solved in polynomial time using dynamic programming. Start by topologically sorting the vertices and proceed backwards from the sink (distance 0), using the known longest-shortest distances for the preceding vertices.

8.2 Relating the 0-Threshold Partition and Longest-Shortest Paths Problems

The relation between computing 0-threshold partitions and computing longest shortest paths is now easy to describe. To find such a partition in an MPG G, add a *retreat* vertex t to the game graph with a self-loop edge of weight 0, plus a 0-weight *retreat* edge from every vertex of MAX to t. From now on, we assume G has undergone this transformation. Clearly, we have the following property.

Proposition 8.1 *Adding a retreat does not change the 0-threshold partition of the game, except that t is added to the $G_{\leq 0}$ part.* □

This is because we do not create any new loops allowing MAX to create positive cycles, or MIN to create new nonpositive cycles. MAX will prefer playing to t only if all other positional strategies lead to negative loops.

The key point is now as follows. Break the self-loop in t and consider the longest-shortest paths problem for the resulting graph, with t being the unique sink. We have the following equivalence:

Theorem 8.2 *The partition $G_{>0}$ consists exactly of those vertices for which the longest-shortest path distance to t is $+\infty$.* □

The longest shortest paths can be computed by iterative improvement. All the subexponential schemes described in Section 5 are directly applicable with the same bounds, and any other single- or multi-switching scheme works too – in particular those mentioned in Section 6; see also [8]. However, the *bounds* in Section 6 are not known to hold, since the evaluation function is not CLG (because it is not defined on all corners of the hyperstructure, and it is not known

[4] Actually, there may exist negative value loops consisting only of vertices from V_{MIN}. Such loops are easy to identify and eliminate in a preprocessing step. In the sequel we assume that this is already done.

whether it can be extended). Surprisingly (to our knowledge), the longest-shortest path problem was not previously addressed in the literature and its relation to the MPG problem was not noticed or exploited before. [5]

8.3 Retreats, Admissible Strategies, and Strategy Measure

As explained above, we modify an MPG by allowing MAX to "surrender" in every vertex. Add a retreat vertex t of MIN with a self-loop of weight 0 and a retreat edge of weight 0 from every vertex of MAX to t. Clearly, MAX secures value a > 0 from a vertex in the original MPG iff the same strategy does the same in the modified game. Assume from now on that the retreat has been added to G. Intuitively, the "add retreats" transformation is useful because MAX can start by a strategy that chooses the retreat edge in every vertex, thus "losing only 0". We call strategies "losing at most 0" *admissible*:

Definition 8.3 *A strategy σ of* MAX *in G is* admissible *if all loops in G_σ are positive, except the loop over t.* □

The strategy improvement algorithm iterates only through admissible strategies of MAX. This guarantees that the only losing (for MAX) loop in G_σ is the one over t.

8.3.1 Measuring the Quality of Strategies

We now define a measure that evaluates the "quality" of an admissible strategy. It can be computed in strongly polynomial time, as shown in Section 8.5.

Given an admissible strategy σ, the best MIN can do is to reach the 0-mean self-loop over t. Any other reachable loop will be positive, by the definition of an admissible strategy. The shortest path from every vertex v to t is well-defined, because there are no negative cycles in G_σ. Therefore, we define the value of a strategy as follows.

Definition 8.4 *For an admissible strategy σ of* MAX*, the value* $\mathrm{val}_\sigma(v)$ *of vertex v is defined as the shortest path distance from v to t in G_σ, or $+\infty$ if t is not reachable.* □

Note that there is a positional counterstrategy of MIN that guarantees the shortest paths are taken in each vertex, namely the strategy defined by the shortest path forest [15]. The relative quality of two admissible strategies is defined componentwise in the strategy value:

[5] The authors would appreciate any references.

Definition 8.5 *Let σ and σ' be two admissible strategies. Say that σ is* better than σ'*, formally $\sigma > \sigma'$, if $\mathrm{val}_\sigma(v) \geq \mathrm{val}_{\sigma'}(v)$ for all vertices $v \in V$, with strict inequality for at least one vertex. Say that $\sigma \geq \sigma'$, if $\sigma > \sigma'$ or they have equal values.* □

The following definition makes a distinction between switches that improve the strategy value, and switches that merely look like they do. Later we will prove that the two notions are equivalent.

Definition 8.6 *Given an admissible strategy σ, a switch in vertex v to u is:*

(1) attractive*, if $w(v,u) + \mathrm{val}_\sigma(u) > \mathrm{val}_\sigma(v)$;*
(2) profitable*, if $\sigma[v \mapsto u]$ is admissible and $\sigma[v \mapsto u] > \sigma$.* □

8.3.2 Requirements for the Measure

The following properties are crucial for any strategy improvement algorithm to work.

(1) Every admissible strategy that is better than all other admissible strategies (in the sense of Definition 8.5), is winning in every vertex of Max's winning set. This is evident from the definitions.
(2) If a strategy has no profitable switches then it is optimal. This is shown in two steps.
   (a) Every attractive switch is also profitable (Theorem 8.7).
   (b) A strategy without attractive switches wins in every vertex of Max's winning set (Theorem 8.8).

   These are the main theorems of the following section.

Property (2a) has another advantage: to find profitable switches, we only need to test attractivity, which is efficient as soon as the measure has been computed. Testing profitability would otherwise require recomputing the measure for every possible switch.

In addition, we prove the following property, allowing an algorithm to change the strategy in more than one vertex at a time.

3. If several switches in an admissible strategy σ are attractive at the same time, then making any subset of them results in a better strategy (Theorem 8.7).

In this section we prove the two major theorems, guaranteeing that every step is improving and that the final strategy is the best, respectively. Afterwards, we give two corollaries that are not strictly necessary for the algorithms to work, but which with little extra effort give additional insights into the problem.

Theorem 8.7 *If σ is an admissible strategy then any strategy obtained by one or more attractive switches is admissible and better. Formally, if the switches in s_i to t_i are attractive for $1 \leq i \leq r$ and $\sigma' \stackrel{def}{=} \sigma[s_1 \mapsto t_1][s_2 \mapsto t_2] \cdots [s_r \mapsto t_r]$, then σ' is admissible and $\sigma' > \sigma$.*

Proof. It is enough to prove that all loops in $G_{\sigma'}$ are positive and the value does not decrease in any vertex. Then it follows that σ' is admissible and the value in every s_i increases strictly, hence $\sigma' > \sigma$, because

$$\begin{aligned} \mathrm{val}_{\sigma'}(s_i) &= w(s_i, t_i) + \mathrm{val}_{\sigma'}(t_i) && \text{[by definition]} \\ &\geq w(s_i, t_i) + \mathrm{val}_{\sigma}(t_i) && [t_i\text{'s value does not decrease (to be shown)}] \\ &> \mathrm{val}_{\sigma}(s_i). && \text{[the switch in } s_i \text{ to } t_i \text{ is attractive]} \end{aligned}$$

First, we prove that every loop present in $G_{\sigma'}$, but not in G_σ, is positive. Second, we prove that for every path from some vertex v to t present in $G_{\sigma'}$, but not in G_σ, there is a shorter path in G_σ.

1. New loops are positive. Consider an arbitrary loop present in $G_{\sigma'}$, but not in G_σ. Some of the switching vertices s_i must be on the loop; denote them by $\{v_0, \ldots, v_{p-1}\} \subseteq \{s_1, \ldots, s_r\}$, in cyclic order; see Figure 1.

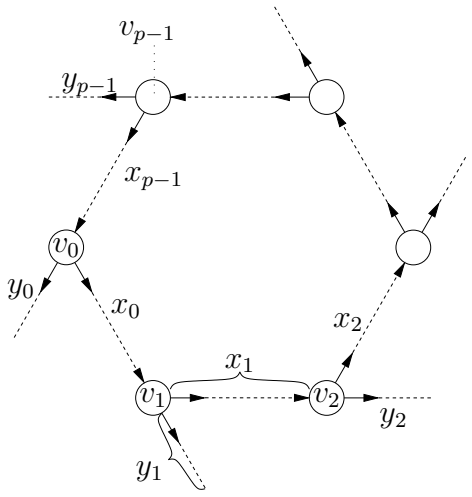

Figure 1. A loop in $G_{\sigma'}$ not in G_σ is depicted. Strategy σ breaks the cycle in vertices $v_0, \ldots, v_{p-1}$, and instead follows the dashed paths to t, each of total edge weight y_i. The weight of the segments between two adjacent v_i's is x_i.

To simplify notation, let $v_p \stackrel{\text{def}}{=} v_0$. Since the switch in v_i $(0 \le i \le p-1)$ is attractive, there is a path in G_σ from v_i to t. Denote by x_i the sum of weights on the shortest path from v_i to v_{i+1} under σ' and let $y_i = \mathrm{val}_\sigma(v_i)$, i.e., y_i is the sum of weights on the path from v_i to t under σ. Moreover, $x_p \stackrel{\text{def}}{=} x_0$ and $y_p \stackrel{\text{def}}{=} y_0$. Note that

$$y_i = \mathrm{val}_\sigma(v_i) < w(v_i, \sigma'(v_i)) + \mathrm{val}_\sigma(\sigma'(v_i)) \le x_i + y_{i+1},$$

where the first inequality holds because the switch in v_i to $\sigma'(v_i)$ is attractive and the second because $\mathrm{val}_\sigma(\sigma'(v_i))$ is the length of a shortest path from $\sigma'(v_i)$ to t and $x_i - w(v_i, \sigma'(v_i)) + y_{i+1}$ is the length of another path. Combining these p equalities for every i, we get

$$\begin{aligned} y_0 &< x_0 + y_1 \\ &< x_0 + x_1 + y_2 \\ &< x_0 + x_1 + x_2 + y_3 \\ &\;\;\vdots \\ &< x_0 + x_1 + \cdots + x_{p-1} + y_0, \end{aligned}$$

therefore, $x_0 + \cdots + x_{p-1} > 0$, hence the loop is positive.

2. New paths are longer. Consider an arbitrary shortest path from a vertex v to t, present in $G_{\sigma'}$ but not in G_σ. It must contain one or more of the switching vertices, say $\{v_1, \ldots, v_p\} \subseteq \{s_1, \ldots, s_r\}$, in order; see Figure 2.

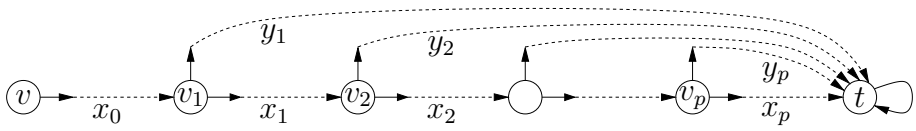

Figure 2. The path going right occurs in $G_{\sigma'}$ but not in G_σ. Strategy σ breaks the path in vertices $v_1, \ldots, v_p$ by going up and following the curved paths to t, each of total edge weight y_i. The edge weight of the segments between two adjacent v_i's is x_i.

Under strategy σ', denote by x_0 the sum of edge weights on the path from v to v_1, by x_p the sum of weights on the path from v_p to t, and by x_i $(1 \le i \le p-1)$ the sum of weights on the path from v_i to v_{i+1}. Let $y_i = \mathrm{val}_\sigma(v_i)$. As above, note that, for $1 \le i \le p-1$,

$$y_i < w(v_i, \sigma'(v_i)) + \mathrm{val}_\sigma(\sigma'(v_i)) \le x_i + y_{i+1},$$

for the same reason as in the previous case, and if we let $y_{p+1} = 0$ it holds for

$i = p$ too. Combining these p inequalities we obtain

$$\begin{aligned} x_0 + y_1 &< x_0 + x_1 + y_2 \\ &< x_0 + x_1 + x_2 + y_3 \\ &< x_0 + x_1 + x_2 + x_3 + y_4 \\ &\vdots \\ &< x_0 + \cdots + x_p. \end{aligned}$$

Thus the path from v to t in G_σ taking value $x_0 + y_1$ is shorter than the new path in $G_{\sigma'}$. □

We also show that every strategy that does not have attractive switches is optimal. This guarantees that the solution computed by the algorithm is indeed an optimal strategy.

