Non-oblivious Strategy Improvement

John Fearnley

Department of Computer Science, University of Warwick, UK
john@dcs.warwick.ac.uk

Abstract. We study strategy improvement algorithms for mean-payoff and parity games. We describe a structural property of these games, and we show that these structures can affect the behaviour of strategy improvement. We show how awareness of these structures can be used to accelerate strategy improvement algorithms. We call our algorithms non-oblivious because they remember properties of the game that they have discovered in previous iterations. We show that non-oblivious strategy improvement algorithms perform well on examples that are known to be hard for oblivious strategy improvement. Hence, we argue that previous strategy improvement algorithms fail because they ignore the structural properties of the game that they are solving.

1 Introduction

In this paper we study strategy improvement for two player infinite games played on finite graphs. In this setting the vertices of a graph are divided between two players. A token is placed on one of the vertices, and in each step the owner of the vertex upon which the token is placed must move the token along one of the outgoing edges of that vertex. In this fashion, the two players form an infinite path in the graph. The payoff of the game is then some property of this path, which depends on the type of game that is being played. Strategy improvement is a technique that originated from Markov decision processes [7], and has since been applied many types of games in this setting, including simple stochastic games [3], discounted-payoff games [12], mean-payoff games [2], and parity games [15,1]. In this paper we will focus on the strategy improvement algorithm of Björklund and Vorobyov [2], which is designed to solve mean-payoff games, but can also be applied to parity games.

Algorithms that solve parity and mean-payoff games have received much interest. One reason for this is that the model checking problem for the modal μ-calculus is polynomial time equivalent to the problem of solving a parity game [4,14], and there is a polynomial time reduction from parity games to mean-payoff games [12]. Therefore, faster algorithms for these games lead to faster model checkers for the μ-calculus. Secondly, both of these games lie in NP $\cap$ co-NP, which implies that neither of the two problems are likely to be complete for either class. Despite this, no polynomial time algorithms have been found.

The approach of strategy improvement can be described as follows. The algorithm begins by choosing one of the players to be the strategy improver, and then picks an arbitrary strategy for that player. A strategy for a player consists of a function that picks one edge for each of that player's vertices. Strategy improvement then computes a set of profitable edges for that strategy. If the strategy is switched so that it chooses some subset of the profitable edges, rather than the edges that are currently chosen, then strategy improvement guarantees that the resulting strategy is better in some well-defined measure. So, the algorithm picks some subset of the profitable edges to create a new, improved, strategy to be considered in the next iteration. This process is repeated until a strategy is found that has no profitable edges, and this strategy is guaranteed optimal for the strategy improver. Since any subset of the profitable edges could be used to create an improved strategy in each iteration, some method is needed to determine which subset to choose in each iteration. We call this method a switching policy, and the choice of switching policy can have a dramatic effect on the running time of the algorithm.

A significant amount of research has been dedicated to finding good switching policies. In terms of complexity bounds, the current best switching policies are randomized, and run in an expected $O(2^{\sqrt{n \log n}})$ number of iterations [2]. Another interesting switching policy is the optimal switching policy given by Schewe [13]. An optimal switching policy always picks the subset of profitable edges that yields the best possible successor strategy, according to the measure that strategy improvement uses to compare strategies. It is not difficult to show that such a subset of profitable edges must exist, but computing an optimal subset of profitable edges seemed to be difficult, since there can be exponentially many subsets of profitable edges to check. Nevertheless, Schewe's result is a polynomial time algorithm that computes an optimal subset of edges. Therefore, optimal switching policies can now be realistically implemented. It is important to note that the word "optimal" applies only to the subset of profitable edges that is chosen to be switched in each iteration. It is not the case that a strategy improvement algorithm equipped with an optimal switching policy will have an optimal running time.

Perhaps the most widely studied switching policy is the all-switches policy, which simply selects the entire set of profitable edges in every iteration. Although the best upper bound for this policy is $O(2^n/n)$ iterations [11], it has been found to work extremely well in practice. Indeed, for a period of ten years there were no known examples upon which the all switches policy took significantly more than a linear number of iterations. It was for this reason that the all-switches policy was widely held to be a contender for a proof of polynomial time termination.