Theorem 8.8 *If σ is an admissible strategy with no attractive switches, then $\sigma \geq \sigma'$ for all admissible strategies σ'.*

Proof. The proof is in two steps: first we prove the special case when MIN does not have any choices, and then we extend the result to general games.

1. One-player games. Assume MIN does not have any choices, i.e., the out-degree of every vertex in V_{MIN} is one. Let σ be an admissible strategy with no attractive switches. We claim that every admissible strategy σ' is no better than σ, i.e., $\sigma' \leq \sigma$.

1a. Finite values cannot become infinite. First, we prove that if $\text{val}_\sigma(v) < \infty$ then $\text{val}_{\sigma'}(v) < \infty$. Consider an arbitrary loop formed in $G_{\sigma'}$, not formed in G_σ. There is at least one vertex on this loop where σ and σ' make different choices; assume they differ in the vertices $v_0, \ldots, v_{p-1}$ in cyclic order: again, Figure 1 depicts the situation, and again let $v_p \stackrel{\text{def}}{=} v_0$. Denote by x_i the sum of edge weights on the path from v_i to v_{i+1} under strategy σ', and let $y_i = \text{val}_\sigma(v_i)$, i.e., y_i is the sum of edge weights on the path from v_i to t under strategy σ. Let $x_p \stackrel{\text{def}}{=} x_0$ and $y_p \stackrel{\text{def}}{=} y_0$. The condition "no switch is attractive for σ" says exactly that $y_i \geq x_i + y_{i+1}$. Combining these p inequalities, we obtain

$$\begin{aligned} y_0 &\geq x_0 + y_1 && \text{[by non-attractiveness in } v_0\text{]} \\ &\geq x_0 + x_1 + y_2 && \text{[by non-attractiveness in } v_1\text{]} \\ &\geq x_0 + x_1 + x_2 + y_3 && \text{[by non-attractiveness in } v_2\text{]} \\ &\vdots \\ &\geq x_0 + x_1 + \cdots + x_{p-1} + y_0. && \text{[by non-attractiveness in } v_{p-1}\text{]} \end{aligned}$$

Thus $x_0 + x_1 + \cdots + x_{p-1} \leq 0$. Consequently, every loop present under σ', but not under σ, is losing for MAX.

1b. Finite values do not improve finitely. Assume $\mathrm{val}_\sigma(v) < \mathrm{val}_{\sigma'}(v) < \infty$. As in the previous case, consider the path from v to t under σ'. The strategies must make different choices in one or more vertices on this path, say in $v_1, \ldots, v_p$, in order: Figure 2 applies to this proof too.

Under strategy σ', denote by x_0 the sum of weights on the path from v to v_1, by x_p the sum of weights on the path from v_p to t, and by x_i $(1 \le i \le p-1)$ the sum of weights on the path from v_i to v_{i+1}. Let $y_i = \mathrm{val}_\sigma(v_i)$, i.e., y_i is the sum of weights on the path from v_i to t under σ. The condition "no switch is attractive for σ" says exactly that $y_i \ge x_i + y_{i+1}$, for $1 \le i \le p-1$, and $y_p \ge x_p$. Combining these p inequalities, we obtain

$$\begin{aligned}
y_1 &\ge x_1 + y_2 && \text{[by non-attractiveness in } v_1\text{]}\\
&\ge x_1 + x_2 + y_3 && \text{[by non-attractiveness in } v_2\text{]}\\
&\ge x_1 + x_2 + x_3 + y_4 && \text{[by non-attractiveness in } v_3\text{]}\\
&\vdots\\
&\ge x_1 + x_2 + \cdots + x_{p-1} + y_p && \text{[by non-attractiveness in } v_{p-1}\text{]}\\
&\ge x_1 + x_2 + \cdots + x_{p-1} + x_p, && \text{[by non-attractiveness in } v_p\text{]}
\end{aligned}$$

and in particular $x_0 + y_1 \ge x_0 + \cdots + x_p$. But $x_0 + y_1 = \mathrm{val}_\sigma(v)$ and $x_0 + \cdots + x_p = \mathrm{val}_{\sigma'}(v)$, so indeed we cannot have better finite values under σ' than under σ.

2. Two-player games. Finally, we prove the claim in full generality, where Min may have choices. The simple observation is that Min does not need to use these choices, and the situation reduces to the one-player game we already considered. Specifically, let σ be as before and let τ be an optimal counterstrategy of Min, obtained from the shortest-path forest for vertices with value $< \infty$ and defined arbitrarily in other vertices. Clearly, in the game G_τ, the values of all vertices under σ are the same as in G, and σ does not have any attractive switches. By Case 1, in G_τ we also have $\sigma \ge \sigma'$. Finally, σ' cannot be better in G than in G_τ, since the latter game restricts choices for Min. □

The following property is not necessary for correctness, but completes the discussion on attractiveness versus profitability.

Corollary 8.9 *If σ is an admissible strategy and an admissible strategy σ' is obtained from σ by one or more* non-attractive *switches, then $\sigma' \le \sigma$.*

Proof. Consider the game (V, E', w'), where $E' = E \setminus \{(u,v) : u \in V_{\mathrm{Max}} \wedge \sigma(u) \ne v \wedge \sigma'(u) \ne v\}$ and w' is w restricted to E'. In this game, σ has no attractive switches, and both σ and σ' are strategies in the game, hence by Theorem 8.8, $\sigma \ge \sigma'$. □

As a special case (together with Theorem 8.7), we have the following equivalence.

Corollary 8.10 *A single switch is attractive if and only if it is profitable.* □

8.5 Efficient Computation of the Measure

The measure can be straightforwardly computed using the Bellman–Ford algorithm for single-sink shortest paths in graphs with negative weights [15]. This algorithm requires time $O(n \cdot |E|)$. The bound on the depth of the measure is $O(n \cdot W)$, because every vertex can have its shortest path length improved at most $O(n \cdot W)$ times. There are n vertices, so the number of switches cannot exceed $O(n^2 \cdot W)$. Together with the $O(n \cdot |E|)$ bound per iteration this gives total time $O(n^3 \cdot |E| \cdot W)$. In this section we show how to reuse the measure computed in previous iteration to improve this upper bound to $O(n^2 \cdot |E| \cdot W)$. Since there are no known nontrivial lower bounds on the number of improvement steps, it is practically important to reduce the cost of each iteration.

We first compute the set of vertices that have different values under the old and the new strategies, and then we recompute the values only in these vertices, using the Bellman–Ford algorithm. If the algorithm improves the value of n_i vertices in iteration i, the Bellman–Ford algorithm will only need to relax each edge n_i times, hence runs in time $O(n_i \cdot |E|)$. Due to the depth of the measure, the sum of all n_i's does not exceed $n^2 \cdot W$, so the total running time becomes at most $O(n^2 \cdot |E| \cdot W)$. It remains to compute which vertices need to change their values. Algorithm 4 does this, taking as arguments a game G, the set $U \subseteq V$ of switching vertices and the previous measure $d : V \to \mathbb{N} \cup \{\infty\}$.

Algorithm 4: Mark all vertices v for which $\mathrm{val}_\sigma(v) \neq \mathrm{val}_{\sigma'}(v)$.
MARK-CHANGING-VERTICES($G, U \subseteq V, d : V \to \mathbb{N} \cup \{\infty\}$)
(1) mark all vertices in U
(2) **while** $U \neq \emptyset$
(3) remove some vertex v from U
(4) **foreach** unmarked predecessor u of v in $G_{\sigma'}$
(5) **if** $w(u, x) + d[x] > d[u]$ for all unmarked successors x of u in $G_{\sigma'}$
(6) mark u
(7) $U \leftarrow U \cup \{u\}$

Proposition 8.11 *If an attractive (multiple) switch changes strategy σ to σ', then for every vertex $v \in V$, the following statements are equivalent.*

(1) Algorithm 4 marks v.
(2) Every shortest path from v to t in G_σ passes through some switch vertex.
(3) $\mathrm{val}_\sigma(v) \neq \mathrm{val}_{\sigma'}(v)$.

Proof. (1) $\implies$ (2). By induction on the set of marked vertices, which expands as the algorithm proceeds. The base case holds since the vertices marked before the while loop are the switch vertices; these clearly satisfy (2). For the induction step, assume the claim holds for every marked vertex and that vertex u is about to be marked on line 6. Let x be any successor of u included in some shortest path from u to t in G_σ. Since $w(u,x) + d[x] = d[u]$, and by the condition on line 5, x must be marked already, and by the induction hypothesis every shortest path through x passes through U. This completes the induction step.

(2) $\implies$ (1). For an arbitrary vertex v, consider all its shortest paths in G_σ. Denote by $\overline{v}$ the maximal number of edges passed by such a path before a vertex in U is reached (so $\overline{v}$ is the unweighted length of an initial segment). The proof is by induction on $\overline{v}$. The base case is clear: $\overline{v} = 0$ if and only if $v \in U$, and all vertices in U are marked. Assume inductively that the claim holds for all vertices u with $\overline{u} < k$ and consider an arbitrary vertex v with $\overline{v} = k$. By the inductive hypothesis, all successors of v that occur on a shortest path are marked. Hence, when the algorithm removes the last of them from U, the condition on line 5 is triggered and v is marked.

(3) $\implies$ (2). If some shortest path from v to t in G_σ does not pass through a switch vertex, then the same path is available also in $G_{\sigma'}$, hence $\mathrm{val}_\sigma(v) = \mathrm{val}_{\sigma'}(v)$.

(2) $\implies$ (3). Assume (2) and consider an arbitrary shortest path from v to t in $G_{\sigma'}$. If it contains any switch vertices, let u be the first of them. The same path from v to u, followed by the path in G_σ from u to t, gives a shorter path in G_σ, since the length of shortest paths strictly increase in switch vertices. If the path does not contain any switch vertices, then by (2) it is longer than every shortest path in G_σ. □

We thus showed that Algorithm 4 does what it is supposed to. To finish the argument, we show that it runs in time $O(|E|)$, so it is dominated by the time used by the Bellman–Ford algorithm.

Proposition 8.12 *Algorithm 4 can be implemented to run in time $O(|E|)$.*

Proof. Every vertex can only be added to U once, so the number of iterations of the while loop is at most n. The condition on line 5 can be tested in constant time if we keep track, for each vertex v, of the number of unmarked successors u with $w(u,x) + d[x] = d[u]$. Thus, the time taken by the foreach loop is linear in the number of predecessors of v, and the claim follows. □

The strategy evaluation function for mean payoff games presented in this section applies to solving the p-threshold partition problem for integer thresholds p. It can be used together with all the algorithms surveyed in Sections 5 and 6, with the same subexponential complexity bound $2^{O(\sqrt{n_0 \log n_0})}$ for the algorithms from combinatorial linear programming.

Solving the p-threshold partition problem is sufficient for some applications, e.g., finding the winner in a parity game by reduction to mean payoff games, but what about finding the exact values of all vertices and computing the ergodic partition? This can be done by dichotomy on the interval of possible edge weights. When the intervals get smaller, we have to consider the case where the threshold value is a non-integral rational. This means that after subtracting the threshold, the edge weights are no longer integers. They are, however, all rationals with the same denominator. Straightforward computations show that this only adds a linear factor to the depth of the measure [8], and the ergodic partition problem can be solved in time $\log W \cdot 2^{O(\sqrt{n_0 \log n_0})}$.

Since the evaluation function does not assign values to all strategies, only to admissible ones, it is strictly speaking not CLG, however. Whether it can be modified or efficiently extended to a CLG function is still open.

9 A Strategy Evaluation Function for Parity Games

In this section we describe a function EVALUATE that assigns values from a partially ordered codomain to strategies of Player 0 in parity games.[6] In Section 10 we show that this function is CLG, and therefore all algorithms described in earlier sections apply to solving games. The function we describe is similar to the one by Vöge and Jurdziński, but more economical, in the sense that the longest possible improving chain in the partial order on the codomain is shorter, $O(n^3(n/k+1)^k)$; see Section 9.4. This gives an additional bound on the number of improvement steps. As a consequence, with our function, the local search type algorithms from Section 6 get a substantially improved upper bound for optimizing parity games, and the Kalai- and Matoušek–Sharir–Welzl-style algorithms from Section 5 get a

$$\min\left(O\left(n^3\left(\frac{n}{k}+1\right)^k\right), 2^{O(\sqrt{n_0 \log n_0})}\right)$$

[6] Actually, every parity game has its own EVALUATE function.

upper bound on the number of iterations.

9.1 Values, Comparisons, Attractive Switches

In this section we define the value of a strategy under EVALUATE – the target iteratively improved by our algorithms, until a local optimum is found (which will also be proved global). Given a strategy σ of Player 0, its value is a vector of values of all vertices of the game. The value of each vertex is computed with respect to the pair of strategies (σ, τ) of Players 0 and 1, respectively, where τ is an '*optimal*' response counterstrategy against σ. The optimality of τ is essential for guiding Player 0 in improving σ. We delay the issue of constructing optimal counterstrategies until Section 9.5, assuming for now that Player 1 always responds with an optimal counterstrategy.

9.1.1 Vertex Values

For technical reasons to be explained shortly, each vertex is assigned a unique value, called a *tint*.

Definition 9.1 (Tints) *A bijection* $\mathbf{t} : V \to \{1, \ldots, n\}$ *such that*

$$c(u) \leq c(v) \Rightarrow \mathbf{t}(u) \leq \mathbf{t}(v)$$

assigns tints to vertices. The color of a tint $s \in \{1, \ldots, n\}$ *equals* $c(\mathbf{t}^{-1}(s))$. □

Note that tints of vertices of the same color form a consecutive segment of natural numbers. Subsequently we identify vertices with their tints, and slightly abuse notation by writing $c(t)$ for the color of the vertex with tint t.

Definition 9.2 (Winning and Losing Colors and Tints) *Even colors are called* winning for Player 0 *and odd colors are* winning for Player 1 . *Tint* t *is* winning for Player i *if its color* $c(t)$ *is winning for Player* i. *A color or tint is* losing *for a player if it is winning for his adversary.* □

Note that tints of different colors are ordered as these colors. In Section 9.1.2 we will define that within the same winning (resp. losing) color the bigger (resp. smaller) tint is better for Player 0.

When the players fix their positional strategies, the *trace* of a play from any vertex is defined as the set of vertices visited: a (possibly empty) simple path leading to a simple loop. Roughly, the value of a vertex with respect to a pair of positional strategies consists of a loop value (largest or major tint) and a

path value (a record of the numbers of more significant colors on the path to the major, plus the length of this path), as defined below.

Notation 9.3 *Denote by V^i the set of vertices of* color *i and by $V^{>t}$ the set of vertices with* tints *numerically bigger than t.* □

Definition 9.4 (Trace, Loop Major) *Suppose the players fix positional strategies σ and τ, respectively (not necessarily optimal). Then from every vertex u_0 the* trace of the play *is the sequence of vertices visited, up to the first repetition. It takes a δ-shape form: an initial simple path (of length $q \geq 0$) ending in a loop of length $s - q$:*

$$u_0, \dots u_q, \dots u_s = u_q, \tag{10}$$

where all vertices u_i are distinct, except $u_q = u_s$. The vertex of maximal tint on the loop is called the loop major. □

Definition 9.5 (Vertex Values and Path Values) *Consider the trace of a play $u_0, \dots, u_q, \dots, u_s = u_q$ with respect to a pair of positional strategies σ and τ from vertex u_0 as in Definition 9.4. Let u_r, with $r \in \{q, \dots, s-1\}$, be the loop major. The* value *$\nu_{\sigma,\tau}(u_0)$ of u_0 with respect to σ and τ has the form (t, P, p) and consists of:*

Loop Value (Tint) *t equal to the tint $\mathbf{t}(u_r)$ of the loop major.*
Path Color Hit Record Relative to t *defined as a vector*

$$P = (m_k, m_{k-1}, \dots, m_j, \underbrace{0, \dots, 0}_{j-1 \text{ times}}),$$

where $j = c(t)$ is the color of the major tint t, and

$$m_i = \left|\{u_0, u_1, \dots, u_r\} \cap V^i \cap V^{>t}\right|$$

is the number of vertices of color $i \geq j$ on the path beginning in u_0 and ending the first time it hits the the loop major (except that for the color j of the major we account only for the vertices with tints bigger than t.)
Path Length *$p = r$.*

Call the pair (P, p) the path value. □

Note that the path value is $(\overline{0}, 0)$ for the loop major and that the color hit record is $\overline{0}$ for all vertices on the loop.

The reason of the complexity of this definition is to meet the criteria enumerated in the beginning of Section 9 and simultaneously obtain the 'tightest

possible' bound on the number of iterative improvements. Theorem 9.26 settles such a bound for the measure from Definition 9.5, namely $O(n^3 \cdot (n/k+1)^k)$.

Vöge and Jurdziński [46] assign similar values to vertices with respect to a pair of strategies. The difference lies in the definition of path color hit records. The evaluation in [46] records the set of all vertices with bigger tints than the loop major, instead of recording the number of vertices of each color. The benefit of our modification is that instead of 2^n possible path color hit records, we get at most $(n/k+1)^k$; see Theorem 9.26.

9.1.2 Value Comparison

Now we proceed to comparison of vertex and path values. Later this allows us to define what it means for a strategy to be 'better' than another, with respect to the measure. In the sequel, when saying 'attractive', 'better', 'worse', 'profitable', etc., we consistently take the viewpoint of Player 0.

Definition 9.6 (Preference Orders) *The* preference order *on colors (as seen by Player 0) is defined as follows:*

$$c \prec c' \quad \textit{iff} \quad (-1)^c \cdot c < (-1)^{c'} \cdot c'.$$

The preference order *on tints (as seen by Player 0) is as follows:*

$$t \prec t' \quad \textit{iff} \quad (-1)^{c(t)} \cdot t < (-1)^{c(t')} \cdot t'. \qquad \square$$

We thus have $\cdots \prec 5 \prec 3 \prec 1 \prec 0 \prec 2 \prec 4 \prec \ldots$ on colors.

Definition 9.7 ("Lexicographic" Ordering) *Given two vectors (indexed in descending order from the maximal color k to some $l \geq 1$)*

$$P = (m_k, m_{k-1}, \ldots, m_{l+1}, m_l),$$
$$P' = (m'_k, m'_{k-1}, \ldots, m'_{l+1}, m'_l),$$

define $P \prec P'$ if the vector

$$\Big((-1)^k \cdot m_k, (-1)^{k-1} \cdot m_{k-1}, \ldots, (-1)^{l+1} \cdot m_{l+1}, (-1)^l \cdot m_l\Big)$$

is lexicographically smaller (assuming the usual ordering of integers) than the vector $\Big((-1)^k \cdot m'_k, (-1)^{k-1} \cdot m'_{k-1}, \ldots, (-1)^{l+1} \cdot m'_{l+1}, (-1)^l \cdot m'_l\Big)$. $\square$

This ordering is consistent with what we consider 'better' for Player 0.

Definition 9.8 (Path Value Comparison) *For two vertex values* (t, P_1, p_1) *and* (t, P_2, p_2)*, where* t *is a tint,* $j = c(t)$ *is its color, and*

$$P_1 = (m_k, m_{k-1}, \ldots, m_{j+1}, m_j, \ldots, m_1),$$
$$P_2 = (m'_k, m'_{k-1}, \ldots, m'_{j+1}, m'_j, \ldots, m'_1),$$

say that $(P_1, p_1) \prec_t (P_2, p_2)$ *(pronounced: for Player 0, the path value* (P_2, p_2) *is* more attractive modulo t *than the path value* (P_1, p_1)*), if:*

(1) either $(m_k, m_{k-1}, \ldots, m_{j+1}, m_j) \prec (m'_k, m'_{k-1}, \ldots, m'_{j+1}, m'_j)$,
(2) or $(m_k, m_{k-1}, \ldots, m_{j+1}, m_j) = (m'_k, m'_{k-1}, \ldots, m'_{j+1}, m'_j)$ *and*

$$(-1)^j \cdot p_1 > (-1)^j \cdot p_2. \tag{11}$$

Remark. Note that (11) means that shorter paths are better for Player 0 when the loop tint t is 0-winning, and vice versa. □

Definition 9.9 (Vertex Value Comparison) *For two vertex values define* $(t_1, P_1, p_1) \prec (t_2, P_2, p_2)$ *if*

(1) either $t_1 \prec t_2$,
(2) or $t_1 = t_2 = t$*, and* $(P_1, p_1) \prec_t (P_2, p_2)$. □

Definition 9.10 (Vertex Values) *The value* $\nu_\sigma(v)$ *of a vertex* v *with respect to a strategy* σ *of Player 0 is the minimum of the values* $\nu_{\sigma,\tau}(v)$*, taken over all strategies* τ *of Player 1.* □

In Section 9.5 we show that the 'minimum' in this definition can be achieved in all vertices simultaneously by a positional strategy τ of Player 1.

Definition 9.11 (Strategy Value) *The* value *assigned to a strategy* σ *of Player 0 by* Evaluate *is a vector of values of all vertices with respect to the pair of strategies* (σ, τ)*, where* τ *is an optimal response counterstrategy of Player 1 against* σ*; see Section 9.5.* □

Strategies are compared componentwise in all vertices:

Definition 9.12 (Strategy Comparison) *A strategy* σ' improves σ*, symbolically* $\sigma \prec \sigma'$*, if:*

(1) $\nu_\sigma(v) \preceq \nu_{\sigma'}(v)$ *for all vertices* v*, and*
(2) there is at least one vertex u *with* $\nu_\sigma(u) \prec \nu_{\sigma'}(u)$ □

We conclude this section with a simple but important property.

Proposition 9.13 (Transitivity) *The relations $\prec$ on colors, tints, vertex values, strategies, and $\prec_t$ on path values (for each t) are transitive.* □

9.1.3 Attractive Switches

We are now ready to formally define attractive switches.

Definition 9.14 (Attractive Switch) *Let v_1, v_2 be successors of a vertex v, let σ be a positional strategy of Player 0 with $\sigma(v) = v_1$, and let $\nu_\sigma(v_1) = (t_1, P_1, p_1)$ and $\nu_\sigma(v_2) = (t_2, P_2, p_2)$ be the values with respect to σ of the vertices v_1 and v_2, respectively. Consider a single switch in strategy σ, consisting in changing the successor of v from v_1 to v_2. The switch is called* attractive *if:*

(1) $(t_1, P_1, p_1) \prec (t_2, P_2, p_2)$,
(2) unless $t_1 = t_2 = \mathbf{t}(v)$ *and* $P_1 = P_2 = \overline{0}$.

The second condition in the preceding definition is necessary in order for every attractive switch to give a value improvement. Indeed, if $t_1 = t_2 = \mathbf{t}(v)$ and $P_1 = P_2 = \overline{0}$, then Player 1 can ensure the same value in every vertex after the switch by using the same counterstrategy as against σ. For both v_1 and v_2 the values will remain the same because they use the same path as before to the loop major v. Since v is loop major both before and after the switch it will also have the same value as before.

Remark. Note that deciding whether a switch is attractive (when comparing values of its successors) we do not directly account for the color/tint of the current vertex. However, this color/tint may be included in the values of successors possibly dependent on the current vertex. □

In Section 9.2 we show profitability of attractive switches, guaranteeing that the new strategy is better in the sense of Definition 9.12. In Section 9.4 we bound the maximal possible number of attractive/profitable switches.

Any optimization algorithm can terminate once it finds a stable strategy, without attractive switches.

9.1.4 An Example of Strategy Evaluation and Iterative Improvement

Example 9.15 To illustrate the computation of the measure and the general flavor of strategy improvement algorithms, Figure 3 shows an example parity game with a sequence of improving strategies.