However, Friedmann has recently found a family of examples that force a strategy improvement algorithm equipped with the all-switches policy to take an exponential number of steps [5]. Using the standard reductions [12,16], these examples can be generalised to provide exponential lower bounds for all-switches on mean-payoff and discounted-payoff games. Even more surprisingly, Friedmann's example can be generalised to provide an exponential lower bound for strategy improvement algorithms equipped with an optimal switching policy [6]. This recent revelation appears to imply that there is no longer any hope for strategy improvement, since an exponential number of iterations can be forced even if the best possible improvement is made in every step.

Our contributions. Despite ten years of research into strategy improvement algorithms, and the recent advances in the complexity of some widely studied switching policies, the underlying combinatorial structure of mean-payoff and parity games remains somewhat mysterious. There is no previous work which links the structural properties of a parity or mean-payoff game with the behaviour of strategy improvement on those games. In this paper, we introduce a structural property of these games that we call a snare. We show how the existence of a snare in a parity or mean-payoff game places a restriction on the form that a winning strategy can take for these games.

We argue that snares play a fundamental role in the behaviour of strategy improvement algorithms. We show that there is a certain type of profitable edge, which we call a back edge, that is the mechanism that strategy improvement uses to deal with snares. We show how each profitable back edge encountered by strategy improvement corresponds to some snare that exists in the game. Hence, we argue that the concept of a snare is a new tool that can be used in the analysis of strategy improvement algorithms.

We then go on to show that, in addition to being an analytical tool, awareness of snares can be used to accelerate the process of strategy improvement. We propose that strategy improvement algorithms should remember the snares that they have seen in previous iterations, and we give a procedure that uses a previously recorded snare to improve a strategy. Strategy improvement algorithms can choose to apply this procedure instead of switching a subset of profitable edges. We give one reasonable example of a strategy improvement algorithm that uses these techniques. We call our algorithms non-oblivious strategy improvement algorithms because they remember information about their previous iterations, whereas previous techniques make their decisions based only on the information available in the current iteration.

In order to demonstrate how non-oblivious techniques can be more powerful than traditional strategy improvement, we study Friedmann's family of examples that cause the all-switches and the optimal switching policies to take exponential time. We show that in certain situations non-oblivious strategy improvement makes better progress than even the optimal oblivious switching policy. We go on to show that this behaviour allows our non-oblivious strategy improvement algorithms to terminate in polynomial time on Friedmann's examples. This fact implies that it is ignorance of snares that is a key failing of oblivious strategy improvement.

2 Preliminaries

A mean-payoff game is defined by a tuple $(V, V_{\text{Max}}, V_{\text{Min}}, E, w)$ where V is a set of vertices and E is a set of edges, which together form a finite graph. Every vertex must have at least one outgoing edge. The sets V_{Max} and V_{Min} partition V into vertices belonging to player Max and vertices belonging to player Min, respectively. The function $w : V \to \mathbb{Z}$ assigns an integer weight to every vertex.

The game begins by placing a token on a starting vertex v_0. In each step, the player that owns the vertex upon which the token is placed must choose one outgoing edge of that vertex and move the token along it. In this fashion, the two players form an infinite path $\pi = \langle v_0, v_1, v_2, \dots \rangle$, where (v_i, v_{i+1}) is in E for every i in $\mathbb{N}$. The *payoff* of an infinite path is defined to be $\mathcal{M}(\pi) = \liminf_{n\to\infty}(1/n)\sum_{i=0}^{n} w(v_i)$. The objective of Max is to maximize the value of $\mathcal{M}(\pi)$, and the objective of Min is to minimize it.