1. The parity game. Player 0 owns vertices a and c, and Player 1 owns vertex b.

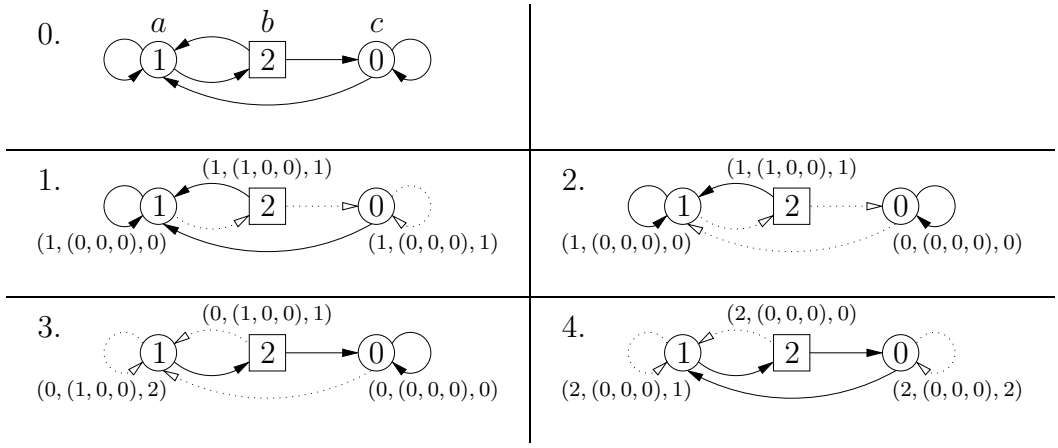

Figure 3. An example run of a strategy improvement algorithm.

In this particular game, there are three colors and three tints, and they happen to coincide. We will now show a run of a strategy improvement algorithm using the measure defined earlier in this section. Steps 1–4 show the strategies in each step. Dotted arrows are those *not* selected by the current strategy of Player 0 and the corresponding optimal counterstrategy of Player 1.

2. In the initial strategy, Player 0 goes from a to a and from c to a. Player 1's optimal counterstrategy goes left in b, since that choice provides for a shorter path length. The switch in a is attractive since b has a better value than a (the good color 2 is in the color hit record) and the switch in c is attractive since the path length is longer and the loop is losing.

3. Assume Player 0 makes the switch in b (the one in a was also attractive). The optimal counterstrategy of Player 1 is still to go left, since it gives a better loop value in b. As guaranteed by Theorem 9.20, the new strategy gave a better value even after Player 1 recomputed his counterstrategy. Now the switch in c is non-attractive, but the switch in a is still attractive.

4. Player 0 has now switched in a. The optimal counterstrategy of Player 1 has now changed: going left would give a loop over a and b, with tint 2, but going right provides the 1-better loop tint 0. The vertex values of both a and b have improved. The switch in c has become attractive, because the good color 2 appears in the color hit record.

5. After switching in c, Player 0 has now reached a strategy with no attractive switches. As guaranteed by Theorem 9.21, this means that the strategy is optimal. Note that *both* Player 1's two possible counterstrategies are equally good at this point: even if going right gives a greater loop length, this is not reflected in any vertex value (the path length of the loop major is 0).

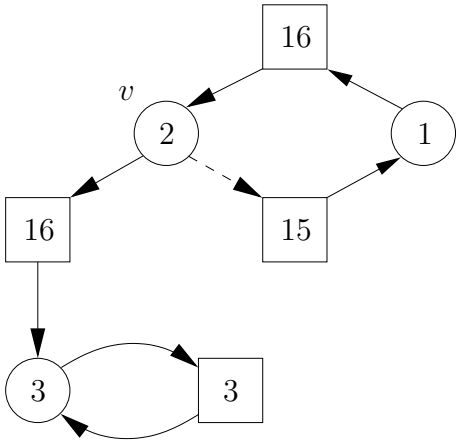

Figure 4. With a more shallow measure, the non-attractive switch in v allows Player 0 to win in more vertices.

9.1.5 Are More Shallow Measures Possible?

One could imagine even tighter measures than given by Definition 9.5. For instance, instead of our color hit record, one could try to record only one bit for every color $> c(t)$, indicating whether a vertex of this color appears on the path. However, this definition would violate the *optimality of stable strategies* required in the beginning of Section 9, as demonstrated by Figure 4. Round and square vertices in Figure 4 belong to Player 0 and 1 respectively. The value of the left successor of vertex v colored 2 would be better than the value of the right successor, since the odd color 15 is part of the path from the vertex colored 15 to the loop. However, switching right in v (with a worse value) would let Player 0 win in more vertices. We thus have a stable strategy that is not optimal.

9.1.6 Path Comparison Properties

Proofs in the subsequent sections rely on the following technical lemmas.

Lemma 9.16 *If* $(P_0, p_0) \succeq_t (P_1, p_1)$*, then* $(Q + P_0, q + p_0) \succeq_t (Q + P_1, q + p_1)$*, where* $Q + P_i$ *is vector addition.*

Intuitively, by extending the paths P_0 and P_1 with the same path Q, we do not change their initial order.

Proof. The first position in which the vectors $Q + P_0$ and $Q + P_1$ differ is the same as the first position in which P_0 and P_1 differ. If this position is one of $k, \ldots, c(t)$, then $(Q + P_0, q + p_0) \succ_t (Q + P_1, q + p_1)$, since $(P_0, p_0) \succeq_t (P_1, p_1)$. If, on the other hand, P_0 and P_1 coincide in all positions $k, \ldots, c(t)$, then $(-1)^{c(t)} \cdot (q + p_0) \leq (-1)^{c(t)} \cdot (q + p_1)$ iff $(-1)^{c(t)} \cdot p_0 \leq (-1)^{c(t)} \cdot p_1$, and the conclusion follows. □

Lemma 9.17 *Let $p \geq 0$, $q > 0$ be integers and P, Q be path color hit records relative to a tint t, where Q corresponds to a path with the largest tint $t' \succ t$. Then $(P, p) \prec_t (P + Q, p + q)$.*

Intuitively, adding a good path Q to P gives an improvement.

Proof. Assume $t \prec t'$.

If $t < t'$ (numeric comparison) then t' is 0-winning and the first position (counting from the largest color k in decreasing order) where P and $P + Q$ are different is $c(t')$, and $P + Q$ is numerically greater at this position. Hence $(P, p) \prec_t (P + Q, p + q)$.

If $t' < t$ then t is 0-losing and P and $P+Q$ coincide in all positions $k, \ldots, c(t)$. Since t is 0-losing, the larger path length determines that $(P, p) \prec_t (P+Q, p+q)$. □

9.2 Profitability of Attractive Switches

Our current aim consists in showing that optimization algorithms may proceed by making attractive switches. Attractiveness is established locally, by comparing values of the successors of a vertex with respect to the current strategy; see Definition 9.14. In this section we prove that making an attractive switch actually leads to an improvement, which is captured by the notion of profitability.

Definition 9.18 (Profitability) *Say that a single switch in a vertex v from σ to σ' is* profitable *if:*

(1) $\nu_\sigma(w) \preceq \nu_{\sigma'}(w)$, for all vertices w, and
(2) $\nu_\sigma(v) \prec \nu_{\sigma'}(v)$. □

Condition 1 requires that all vertices (non-strictly) improve their values after a switch, while 2 stipulates a strict value improvement for the switch vertex itself.

Proceeding by profitable switches guarantees termination, as follows from the definition.

Proposition 9.19 *Every sequence of profitable switches terminates.* □

Section 9.4 establishes an upper bound on the number of profitable switches.

Theorem 9.20 (Profitability) *Every attractive switch is profitable.*

(Corollary 9.23 will show the converse, i.e., the notions of profitable and attractive are actually *equivalent*.)

Proof. Consider an attractive (say left-to-right) switch in vertex v: from vertex v_1 with value (t_1, P_1, p_1) to vertex v_2 with value (t_2, P_2, p_2). Let σ and σ' be the strategies before and after the switch, respectively. Let τ and τ' be optimal counterstrategies against σ and σ', respectively. Such counterstrategies exist by Lemma 9.28.

We start by proving that the second property of Definition 9.18 holds for all attractive switches. There are two major cases.

Case 1. In his optimal response after the switch in v, Player 1 *does not revisit* v starting from v_2. Assume $\nu_{\sigma'}(v_2) \prec \nu_\sigma(v_2)$. This contradicts the fact that τ is optimal, since the trace taken by τ' not going through v was available also before the switch. So $\nu_{\sigma'}(v_2) \succeq \nu_\sigma(v_2)$. If $t_1 \prec t_2$ the loop value at v will improve after the switch. If $t_1 = t_2$, Lemma 9.16 implies $\nu_\sigma(v) \prec \nu_{\sigma'}(v)$. This finishes the proof of Case 1.

Case 2. In his optimal response after the switch in v, Player 1 *does revisit* v starting from v_2. Consequently, we get a loop through v and v_2. We should prove that the value of v on this loop is better for Player 0 than the value before the switch.

In this case, the loop values t_1 and t_2 *coincide*. Indeed, Player 1 can reach v from v_2, for which the loop value before the switch is t_1, so by optimality of τ we have $t_2 \preceq t_1$. Also, attractiveness means that $t_2 \succeq t_1$. Denote $t := t_1 = t_2$.

Any path from v_2 to v could have been taken by Player 1 *before the switch* (i.e., under σ, τ). Respectively, we analyze cases when such a path was or was not used by Player 1 before the switch.

Subcase 2.1. Assume τ can be chosen such that the path from v_2 to t goes through v, and consider some such choice of τ. By definition, (P_2, p_2) is the $\prec_t$-smallest path value of any paths from v_2 to t.

1. If $P_1 \neq P_2$, then the largest color in $P_2 - P_1$ is even, since the switch is attractive. So the largest tint t' on the path from v_2 to v is $\prec$-better than t. Since τ is optimal, this is the worst possible path from v_2 to v. Hence the new loop over v and v_2 has t' as the $<$-largest tint, so the loop tint has improved.

2. If $P_1 = P_2$, consider two cases.

(a) Before the switch, v was on the loop over t . This implies $P_1 = P_2 = \overline{0}$. Thus there is no path from v_2 to t or v with an odd-colored highest tint bigger than t. This means that the loop after the switch will have a major $t' \leq t$.
   (i) If $t' < t$, assume towards a contradiction that t is even. Then by attractiveness $p_2 < p_1$, so no 1-optimal path from v_2 to t passes through v. In other words, any 1-optimal path from v_2 to v passes through t. Thus the loop tint t' after the switch is t, a contradiction. So t is odd, hence $t' \succ t$ and we are done.
   (ii) If $t = t'$, then the path length will improve after the switch, since the trace taken from v_2 to t achieving (P_2, p_2) is 1-optimal. By the attractiveness of the switch combined with clause 2 of Definition 9.14, we must have $t \neq v$. This in turn implies that the improved path length will be reflected in the value of v (recall that path lengths are not reflected in loop majors).

(b) If v was not on the loop over t before the switch, then $p_1 < p_2$ so $c(t)$ is odd. Also, by attractiveness there can be no path from v_2 to v with a highest tint t' such that $c(t')$ is odd and $t' > t$. Therefore, the loop tint after the switch is smaller than or equal to t. If it is smaller, then it is also better, and we are done. If it is equal, then the path value in v_2 after the switch will be (P_2, p_2) or better; any worse path from v_2 to t would have been taken before the switch. Thus by Lemma 9.16 the value will improve in v.

Subcase 2.2. Suppose that no optimal counterstrategy before the switch gives a trace from v_2 that passes through v. Thus, the path P_2 from v_2 to t under (σ, τ) does not pass through v, and we have the situation depicted in Figure 5 (where the round vertex v belongs to Player 0, square vertices v_1 and v_2 may belong to any player, P_2' is a path from v_2 through v to the loop with major tint t).

In the sequel, we abuse notation and identify the paths P_1, P_2, P_2', P_2'' with their color hit records. From attractiveness and optimality conditions we derive:

$$(t, P_1, p_1) \prec (t, P_2, p_2) \preceq (t, P_2', p_2') \equiv (t, P_1 + P_2'', p_1 + p_2''),$$

where (P_2'', p_2'') is the path from v_2 to v. The first inequality above follows from attractiveness, the second says that Player 1 did not take the path through v before the switch.

By transitivity, $(t, P_1, p_1) \prec (t, P_1 + P_2'', p_1 + p_2'')$. By definition of $\prec$ it follows that:

- either P_2'' contains a major color from $k, ..., c(t)$ winning for Player 0 and corresponding to a 0-better tint than t,

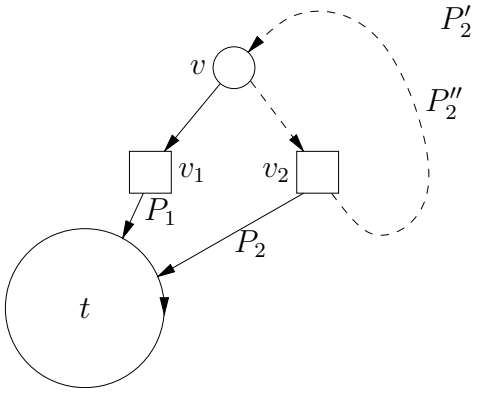

Figure 5. Illustration to case 2.2 of the proof that attractiveness implies profitability.

- or P_2'' does not contain a color from $k, \ldots, c(t)$ (note that $c(t)$ is odd in this case, since $p_2'' > 0$).

In both cases, the new loop through v_2 and v improves the loop value of v after the switch. This finishes the proof of the first claim.

Proof for property 1 of Definition 9.18. Consider any vertex $w \neq v$, and a 1-optimal trace from w under σ'. If the trace does not contain v it would be possible under σ as well, and thus $\nu_\sigma(w) \preceq \nu_{\sigma'}(w)$. Assume the trace contains v.

Let t and t' be the loop values of w under σ and σ', respectively. We first prove that $t' \succeq t$. Since the trace from w reaches v after the switch, they have the same loop value, which is better than or equal to the loop value of v before the switch (as shown above), which is better than or equal to t (since v was reachable from w before the switch).

We now prove that the path value of w does not decrease with the switch. If $t' = \mathbf{t}(w)$ then the path value is constant $(\overline{0}, 0)$ so either it did not change with the switch or the loop tint changed, hence improved, with the switch. Assume $t' \neq \mathbf{t}(w)$ and consider the path P taken in the optimal counterstrategy of Player 1 from w to t against σ'. If P does not contain v (i.e., v occurs later than t on the loop) then it was possible under σ as well, so the 1-optimal path before the switch was at least as bad as P. If P contains v, consider the initial segment of P leading up to v. This segment was possible under σ, so Lemma 9.16 implies that the 1-optimal path before the switch was at least as bad as P.

It follows that $\nu_\sigma(w) \preceq \nu_{\sigma'}(w)$. □

9.3 Stability Implies Global Optimality

In this section we show that iterative improvement can terminate once a stable strategy with no attractive switches is found. In more general terms, every local optimum is global. This is one of the main motivations for the complex strategy evaluation definitions.

The theorem below states that any stable strategy has a value that is better than or equal to the values of all other strategies. This means that any stable strategy is a *global optimum* of EVALUATE, which is crucial in proving that EVALUATE is CLG. As a consequence of the theorem, we also derive that any stable strategy is *winning* from all vertices in the winning set of Player 0 (Corollary 9.22), which means that it is a solution to the parity game. This ensures correctness of the algorithms.

Theorem 9.21 (Optimality) *Let σ be a stable strategy of Player 0. Then for all strategies σ' of Player 0 and all vertices v of the game, $\nu_\sigma(v) \succeq \nu_{\sigma'}(v)$.*

Proof. Let σ be a strategy with no attractive switches with respect to $\tau \equiv \tau(\sigma)$, an optimal counterstrategy of Player 1. We will show that no strategy σ' of Player 0 can improve the value of any vertex, even if Player 1 fixes and continues to invariably use his strategy τ.

Suppose, towards a contradiction, that there is a strategy σ' and a vertex w_0 for which $\nu_{\sigma,\tau}(w_0) \prec \nu_{\sigma',\tau}(w_0)$. Figure 6 depicts this situation, where round vertices belong to Player 0 and square vertices may belong to any player, bold paths are parts of the traces determined by (σ, τ) where σ and σ' coincide, dashed edges are taken only by σ' and normal edges are taken only by σ. Also, (r_i, C_i, c_i) and (t, B_i, b_i) are values of vertices, whereas (D_i, d_i) and (P_i, p_i) are path values, all computed with respect to (σ, τ). In particular, the loop shown in Figure 6 is determined by the trace from w_0 under the pair of strategies (σ', τ). Denote this loop λ.

We derive the desired contradiction in several steps.

1. First, we claim that all loop values of vertices u_i with respect to (σ, τ) *coincide* (call this value t, as shown in the picture). Indeed, suppose $u_{(j+1 \mod m)}$ has a 0-better loop value than u_j (such a pair should exist if not all loop values of u_i's are equal). Then switching from σ in u_j is attractive, contradicting the assumption that σ is stable with respect to τ.

2. Second, we show with a similar argument that $r_1 \succeq r_2 \succeq \cdots \succeq r_q \succeq r_{q+1} \equiv$

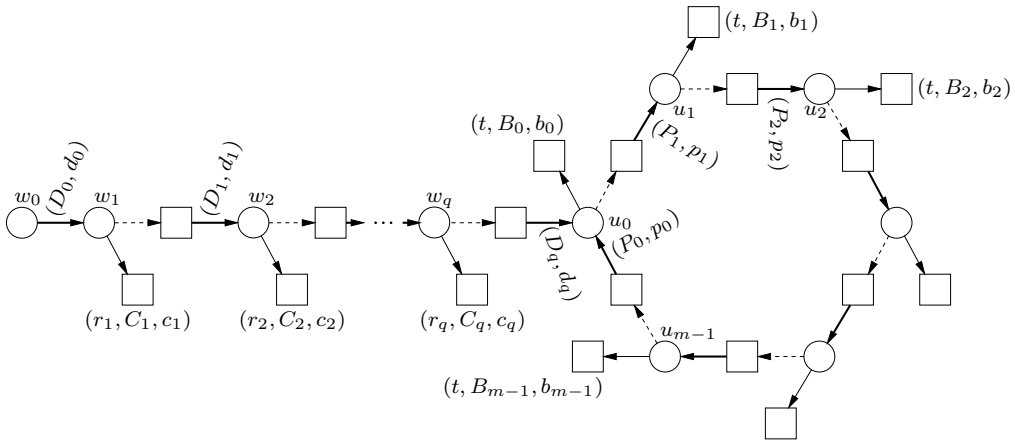

Figure 6. Illustration to the proof that stable strategies are optimal.

t. Suppose not, and $r_i \prec r_{i+1}$ for some i. Then the switch in w_i is attractive, contradicting the assumption that σ is stable with respect to τ.

3. Now we prove that the vertex of tint t actually belongs to λ. Suppose not. Then the major tint t' on λ must satisfy $t' \succ t$: by assumption $\nu_{\sigma',\tau}(w_0) \succeq \nu_{\sigma,\tau}(w_0)$, so $t' \succeq r_1$, and we showed that $r_1 \succeq t$. Since none of the switches in u_i (from a normal to a dashed arrow) is attractive, we derive the following system of inequalities, one for each u_i:

$$\begin{aligned} (B_0, b_0) &\succeq_t (P_1 + B_1, p_1 + b_1) \\ (B_1, b_1) &\succeq_t (P_2 + B_2, p_2 + b_2) \\ &\dots \\ (B_{m-1}, b_{m-1}) &\succeq_t (P_0 + B_0, p_0 + b_0) \end{aligned} \tag{12}$$

Lemma 9.16 applied to this system of inequalities implies

$$(B_0, b_0) \succeq_t (P_0 + P_1 + \dots + P_{m-1} + B_0, p_0 + p_1 + \dots + p_{m-1} + b_0) \tag{13}$$

By Lemma 9.17, (13) is a contradiction, because the largest tint represented in $P_0 + P_1 + \dots + P_{m-1}$ is t', and $t' \succ t$. Thus t is on λ.

4. We now proceed to proving that $t = t'$, where t' again is the loop major on λ. Suppose not. Then we know that $t < t'$ (numerical comparison), since t' is the major tint on λ. This also implies that $c(t')$ is even since $t' \succ t$. Assume without loss of generality that t belongs to the segment of λ between the successor of u_{m-1} and u_0 (we do not lose generality because the argument does not deal with the initial path through w_i's). Let (P'_0, p'_0) be the path value in the vertex succeeding u_{m-1} on λ. We can now build a system of inequalities similar to (12):

$$
\begin{aligned}
(B_0, b_0) &\succeq_t (P_1 + B_1, p_1 + b_1) \\
(B_1, b_1) &\succeq_t (P_2 + B_2, p_2 + b_2) \\
&\dots \\
(B_{m-1}, b_{m-1}) &\succeq_t (P'_0, p'_0)
\end{aligned} \tag{14}
$$

Again applying Lemma 9.16 gives

$$
(B_0, b_0) \succeq_t (P_1 + \dots + P_{m-1} + P'_0, p_1 + \dots + p_{m-1} + p'_0) \tag{15}
$$

The vertex succeeding u_0 under σ is part of the loop over t under σ and thus B_0 must be the all-zero vector. If t' belongs to the part of λ represented by $(P_1 + \dots + P_{m-1} + P'_0, p_1 + \dots + p_{m-1} + p'_0)$, then (15) is a contradiction, since t' would be the highest tint on this segment and $c(t')$ is even. Also, if t' instead lies between t and u_0 on λ, we get a contradiction since then t' and not t would be the major tint on the loop under σ. Thus $t = t'$.

5. We proceed to demonstrating that $t = t' = r_i$, for all i. Indeed, since $\nu_{\sigma,\tau}(w_0) \prec \nu_{\sigma',\tau}(w_0)$, we must have $t' \succeq r_1$. Together with $t = t'$, shown above, and the fact that the sequence $r_1, r_2, ..., r_{q+1} \equiv t$ is nonincreasing, we obtain $t = t' = r_i$, for all i.

6. Now we finally show the contradiction. We know that all paths from all w_i's and u_j's, both with respect to (σ, τ) and (σ', τ), lead to the *same* major vertex t on λ. Let u_{i-1} be the last vertex on the path from w_0 to t under (σ', τ) where σ and σ' diverge. Let (P'_i, p'_i) be the path value of its successor with respect to (σ, τ).

On the one hand, from the assumption that (σ', τ) gives a better value for w_0 we obtain:

$$
\begin{aligned}
&(D_0 + C_1, d_0 + c_1) \prec_t \\
&(D_0 + D_1 + \dots + D_q + P_1 + \dots + P'_i, d_0 + d_1 \dots + d_q + p_1 + \dots + p'_i)
\end{aligned} \tag{16}
$$

On the other hand, from non-attractiveness of switches, we have

$$
\begin{aligned}
(C_1, c_1) &\succeq_t (D_1 + C_2, d_1 + c_2) \\
(C_2, c_2) &\succeq_t (D_2 + C_3, d_2 + c_3) \\
(C_3, c_3) &\succeq_t (D_3 + C_4, d_3 + c_4) \\
&\dots \\
(C_q, c_q) &\succeq_t (D_q + B_0, d_q + b_0) \\
(B_0, b_0) &\succeq_t (P_1 + B_1, p_1 + b_1) \\
(B_1, b_1) &\succeq_t (P_2 + B_2, p_2 + b_2) \\
&\dots \\
(B_{i-2}, b_{i-2}) &\succeq_t (P_{i-1} + B_{i-1}, p_{i-1} + b_{i-1}) \\
(B_{i-1}, b_{i-1}) &\succeq_t (P'_i, p'_i)
\end{aligned}
$$

Applying Lemma 9.16 to this system, we derive:

$$(C_1, c_1) \succeq_t (D_1 + \cdots + D_q + P_1 + \cdots + P'_i, d_1 + \cdots + d_q + p_1 + \cdots + p'_i)$$

Applying Lemma 9.16 once again to the last inequality, we get:

$$(D_0 + C_1, d_0 + c_1) \succeq_t (D_0 + \cdots + D_q + P_1 + \cdots + P'_i, d_0 + \cdots + d_q + p_1 + \cdots + p'_i),$$

which contradicts (16). This finishes the proof. □

As consequences of Theorem 9.21, the strategy evaluation function presented in Section 9.1 has the remaining desired property stated in the beginning of Section 9:

Corollary 9.22 (Stable Strategies are Winning) *Any stable strategy of Player 0 is also* winning *from all vertices in the winning set of Player 0 and has a value that is at least as good as that of any other strategy.*

Proof. Let σ be a stable strategy. Theorem 9.21 states that it is at least as good as all other strategies. Moreover, all vertices with 0-winning loop values under σ are winning for Player 0, since the traces from them under σ and an optimal counterstrategy of Player 1 lead to 0-winning loops. Also, no vertex with a 1-winning loop value with respect to σ can be winning for Player 0, since none of his other strategies gives them a better value. □

This means that as soon as a stable strategy is found, the iterative improvement algorithms may stop.