A *positional strategy* for Max is a function that chooses one outgoing edge for every vertex belonging to Max. A strategy is denoted by $\sigma : V_{\text{Max}} \to V$, with the condition that $(v, \sigma(v))$ is in E, for every Max vertex v. Positional strategies for player Min are defined analogously. The sets of positional strategies for Max and Min are denoted by Π_{Max} and Π_{Min}, respectively. Given two positional strategies, σ and τ for Max and Min respectively, and a starting vertex v_0, there is a unique path $\langle v_0, v_1, v_2 \dots \rangle$, where $v_{i+1} = \sigma(v_i)$ if v_i is owned by Max and $v_{i+1} = \tau(v_i)$ if v_i is owned by Min. This path is known as the *play* induced by the two strategies σ and τ, and will be denoted by $\text{Play}(v_0, \sigma, \tau)$.

For all v in V we define:

$$\text{Value}_*(v) = \max_{\sigma\in\Pi_{\text{Max}}} \min_{\tau\in\Pi_{\text{Min}}} \mathcal{M}(\text{Play}(v, \sigma, \tau))$$
$$\text{Value}^*(v) = \min_{\tau\in\Pi_{\text{Min}}} \max_{\sigma\in\Pi_{\text{Max}}} \mathcal{M}(\text{Play}(v, \sigma, \tau))$$

These are known as the lower and upper values, respectively. For mean-payoff games we have that the two quantities are equal, a property called determinacy.

Theorem 1 ([10]). *For every starting vertex v in every mean-payoff game we have* $\text{Value}_*(v) = \text{Value}^*(v)$.

For this reason, we define $\text{Value}(v)$ to be the value of the game starting at the vertex v, which is equal to both $\text{Value}_*(v)$ and $\text{Value}^*(v)$. The computational task associated with mean-payoff games is to find $\text{Value}(v)$ for every vertex v.

Computing the 0-mean partition is a decision version of this problem. This requires us to decide whether $\text{Value}(v) > 0$, for every vertex v. Björklund and Vorobyov have shown that only a polynomial number of calls to an algorithm for finding the 0-mean partition are needed to find the value for every vertex in a mean-payoff game [2].

A Max strategy σ is a *winning strategy* for a set of vertices W if $\mathcal{M}(v, \sigma, \tau) > 0$ for every Min strategy τ and every vertex v in W. Similarly, a Min strategy τ is a winning strategy for W if $\mathcal{M}(v, \sigma, \tau) \leq 0$ for every Max strategy σ and every vertex v in W. To solve the 0-mean partition problem we are required to partition the vertices of the graph into the sets $(W_{\text{Max}}, W_{\text{Min}})$, where Max has a winning strategy for W_{Max} and Min has a winning strategy for W_{Min}.

3 Snares

In this section we introduce a structure called that we call a "snare". The dictionary definition[1] of the word snare is "something that serves to entangle the unwary". This is a particularly apt metaphor for these structures since, as we will show, a winning strategy for a player must be careful to avoid being trapped by the snares that are present in that player's winning set.

The definitions in this section could be formalized for either player. We choose to focus on player Max because we will later choose Max to be the strategy improver. For a set of vertices W we define $G \restriction W$ to be the sub-game induced by W, which is G with every vertex not in W removed. A snare for player Max is defined to be a subgame for which player Max can guarantee a win from every vertex.

Definition 2 (Max Snare). *For a game G, a snare is defined to be a tuple (W, χ) where $W \subseteq V$ and $\chi : W \cap V_{Max} \to W$ is a partial strategy for player Max that is winning for every vertex in the subgame $G \restriction W$.*

This should be compared with the concept of a dominion that was introduced by Jurdziński, Paterson, and Zwick [8]. A dominion is also a subgame in which one of the players can guarantee a win, but with the additional constraint that the opponent is unable to leave the dominion. By contrast, the opponent may be capable of leaving a snare. We define an escape edge for Min to be an edge that Min can use to leave a Max snare.