From Theorem 9.21, we can also derive the equivalence of profitability and attractiveness.

Corollary 9.23 *Any profitable switch is also attractive.*

Proof. Suppose that a switch in vertex v from σ to σ' is profitable but not attractive. Remove from G all edges leaving vertices of Player 0 except those used by σ and σ'. In the resulting graph, σ is stable, and thus by Theorem 9.21, the other strategy σ' cannot have a better value. □

This result shows that only looking at attractive switches is no restriction compared to actually evaluating all neighboring strategies (all strategies obtainable by a single switch from the current strategy).

Another consequence of Theorem 9.21 is that any pair of strategies that differ only in one vertex have comparable values. This will be important for showing that Evaluate is a CLG-function in Section 10.

Corollary 9.24 *For any parity game G and any pair of strategies σ and σ' of Player 0 in G that differ only in one vertex, either $\sigma \preceq \sigma'$ or $\sigma' \preceq \sigma$.*

Proof. Consider the subgame of G where all edges from vertices in V_0 that are not used by σ or σ' are removed. The values of σ and σ' cannot be incomparable, since this would mean that both strategies are local maxima in the subgame and thus should have the same value by Theorem 9.21. □

As a corollary to Theorems 9.20, 9.21 and Corollary 9.23 we derive the following

Corollary 9.25 *The* EVALUATE *function of any parity game is RLG.* □

In Section 10 we actually prove that it is CLG.

9.4 Complexity: Depth of the Measure Space

In this section we provide an upper bound on the maximal number of improvement steps of any iterative improvement algorithm using our definition of strategy values.

Theorem 9.26 *The strategy evaluation function allows for at most* $O(n_0 \cdot n^2 \cdot (n/k+1)^k)$ *profitable switches.*

Proof. For each vertex there are $\prod_{i=1}^{k}(|V^i|+1)$ possible color hit records. This expression takes its maximum when there are equally many vertices of each color. Thus at most $(n/k+1)^k$ color hit records are possible. There are also n possible loop values and n possible path lengths. Thus each vertex can improve its value at most $n^2(n/k+1)^k$ times. In each switch, at least one of Player 0's n_0 vertices improves its value, so we get an $O(n_0 \cdot n^2 \cdot (n/k+1)^k)$ bound on the number of profitable single switches. □

9.5 Computing Optimal Counterstrategies

In this section we present an algorithm for computing the EVALUATE function for any parity game. Since values of strategies of Player 0 are defined as minima over all counterstrategies of Player 1, the algorithm implicitly computes optimal counterstrategies. A counterstrategy τ is *optimal* against σ if for every vertex v in the game and every strategy τ' of Player 1, $\nu_{\sigma,\tau}(v) \preceq \nu_{\sigma,\tau'}(v)$. The correctness proof for the algorithm shows that there is at least one counterstrategy that achieves the best possible value from every vertex. We also show that the algorithm has polynomial time complexity.

To compute the values under a strategy σ, partition vertices of G_σ into classes L_t containing the vertices from which Player 1 can ensure the loop tint t,

but cannot guarantee any worse loop tint. This can be done by using finite reachability in G_σ as follows. For each tint t in $\prec$-ascending order, check whether t can be reached from itself without passing any tint $t' > t$. If so, Player 1 can form a loop with t as major. Since the tints are considered in $\prec$-ascending order, t will be the best loop value Player 1 can achieve for all vertices from which t is reachable. Remove them from the graph, place them in class L_t, and proceed with the next tint.

In a class L_t there may be cycles that do not contain t. As the following property shows, such loops must, however, have majors that are 0-better than t. This is important for the correctness of the algorithm.

Property 9.27 In every class L_t the following holds. If a tint t' is the biggest tint on a cycle, then $t \prec t'$. □

Once the partition into classes L_t has been accomplished, the loop value of each vertex is known and we can proceed to computing the path values. For each class L_t, Algorithm 5 below uses dynamic programming to calculate the values of 1-optimal paths of different lengths from each vertex to t. For each vertex, the algorithm first computes the optimal color hit record (abbreviated *chr* in the algorithm) over all paths of length 0 to the loop major (∞ for each vertex except t). Then it calculates the color hit record of optimal paths of length one, length two, and so forth. It uses the values from previous steps in each step except the initial one.

Algorithm 5: Computing path values within a class L_t.

PATH-VALUES(L_t)

(1) $t.chr[0] \leftarrow (0, \ldots, 0)$
(2) **foreach** vertex $v \in L_t$ except t
(3) $v.chr[0] \leftarrow \infty$
(4) **for** $i \leftarrow 1$ **to** $|L_t| - 1$
(5) $t.chr[i] \leftarrow \infty$
(6) **foreach** vertex $v \in L_t$ except t
(7) $v.chr[i] \leftarrow \min_{\prec_t}\{\text{ADD-COLOR}(t,\, v'.chr[i-1],\, \mathbf{t}(v)) :$
$v' \in L_t$ is a successor of $v\}$
(8) $t.pathvalue \leftarrow ((0, \ldots, 0), 0)$
(9) **foreach** vertex $v \in L_t$ except t
(10) $v.pathvalue \leftarrow \min_{\prec_t}\{(v.chr[i], i) : 0 \leq i < |L_t|\}$

The function ADD-COLOR takes a tint, a color hit record, and a second tint. If the second tint is bigger than the first one, then ADD-COLOR increases the position in the vector representing the color of the second tint. The function always returns ∞ when the second argument has value ∞.

Lemma 9.28 (Properties of Algorithm 5) *Algorithm 5 has the following properties:*

(1) It correctly computes values of 1-optimal paths.
(2) Optimal paths are simple.
(3) The values computed are consistent with an actual positional strategy that guarantees loop value t.

Proof. First we prove clause 2 of the Lemma. Observe that following a path to t with a cycle can never yield a worse value than following the same path, but without traversing the cycle. This is a consequence of Lemma 9.27. If all tints t' on the cycle satisfy $t' < t$, then $c(t)$ must be odd, and since traversing the loop contributes nothing to the color hit record the shorter path will be better for Player 1. If on the other hand the major tint t' on the cycle satisfies $t < t'$, then $c(t')$ must be even and traversing the cycle will give a better color hit record. This proves that 1-optimal paths are simple.

Now we move on to proving clause 1. Consider the class C of all paths p from all vertices in L_t to the major vertex t. We prove by induction on the length of such paths, that for each vertex v the algorithm actually computes the values of the 1-optimal paths of each length from v to t. Note that these paths may be non-simple. Cycles can be traversed a finite number of times (since the algorithm makes at most $|L_t| - 1$ iterations). This is not a contradiction to clause 2, since the computed paths are not necessarily 1-optimal, only the best ones of a particular length.

Base: Path length = 0. There is just one path of length 0 in C (from t to itself) and obviously the algorithm correctly computes the optimal value in this case.

Inductive Case. Assume that the values computed for paths of all lengths $j < i$ are the values of the 1-optimal length j paths from each vertex to the major t. Consider any vertex v. If no successor of v has a value other than ∞ for paths of length $i-1$ there cannot be a path of length i from v to the loop major, and the value $v.chr[i]$ is correctly set to ∞.

If, on the other hand, at least one successor of v has a value for paths of length $i-1$, then consider any successor v' of v. By Lemma 9.16 any optimal path from v directly via v' coincides after the first step with the optimal path from v'. Thus the minimum calculated on line (7) is the correct value for $v.chr[i]$.

This finishes the proof of algorithm correctness. Finally, to prove clause 3 we first observe that the algorithm maintains the following property:

> *If a value (P, p) has been computed for a vertex v other than t, then v has a successor v' for which a value $(P', p-1)$ has been computed, such that*

$P - P'$ is either the vector with all zeros, or it has all zeros except a one in the position for color $c(v)$.

Thus from every vertex except t we can select one outgoing edge (positional strategy) providing for the optimal value.

It remains to select one edge leaving t to complete the description of a positional strategy providing for optimal path values and loop value t. From at least one of the successors of the major t, there must be a path to t on which no tint t' such that $t < t'$ appears. Furthermore, by Property 9.27, no successor of t has a path to t with biggest tint t' such that $t' \prec t$ and $t < t'$. Thus there is at least one successor v of t such that the 1-optimal path from v to t contains no even color bigger than $c(t)$. By going from t to the successor with the worst value Player 1 can therefore ensure that the loop tint for all vertices in L_t will be t. Thus there is a positional strategy corresponding to the path values computed by the algorithm. □

Lemma 9.29 (Complexity of Algorithm 5) *The algorithm for computing* Evaluate *runs in time $O(|V| \cdot |E| \cdot k)$, where $|V|$ is the number of vertices of the graph, $|E|$ is the number of edges, and k is the number of colors.*

Proof. Calculating the classes L_t is straightforwardly done in time $O(|V| \cdot |E|)$. The bottleneck in each call to Path-Values is the double loop on lines (4) to (7). Taking the $\prec_t$-minimum over all successors of v requires one $\prec_t$-comparison per edge leaving v, and each $\prec_t$-comparison requires $O(k)$ integer comparisons. Thus the loop on line (6) needs $O(|E| \cdot k)$ comparisons. It is executed $\sum_t(|L_t| - 1) = O(|V|)$ times altogether (counting all calls to Path-Values), so the total running time is $O(|V| \cdot |E| \cdot k)$. □

10 Parity and Simple Stochastic Games are CLG

In Sections 5 and 6 we described a variety of algorithms that can be used to optimize CLG-functions. We now show that they are all applicable to solving parity and simple stochastic games, by proving that the corresponding strategy evaluation functions, defined in Sections 9 and 7.1, respectively, are in fact CLG.

For parity games, this in particular implies that Evaluate allows for multi-switching algorithms: making any number of *simultaneous* attractive switches gives a strategy with a better value.

For simple stochastic games, we already know that multi-switching works. In fact, we will rely on this when proving that the measure is CLG. The result in both cases shows that all the structural properties of CLG-functions apply

to the strategy spaces together with the evaluation functions. In particular we can express the games in terms of CU-functions, LP-type problems, or AOP optimization.

10.1 Parity Games are CLG

For each vertex v in a parity game, let δ_v be the number of edges leaving v. Recall that the space of strategies for Player 0 in a parity game with $|V_0| = n_0$ and $\sum_{v \in V_0} \delta_v = m$ is an n_0-dimensional hyperstructure with m facets.

Theorem 10.1 *Let $\mathcal{P}$ be the hyperstructure isomorphic to the strategy space of a parity game G. Then the corresponding strategy evaluation function* Evaluate *on $\mathcal{P}$ is CLG.*

Proof. By Theorem 4.3, it is enough to show that the restriction of Evaluate to any two-dimensional *subcube* of $\mathcal{P}$ is CLG. This is the same as saying that for any restriction of G to a game G_2 in which Player 0 has choices in only two vertices, and has exactly two choices in each of them, Evaluate has the following properties on the two-dimensional hypercube $\mathcal{H}$ corresponding to the four possible strategies:

(1) all neighbors have comparable values;
(2) every local maximum is also global;
(3) every local minimum is also global;
(4) every pair of local minima are connected by a path of local minima.

Property (1) follows from Corollary 9.24, and (2) from Theorem 9.21 and Corollary 9.23. We now prove property (3), arguing by symmetry. Let σ_0 and σ_1 be two strategies of Player 0 that are local minima. Let G_2' be the game G_2 except all vertices belong to Player 1, and consider optimizing Player 1's strategy rather than Player 0's. Let τ_0 and τ_1 be the strategies of Player 1 in G_2' that coincide with σ_0 and σ_1 in vertices that were Player 0's in G_2, and coincide with the optimal response strategies of Player 1 to σ_0 and σ_1 in other vertices. Now there are no attractive switches for Player 1: the switches in vertices that were Player 0's are non-attractive since σ_0 and σ_1 are assumed to be local minima, and the switches in other vertices are non-attractive since the strategy in those vertices is optimal for Player 1. Thus both τ_0 and τ_1 are optimal (for Player 1), so they have the same value. It now suffices to note that the values of σ_0 and σ_1 in G_2 are identical to the values of τ_0 and τ_1 in G_2', respectively. This shows (3).

It remains to prove (4). Assume, on the contrary, that two strategies σ_0 and σ_1 are minimal but there is no path on $\mathcal{H}$ of strategies with identical values

between them. Then they are on the diagonal, have the same values, and the other strategies have bigger values.

Let a and b be the vertices where Player 0 has choices. For $x, y \in \{0, 1\}$, denote by $[x, y]$ the strategy corresponding to corner (x, y) on the two-dimensional face. We may without loss of generality assume that $[0, 0]$ and $[1, 1]$ have minimal values while $[1, 0]$ and $[0, 1]$ have non-minimal values (the case when $[1, 0]$ and $[0, 1]$ are minimal is symmetric). Note that both $[0, 1]$ and $[1, 0]$ are local maxima, so they are also global and have the same value. See Figure 3.

$$\begin{array}{ccc} [0,1] & \succ & [1,1] \\ \curlyvee & & \curlywedge \\ [0,0] & \prec & [1,0] \end{array}$$

Figure 7. $[1, 0]$ and $[0, 1]$ are best, the other strategies are worst

Denote by $\nu_\sigma(v)$ the value of vertex v under strategy σ. Let a^+, b^+ be the vertex values of a and b under the strategies $[0, 1]$ and $[1, 0]$ and let a^-, b^- be the values under $[0, 0]$ and $[1, 1]$. Since the switch from $[0, 0]$ to $[1, 0]$ improves the value in a we have $a^- \prec a^+$, and since the switch from $[0, 0]$ to $[0, 1]$ improves the value in b we also have $b^- \prec b^+$.

Note that Player 1 in his optimal response strategy against $[0, 0]$ reaches a from b; if not, the same response strategy would be possible against $[1, 0]$, resulting in $b^+ \preceq b^-$, a contradiction. Symmetric arguments show that b reaches a under strategy $[1, 1]$ and that a reaches b under both strategies $[0, 0]$ and $[1, 1]$. In other words, a and b are on the same loop under strategies $[0, 0]$ and $[1, 1]$.

Let t be the common loop tint of a^- and b^-. Consider four cases depending on whether t coincides with a or b, or is on the path from a to b or from b to a against strategy $[0, 0]$.

Case 1: $a = t$**.** Construct a play starting in b against $[0, 1]$ as follows. First follow the same path as against $[1, 1]$ until we reach a. From a, follow the same path as against $[0, 0]$ until we either reach b or interfere with the choices already made. See Figure 8. This gives a loop over t with the same path from b to t as in $[1, 1]$. Thus $\nu_{[0,1]}(b) \preceq \nu_{[1,1]}(b)$, contradicting our result that $\nu_{[0,1]}(b) = b^+ \succ b^- = \nu_{[1,1]}(b)$.

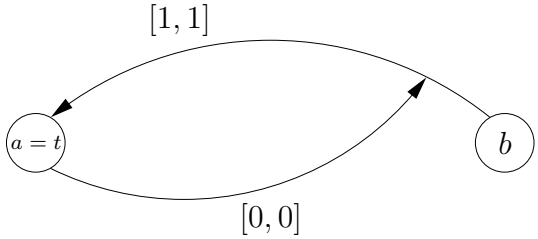

Figure 8. If $a = t$ (Case 1), then Player 1 can achieve the same value in b under strategy $[0, 1]$ as under $[1, 1]$.

Case 2: $b = t$**.** The proof is symmetric to Case 1.

Case 3: t **is on the path from** a **to** b **under** $[0, 0]$**.** We now construct an auxiliary play against strategy $[0, 1]$ looping over t both from a and from b. First select the same path from a to t as against $[0, 0]$. Then follow the path from b to a as against $[1, 1]$ until either a or another vertex on the constructed path from a to t is reached. Finally extend the path from t to b as against $[0, 0]$, until either b or any other vertex on our constructed path is reached. See Figure 9.

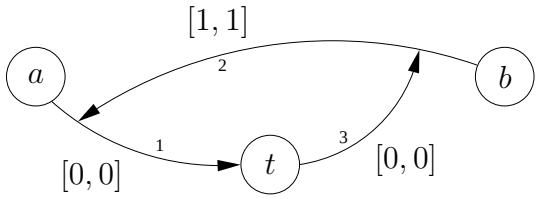

Figure 9. If t is on the path from a to b under $[0, 0]$ (Case 3), then Player 1 can achieve the same value in a under strategy $[0, 1]$ as under $[0, 0]$.

Since the constructed path can be chosen by Player 1 against $[0, 1]$ and it gives the value a^- in a, we must have $a^+ = \nu_{[0,1]}(a) \preceq \nu_{[0,0]}(a) = a^-$. But this contradicts our earlier conclusion that $a^+ \succ a^-$.

Case 4: t **is on the path from** b **to** a **under** $[0, 0]$**.** The proof is symmetric to case 3.

This concludes the proof. □

The evaluation function of Vöge and Jurdziński [46] can also be shown to be CLG. An analog of Theorem 10.1 also holds for simple stochastic games. Its proof will be sketched in Section 10.2.

10.2 Simple Stochastic Games are CLG

We now prove that the strategy improvement functions for SSGs that halt with probability 1 are CLG. We rely on the following important properties of the strategy improvement function for SSGs [30,16,12,13].

Theorem 10.2 *In any simple stochastic game, the strategy evaluation function satisfies the following properties:*

(1) any locally optimal strategy (having no better neighbors) is globally optimal (and this holds on every subgame);

(2) multi-switching (changing the strategy in many coordinates where single-switching gives an improvement) gives an improvement. □

This is enough to prove that the value functions of SSGs are CLG.

Theorem 10.3 *Let $\mathcal{P}$ be the hyperstructure isomorphic to the strategy space of a simple stochastic game. Then the corresponding strategy evaluation function on $\mathcal{P}$ is CLG.*

Proof. As in the proof that parity games are CLG, we show that the requirements of Theorem 4.8 hold on every 2-dimensional subcube of the strategy space:

(1) all neighbors have comparable values;
(2) every local maximum is also global;
(3) every local minimum is also global;
(4) every pair of local minima is connected by a path of local minima.

Property (1) holds because values of strategies in a simple stochastic game are real numbers. By Theorem 10.2, any local maximum is global. It remains to prove properties (3) and (4). The proof that local minima are global is identical to the proof for parity games, so we omit restating it. To prove (4), consider two strategies σ and σ' that are local minima on a 2-dimensional subcube and assume they are not connected by a path of local minima. Thus they have identical values, are diagonally opposite on the subcube, and the other two corners of the cube have bigger values. Hence both switches in σ give better values, so making both switches should also give a better value by Theorem 10.2. This is a contradiction, since such a multi-switch would end up in σ' which has the same value as σ. □

As a corollary, the subexponential Kalai-style and Matoušek–Sharir–Welzl-Style algorithms for the general RLG-optimization problem (see Section 5) apply and give subexponential decision algorithms for general simple stochastic games of arbitrary outdegree. Recall that the previous subexponential algorithm due to Ludwig [36] applies only to binary SSGs and becomes exponential whenever a game of unbounded outdegree is first reduced to a binary one, with a quadratic blow-up in the number of vertices.

11 Application: Two Structural Complexity Results

As an application of the theory developed in previous sections, we show how two results on structural complexity follow as corollaries.

11.1 Non-PLS-Completeness of Parity and Simple Stochastic Games

As mentioned in the introduction, the complexity class PLS (Polynomial Time Local Search) [32,50] intuitively captures local search problems where the next feasible solution is taken from a polynomially big neighborhood. It is immediate that CLG optimization using a single-switching algorithm is in PLS, provided the function optimized is polynomial time computable. Yannakakis [50] showed PLS-completeness of many natural problems and asked whether simple stochastic games are PLS-complete. Luckily, the following theorem gives a negative answer (also for other problems we consider). This is good news, since tight PLS-complete problems should have exponential lower bounds on improving paths to the optimum for any pivoting rules, including randomized ones and rules that may not be implementable in polynomial time.

Theorem 11.1 *The problems of optimizing CU-, CLG-, RLG-functions, as well as finding optimal strategies in parity and simple stochastic games (all with the standard 1-flip neighborhood structure) are not complete with respect to the tight PLS-reductions.*

Proof. This follows from [50, Lemmas 11, 12, Theorem 13], and our Theorem 4.4 (and Corollary afterwards), hence by Theorems 10.1 and 10.3 for parity and simple stochastic games. □

11.2 UP ∩ coUP*-Membership*

Jurdziński [33] showed that parity games is in the complexity class UP ∩ coUP. Recall that the decision problem for Parity Games consists in deciding whether Player 0 wins a parity game starting from a given vertex. UP is the subclass of NP where it is possible to use unique membership certificates, and coUP is is the corresponding subclass of coNP with unique non-membership certificates. Jurdziński [33] uses a reduction via mean payoff games to discounted mean payoff games, and a characterization of values in discounted mean payoff games as unique solutions to certain equations, to show UP ∩ coUP-membership. We note that the reduction from parity games via CLG-functions to CU-functions gives another proof of this fact. As a certificate we take the global maximum of the CU-function with strategies as the domain and strategy values (reduced to CU) as the codomain. By the definition of CU-function, this maximum is unique. Also, the strategy is clearly polynomial in size. By Theorem 9.21, we can verify in polynomial time that this is a global maximum, by computing the measure using Algorithm 5 and testing whether all switches are non-attractive. By Corollary 9.22, we only need to look at the measure to determine the winner. Thus, if Player 0 wins, this is a certificate of UP-membership, and if Player 1 wins it is a certificate

of coUP-membership. This shows both UP- and coUP-membership of the Parity Games decision problem.

12 Conclusions

We combined ideas from three areas: game theory, discrete optimization on hypercubes, and linear programming, to obtain new results in the two former domains, mainly regarding complexity.

After defining the optimization problem for RLG-functions, CLG-functions and CU-functions on hyperstructures, we showed several important properties of such functions. They can all be optimized in subexponential time, by adapting algorithms from linear programming by Kalai [35], Matoušek–Sharir-Welzl [38], and from simple stochastic games by Ludwig [36]. The Ludwig-style algorithm is a specialization of the MSW-style algorithm, working on hypercubes rather than hyperstructures. CLG-functions and CU-functions allow for optimization algorithms that are intuitively more "aggressive", proceeding from a current iterate to next by "big" steps (multi-switching). Although no subexponential bounds are known for these algorithms, we settle a nontrivial exponential upper bound for the random multiple switch algorithm on CU-functions on hypercubes. We also prove that CLG-functions reduce to CU-functions. This is an important result because the optimization problem for CU-functions is already rather well studied [49,28,48,45,4,3] and because they possess additional properties compared to CLG-functions.

As an application and the main motivation for the study of the abstract problems above, we demonstrated that finding the winner in parity and simple stochastic games reduces to optimizing a CLG-function. The technique we use is iterative strategy improvement, earlier studied for simple stochastic games by Hoffman–Karp and later by Condon [30,12,13] and for parity games by Vöge–Jurdziński [46]. After assigning a value to each strategy, we identify strategies with vertices of a hyperstructure and show that the resulting function is CLG. For parity games we use a new value function [5], similar to but more economical than that in [46]. Thus all results for CLG-functions carry over to parity and simple stochastic games. In particular, we get upper bounds subexponential in the number of vertices of Player 0. As corollaries, we also obtain simple proofs of two results regarding the structural complexity of games, and a reduction from the games to the class of LP-type problems defined by Matoušek, Sharir, and Welzl [38].

We also defined a new discrete strategy evaluation function for mean payoff games, which allows for efficient iterative strategy improvement without the need to reduce to discounted payoff or simple stochastic games.

Acknowledgments. We are grateful to Leonid Khachiyan, Vladimir Gurvich, and Endre Boros for inspiring discussions and illuminating ideas.