Definition 3 (Escapes). *Let W be a set of vertices. We define the escapes from W as $\mathrm{Esc}(W) = \{(v, u) \in E \;:\; v \in W \cap V_{Min} \text{ and } u \notin W\}$.*

It is in Min's interests to use at least one escape edge from a snare, since if Min stays in a Max snare forever, then Max can use the strategy χ to ensure a positive payoff. In fact, we can prove that if τ is a winning strategy for Min for some subset of vertices then τ must use at least one escape from every Max snare that exists in that subset of vertices.

Theorem 4. *Suppose that τ is a winning strategy for Min on a set of vertices S. If (W, χ) is a Max snare where $W \subset S$, then there is some edge (v, u) in $\mathrm{Esc}(W)$ such that $\tau(v) = u$.*

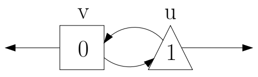

Fig. 1. A simple snare

Figure 1 shows an example of a subgame upon which a snare can be defined. In all of our diagrams, boxes are used to represent Max vertices and triangles are

[1] American Heritage Dictionary of the English Language, Fourth Edition.

used to represent Min vertices. The weight assigned to each vertex is shown on that vertex. If we take $W = \{v, u\}$ and $\chi(v) = u$ then (W, χ) will be a Max snare in every game that contains this structure as a subgame. This is because the cycle is positive, and therefore χ is a winning for Max on the subgame induced by W. There is one escape from this snare, which is the edge Min can use to break the cycle at u.

Since the example is so simple, Theorem 4 gives a particularly strong property for this snare: every winning strategy for Min must use the escape edge at u. If Min uses the edge (u, v) in some strategy, then Max could respond by using the edge (v, u) to guarantee a positive cycle, and therefore the strategy would not be winning for Min. This is a strong property because we can essentially ignore the edge (u, v) in every game into which the example is embedded. This property does not hold for snares that have more than one escape.

4 Strategy Improvement

In this section we will summarise Björklund and Vorobyov's strategy improvement algorithm for finding the 0-mean partition of a mean-payoff game [2]. Their algorithm requires that the game is modified by adding retreat edges from every Max vertex to a special sink vertex.

Definition 5 (Modified Game). *A game $(V, V_{Max}, V_{Min}, E, w)$ will be modified to create $(V \cup \{s\}, V_{Max} \cup \{s\}, V_{Min}, E', w')$, where $E' = E \cup \{(v, s) : v \in V_{Max}\}$, and $w'(v) = w(v)$ for all vertices v in V, and $w'(s) = 0$.*

Strategy improvement always works with the modified game, and for the rest of the paper we will assume that the game has been modified.

Given two strategies, one for each player, the play induced by the two strategies is either a finite path that ends at the sink or a finite initial path followed by an infinitely repeated cycle. This is used to define the valuation of a vertex.

Definition 6 (Valuation). *Let σ be a positional strategy for Max and τ be a positional strategy for Min. If* $\text{Play}(v_0, \sigma, \tau) = \langle v_0, v_1, \dots v_k, \langle c_0, c_1, \dots c_l\rangle^\omega\rangle$, *for some vertex v_0, then we define* $\text{Val}^{\sigma,\tau}(v_0) = -\infty$ *if* $\sum_{i=0}^{l} w(c_i) \leq 0$ *and* ∞ *otherwise. Alternatively, if* $\text{Play}(v, \sigma, \tau) = \langle v_0, v_1, \dots v_k, s\rangle$ *then we define* $\text{Val}^{\sigma,\tau}(v_0) = \sum_{i=0}^{k} w(v_i)$.

Strategy improvement algorithms choose one player to be the strategy improver, which we choose to be Max. For a Max strategy σ, we define $\text{br}(\sigma)$ to be the *best response* to σ, which is a Min strategy with the property $\text{Val}^{\sigma,\text{br}(\sigma)}(v) \leq \text{Val}^{\sigma,\tau}(v)$ for every vertex v and every Min strategy τ. Such a strategy always exists, and Björklund and Vorobyov give a method to compute it in polynomial time [2]. We will frequently want to refer to the valuation of a vertex v when the Max strategy σ is played against $\text{br}(\sigma)$, so we define $\text{Val}^\sigma(v)$ to be shorthand for $\text{Val}^{\sigma,\text{br}(\sigma)}(v)$. Occasionally, we will need to refer to valuations from multiple games. We use $\text{Val}^\sigma_G(v)$ to give the valuation of the vertex v when σ is played against $\text{br}(\sigma)$ in the game G. We extend all of our notations in a similar manner, by placing the game in the subscript.