References

[1] E. Aarts and J. K. Lenstra, editors. *Local Search in Combinatorial Optimization*. John Wiley & Sons, 1997.

[2] D. Aldous. Minimization algorithms and random walk on the d-cube. *The Annals of Probability*, 11(2):403–413, 1983.

[3] H. Björklund, S. Sandberg, and S. Vorobyov. An experimental study of algorithms for completely unimodal optimization. Technical Report 2002-030, Department of Information Technology, Uppsala University, October 2002. `http://www.it.uu.se/research/reports/`.

[4] H. Björklund, S. Sandberg, and S. Vorobyov. Optimization on completely unimodal hypercubes. Technical Report 2002-018, Uppsala University / Information Technology, May 2002. `http://www.it.uu.se/research/reports/`.

[5] H. Björklund, S. Sandberg, and S. Vorobyov. A discrete subexponential algorithm for parity games. In H. Alt and M. Habib, editors, *20th International Symposium on Theoretical Aspects of Computer Science, STACS'2003*, volume 2607 of *Lecture Notes in Computer Science*, pages 663–674, Berlin, 2003. Springer-Verlag. Full preliminary version: TR-2002-026, Department of Information Technology, Uppsala University, September 2002.

[6] H. Björklund, S. Sandberg, and S. Vorobyov. A discrete subexponential algorithm for parity games. In H. Alt and M. Habib, editors, *20th International Symposium on Theoretical Aspects of Computer Science, STACS'2003*, volume 2607 of *Lecture Notes in Computer Science*, pages 663–674, Berlin, 2003. Springer-Verlag.

[7] H. Björklund, S. Sandberg, and S. Vorobyov. On combinatorial structure and algorithms for parity games. Technical Report 2003-002, Department of Information Technology, Uppsala University, January 2003. `http://www.it.uu.se/research/reports/`.

[8] H. Bjorklund, S. Sandberg, and S. Vorobyov. A combinatorial strongly subexponential strategy improvement algorithm for mean payoff games. Technical Report DIMACS-2004-05, DIMACS: Center for Discrete Mathematics and Theoretical Computer Science, Rutgers University, NJ, March 2004. `http://dimacs.rutgers.edu/TechnicalReports/`.

[9] H. Björklund, S. Sandberg, and S. Vorobyov. Memoryless determinacy of parity and mean payoff games: A simple proof. *Theoretical Computer Science*, 310(1-3):365–378, January 2004.

[10] E. Boros and P. L. Hammer. Pseudo-boolean optimization. Technical Report RRR 48-2001, RUTCOR Rutger Center for Operations Research, 2001.

[11] E. M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 2000.

[12] A. Condon. The complexity of stochastic games. *Information and Computation*, 96:203–224, 1992.

[13] A. Condon. On algorithms for simple stochastic games. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 13:51–71, 1993.

[14] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. The MIT electrical engineering and computer science series. MIT Press, Cambridge, MA, 6th printing edition, 1992.

[15] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press and McGraw-Hill Book Company, Cambridge, MA, 2 edition, 2001.

[16] C. Derman. *Finite-State Markovian Decision Processes*. Academic Press, 1970.

[17] A. Ehrenfeucht and J. Mycielski. Positional games over a graph. *Notices Amer. Math Soc.*, 20:A–334, 1973.

[18] A. Ehrenfeucht and J. Mycielski. Positional strategies for mean payoff games. *International Journ. of Game Theory*, 8:109–113, 1979.

[19] E. A. Emerson. Model checking and the Mu-calculus. In N. Immerman and Ph. G. Kolaitis, editors, *DIMACS Series in Discrete Mathematics*, volume 31, pages 185–214, 1997.

[20] E. A. Emerson and C. S. Jutla. Tree automata, μ-calculus and determinacy. In *Annual IEEE Symp. on Foundations of Computer Science*, pages 368–377, 1991.

[21] J. Filar and K. Vrieze. *Competitive Markov Decision Processes*. Springer, 1996.

[22] B. Gärtner. A subexponential algorithm for abstract optimization problems. *SIAM Journal on Computing*, 24:1018–1035, 1995.

[23] B. Gärtner. Combinatorial linear programming: Geometry can help. In *RANDOM'98*, volume 1518 of *Lect. Notes Comput. Sci.*, pages 82–96, 1998.

[24] B. Gärtner. The random-facet simplex algorithm on combinatorial cubes. *Random Structures and Algorithms*, 20(3):353–381, 2002.

[25] Goldwasser. A survey of linear programming in randomized subexponential time. *SIGACTN: SIGACT News (ACM Special Interest Group on Automata and Computability Theory)*, 26:96–104, 1995.

[26] E. Grädel, W. Thomas, and T. Wilke, editors. *Automata Logics and Infinite Games. A Guide to Current Research*, volume 2500 of *Lecture Notes in Computer Science*. Springer-Verlag, 2003.

[27] V. A. Gurvich, A. V. Karzanov, and L. G. Khachiyan. Cyclic games and an algorithm to find minimax cycle means in directed graphs. *U.S.S.R. Computational Mathematics and Mathematical Physics*, 28(5):85–91, 1988.

[28] P. L. Hammer, B. Simeone, Th. M. Liebling, and D. De Werra. From linear separability to unimodality: a hierarchy of pseudo-boolean functions. *SIAM J. Disc. Math.*, 1(2):174–184, 1988.

[29] P. Hansen, B. Jaumard, and V. Mathon. Constrained nonlinear 0-1 programming (state-of-the-art survey). *ORSA Journal on Computing*, 5(2):97–119, 1993.

[30] A. J. Hoffman and R. M. Karp. On nonterminating stochastic games. *Management Science*, 12(5):359–370, 1966.

[31] R. A. Howard. *Dynamic Programming and Markov Processes*. M.I.T. Press, 1960.

[32] D. S. Johnson, C. H. Papadimitriou, and M. Yannakakis. How easy is local search? *Journal of Computer and System Sciences*, 37:79–100, 1988.

[33] M. Jurdziński. Deciding the winner in parity games is in UP∩co-UP. *Information Processing Letters*, 68:119–124, 1998.

[34] M. Jurdziński. Small progress measures for solving parity games. In H. Reichel and S. Tison, editors, *17th STACS*, volume 1770 of *Lect. Notes Comput. Sci.*, pages 290–301. Springer-Verlag, 2000.

[35] G. Kalai. A subexponential randomized simplex algorithm. In *24th ACM STOC*, pages 475–482, 1992.

[36] W. Ludwig. A subexponential randomized algorithm for the simple stochastic game problem. *Information and Computation*, 117:151–155, 1995.

[37] Y. Mansour and S. Singh. On the complexity of policy iteration. In *Uncertainty in Artificial Intelligence'99*, 1999.

[38] J. Matoušek, M. Sharir, and M. Welzl. A subexponential bound for linear programming. In *8th ACM Symp. on Computational Geometry*, pages 1–8, 1992.

[39] J. Matoušek, M. Sharir, and M. Welzl. A subexponential bound for linear programming. *Algorithmica*, 16:498–516, 1996.

[40] C. Papadimitriou. Algorithms, games, and the internet. In *ACM Annual Symposium on Theory of Computing*, pages 749–753. ACM, July 2001.

[41] V. Petersson and S. Vorobyov. Parity games: interior-point approach. Technical Report 008, Uppsala University / Information Technology, May 2001. `http://www.its.uu.se/research/reports/`.

[42] A. Puri. *Theory of hybrid systems and discrete events systems*. PhD thesis, EECS Univ. Berkeley, 1995.

[43] M. Sharir and E. Welzl. A combinatorial bound for linear programming and related problems. In *9th Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 577 of *Lecture Notes in Computer Science*, pages 569–579, Berlin, 1992. Springer-Verlag.

[44] C. A. Tovey. Low order polynomial bounds on the expected performance of local improvement algorithms. *Mathematical Programming*, 35:193–224, 1986.

[45] C. A. Tovey. Local improvement on discrete structures. In E. Aarts and Lenstra J. K., editors, *Local Search in Combinatorial Optimization*, pages 57–89. John Wiley & Sons, 1997.

[46] J. Vöge and M. Jurdziński. A discrete strategy improvement algorithm for solving parity games. In E. A. Emerson and A. P. Sistla, editors, *CAV'00: Computer-Aided Verification*, volume 1855 of *Lect. Notes Comput. Sci.*, pages 202–215. Springer-Verlag, 2000.

[47] J. Vöge and M. Jurdziński. A discrete strategy improvement algorithm for solving parity games. Technical Report RS-00-48, BRICS, Department of Computer Science, University of Aarhus, December 2000. `http://www.brics.dk/RS/00/48/`.

[48] D. Wiedemann. Unimodal set-functions. *Congressus Numerantium*, 50:165–169, 1985.

[49] K. Williamson Hoke. Completely unimodal numberings of a simple polytope. *Discrete Applied Mathematics*, 20:69–81, 1988.

[50] M. Yannakakis. Computational complexity. In E. Aarts and Lenstra J. K., editors, *Local Search in Combinatorial Optimization*, pages 19–55. John Wiley & Sons, 1997.

[51] U. Zwick and M. Paterson. The complexity of mean payoff games on graphs. *Theor. Comput. Sci.*, 158:343–359, 1996.

Recent technical reports from the Department of Information Technology

2003-057 Erik Berg and Erik Hagersten: *Low-Overhead Spatial and Temporal Data Locality Analysis*

2003-058 Erik Berg and Erik Hagersten: *StatCache: A Probabilistic Approach to Efficient and Accurate Data Locality Analysis*

2003-059 Jonas Persson and Lina von Sydow: *Pricing European Multi-asset Options Using a Space-time Adaptive FD-method*

2003-060 Pierre Flener: *Realism in Project-Based Software Engineering Courses: Rewards, Risks, and Recommendations*

2003-061 Lars Ferm and Per Lötstedt: *Space-Time Adaptive Solution of First Order PDEs*

2003-062 Emilio Tuosto, Björn Victor, and Kidane Yemane: *Polyadic History-Dependent Automata for the Fusion Calculus*

2003-063 Michael Baldamus, Joachim Parrow, and Björn Victor: *Spi Calculus Translated to π-Calculus Preserving May-Testing*

2003-064 Arnim Brüger, Bertil Gustafsson, Per Lötstedt, and Jonas Nilsson: *High Order Accurate Solution of the Incompressible Navier-Stokes Equations*

2003-065 Michael Baldamus, Richard Mayr, and Gerardo Schneider: *A Backward/Forward Strategy for Verifying Safety Properties of Infinite-State Systems*

2004-001 Torsten Söderström, Torbjörn Wigren, and Emad Abd-Elrady: *Maximum Likelihood Modeling of Orbits of Nonlinear ODEs*

2004-002 Pablo Giambiagi, Gerardo Schneider, and Frank D. Valencia: *On the Expressiveness of CCS-like Calculi*

2004-003 Johan Elf, Per Lötstedt, and Paul Sjöberg: *Problems of High Dimension in Molecular Biology*

2004-004 Torbjörn Wigren: *Recursive Prediction Error Identification of Nonlinear State Space Models*

2004-005 Håkan Zeffer, Zoran Radovic, Oskar Grenholm, and Erik Hagersten: *Evaluation, Implementation and Performance of Write Permission Caching in the DSZOOM System*

2004-006 Henrik Löf, Markus Nordén, and Sverker Holmgren: *Improving Geographical Locality of Data for Shared Memory Implementations of PDE Solvers*

2004-007 Erik Nordström, Per Gunningberg, and Christian Tschudin: *Comparison of Gateway Forwarding Strategies in Ad hoc Networks*

2004-008 Pär Samuelsson and Bengt Carlsson: *An Integrating Linearization Method for Static Input Nonlinearities*

2004-009 Henrik Johansson and Johan Steensland: *A Characterization of a Hybrid and Dynamic Partitioner for SAMR Applications*

2004-010 Neil Ghani, Kidane Yemane, and Björn Victor: *Relationally Staged Computations in Calculi of Mobile Processes*

2004-011 Henrik Björklund, Sven Sandberg, and Sergei Vorobyov: *Randomized Subexponential Algorithms for Infinite Games*

UPPSALA
UNIVERSITET

April 2004
ISSN 1404-3203
http://www.it.uu.se/