For a Max strategy σ and an edge (v, u) that is not chosen by σ, we say (v, u) is *profitable* in σ if $\text{Val}^{\sigma}(\sigma(v)) < \text{Val}^{\sigma}(u)$. *Switching* an edge (v, u) in σ is denoted by $\sigma[v \mapsto u]$. This operation creates a new strategy where, for a vertex $w \in V_{\text{Max}}$ we have $\sigma[v \mapsto u](w) = u$ if $w = v$, and $\sigma(w)$ otherwise. Let F be a set of edges that contains at most one outgoing edge from each vertex. We define $\sigma[F]$ to be σ with every edge in F switched. The concept of profitability is important because switching profitable edges creates an improved strategy.

Theorem 7 ([2]). *Let σ be a strategy and P be the set of edges that are profitable in σ. Let $F \subseteq P$ be a subset of the profitable edges that contains at most one outgoing edge from each vertex. For every vertex v we have $\text{Val}^{\sigma}(v) \leq \text{Val}^{\sigma[W]}(v)$, and there is a vertex for which the inequality is strict.*

The second property that can be shown is that a strategy with no profitable edges is optimal. An optimal strategy is a Max strategy σ such that $\text{Val}^{\sigma}(v) \geq \text{Val}^{\chi}(v)$ for every Max strategy χ and every vertex v. The 0-mean partition can be derived from an optimal strategy σ: the set W_{Max} contains every vertex v with $\text{Val}^{\sigma}(v) = \infty$, and W_{Min} contains every vertex v with $\text{Val}^{\sigma}(v) < \infty$.

Theorem 8 ([2]). *A strategy with no profitable edges is optimal.*

Strategy improvement begins by choosing a strategy σ_0 with the property that $\text{Val}^{\sigma_0}(v) > -\infty$ for every vertex v. One way to achieve this is to set $\sigma_0(v) = s$ for every vertex v in V_{Max}. This guarantees the property unless there is some negative cycle that Min can enforce without passing through a Max vertex. Clearly, for a vertex v on one of these cycles, Max has no strategy σ with $\text{Val}^{\sigma}(v) > -\infty$. These vertices can therefore be removed in a preprocessing step and placed in W_{Min}.

For every strategy σ_i a new strategy $\sigma_{i+1} = \sigma_i[F]$ will be computed, where F is a subset of the profitable edges in σ_i, which contains at most one outgoing edge from each vertex. Theorem 7 implies that $\text{Val}^{\sigma_{i+1}}(v) \geq \text{Val}^{\sigma_i}(v)$ for every vertex v, and that there is a vertex for which the inequality is strict. This implies that a strategy cannot be visited twice by strategy improvement. The fact that there is a finite number of positional strategies for Max implies that strategy improvement must eventually reach a strategy σ_k in which no edges are profitable. Theorem 8 implies that σ_k is the optimal strategy, and strategy improvement terminates.

Strategy improvement requires a rule that determines which profitable edges are switched in each iteration. We will call this a *switching policy.* Oblivious switching policies are defined as $\alpha : 2^E \to 2^E$, where for every set $P \subseteq E$, we have that $\alpha(P)$ contains at most one outgoing edge for each vertex.

Some of the most widely studied switching policies are all-switches policies. These policies always switch every vertex that has a profitable edge, and when a vertex has more than one profitable edge an additional rule must be given to determine which edge to choose. Traditionally this choice is made by choosing the successor with the highest valuation. We must also be careful to break ties when there are two or more successors with the highest valuation. Therefore, for the purposes of defining this switching policy we will assume that each vertex v is given a unique index in the range $\{1, 2, \ldots, |V|\}$, which we will denote as $\text{Index}(v)$.

$$\begin{aligned}\text{All}(F) = \{(v, u) \ : \ &\text{There is no edge } (v, w) \in F \text{ with } \text{Val}^{\sigma}(u) < \text{Val}^{\sigma}(w) \\ &\text{or with } \text{Val}^{\sigma}(u) = \text{Val}^{\sigma}(w) \text{ and } \text{Index}(u) < \text{Index}(w)\}.\end{aligned}$$

In the introduction we described optimal switching policies, which we can now formally define. A switching policy is optimal if it selects a subset of profitable edges F that satisfies $\text{Val}^{\sigma[H]}(v) \leq \text{Val}^{\sigma[F]}(v)$ for every subset of profitable edges H and every vertex v. Schewe has given a method to compute such a set in polynomial time [13]. We will denote an optimal switching policy as Optimal.

5 Strategy Trees

The purpose of this section is to show how a strategy and its best response can be viewed as a tree, and to classify profitable edges by their position in this tree. We will classify edges as either cross edges or back edges. We will later show how profitable back edges are closely related to snares.

It is technically convenient for us to make the assumption that every vertex has a finite valuation under every strategy. The choice of starting strategy ensures that for every strategy σ considered by strategy improvement, we have $\text{Val}^{\sigma}(v) > -\infty$ for every vertex v. Obviously, there may be strategies under which some vertices have a valuation of ∞. The first part of this section is dedicated to rephrasing the problem so that our assumption can be made.

We define the *positive cycle* problem to be the problem of finding a strategy σ with $\text{Val}^{\sigma}(v) = \infty$ for some vertex v, or to prove that there is no strategy with this property. The latter can be done by finding an optimal strategy σ with $\text{Val}^{\sigma}(v) < \infty$ for every vertex v. We can prove that a strategy improvement algorithm for the positive cycle problem can be adapted to find the 0-mean partition.

Proposition 9. *Let α be a strategy improvement algorithm that solves the positive cycle problem in $O(\kappa)$ time. There is a strategy improvement algorithm which finds the 0-mean partition in $O(|V| \cdot \kappa)$ time.*

We consider switching policies that solve the positive cycle problem, and so we can assume that every vertex has a finite valuation under every strategy that our algorithms consider. Our switching policies will terminate when a vertex with infinite valuation is found. With this assumption we can define the strategy tree.

Definition 10 (Strategy Tree). *Given a Max strategy σ and a Min strategy τ we define the tree $T^{\sigma,\tau} = (V, E')$ where $E' = \{(v, u) \ : \ \sigma(v) = u \text{ or } \tau(v) = u\}$.*

In other words, $T^{\sigma,\tau}$ is a tree rooted at the sink whose edges are those chosen by σ and τ. We define T^{σ} to be shorthand for $T^{\sigma,\text{br}(\sigma)}$, and $\text{Subtree}^{\sigma}(v) : V \rightarrow 2^{V}$

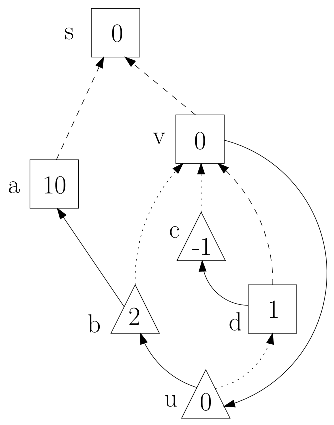

Fig. 2. A strategy tree

to be the function that gives the vertices in the subtree rooted at the vertex v in T^{σ}.

We can now define our classification for profitable edges. Let (v, u) be a profitable edge in the strategy σ. We call this a profitable *back edge* if u is in $\text{Subtree}^{\sigma}(v)$, otherwise we call it a profitable *cross edge*.

Figure 2 gives an example of a strategy tree. In all of our diagrams, the dashed lines give a strategy σ for Max, and the dotted lines show $\text{br}(\sigma)$. The strategy tree contains every vertex, and every edge that is either dashed or dotted. The subtree of v is the set $\{v, b, c, d, u\}$. The edge (v, u) is profitable because $\text{Val}^{\sigma}(v) = 0$ and $\text{Val}^{\sigma}(u) = 1$. Since u is contained in the subtree of v, the edge (v, u) is a profitable back edge.

6 Profitable Back Edges

In this section we will expose the intimate connection between profitable back edges and snares. We will show how every profitable back edge corresponds to some snare that exists in the game. We will also define the concept of snare consistency, and we will show how this concept is linked with the conditions implied by Theorem 4.

Our first task is to show how each profitable back edge corresponds to some Max snare in the game. Recall that a Max snare consists of a set of vertices, and a strategy for Max that is winning for the subgame induced by those vertices. We will begin by defining the set of vertices for the snare that corresponds to a profitable back edge. For a profitable back edge (v, u) in a strategy σ we define the critical set, which is the vertices in $\text{Subtree}^{\sigma}(v)$ that Min can reach when Max plays σ.

Definition 11 (Critical Set). *If (v, u) is a profitable back edge in the strategy σ, then we define the critical set as* $\text{Critical}^{\sigma}(v, u) = \{w \in \text{Subtree}^{\sigma}(v)\ :$

There is a path $\langle u, u_1, \dots u_k = w \rangle$ *where for all* i *with* $1 \leq i \leq k$ *we have* $u_i \in \mathrm{Subtree}^{\sigma}(v)$ *and if* $u_i \in V_{Max}$ *then* $u_{i+1} = \sigma(u_i)\}$*.*

In the example given in Figure 2, the critical set for the edge (v, u) is $\{v, b, d, u\}$. The vertex b is in the critical set because it is in the subtree of v, and Min can reach it from u when Max plays σ. In contrast, the vertex c is not in the critical set because $\sigma(d) = v$, and therefore Min cannot reach c from u when Max plays σ. The vertex a is not in the critical set because it is not in the subtree of v.

Note that in the example, $\sigma[v \mapsto u]$ is a winning strategy for the subgame induced by critical set. The definition of the critical set is intended to capture the largest connected subset of vertices contained in the subtree of v for which $\sigma[v \mapsto u]$ is guaranteed to be a winning strategy.

Proposition 12. *Let* (v, u) *be a profitable back edge in the strategy* σ *and let* C *be* $\mathrm{Critical}^{\sigma}(v, u)$*. The strategy* $\sigma[v \mapsto u]$ *is winning for every vertex in* $G \upharpoonright C$*.*

We can now formally define the snare that is associated with each profitable back edge that is encountered by strategy improvement. For a profitable back edge (v, u) in a strategy σ we define $\mathrm{Snare}^{\sigma}(v, u) = (\mathrm{Critical}^{\sigma}(v, u), \chi)$ where $\chi(v) = \sigma[v \mapsto u](v)$ if $v \in \mathrm{Critical}^{\sigma}(v, u)$, and undefined at other vertices. Proposition 12 confirms that this meets the definition of a snare.

We will now argue that the conditions given by Theorem 4 must be observed in order for strategy improvement to terminate. We begin by defining a concept that we call snare consistency. We say that a Max strategy is consistent with a snare if Min's best response chooses an escape from that snare.

Definition 13 (Snare Consistency). *A strategy* σ *is said to be consistent with the snare* (W, χ) *if* $\mathrm{br}(\sigma)$ *uses some edge in* $\mathrm{Esc}(W)$*.*

In the example given in Figure 2 we can see that σ is not consistent with $\mathrm{Snare}^{\sigma}(v, u)$. This is because $\mathrm{br}(\sigma)$ does not choose the edge (b, a). However, once the edge (v, u) is switched we can prove that $\mathrm{br}(\sigma[v \mapsto u])$ must use the edge (b, a). This is because Min has no other way of connecting every vertex in $\mathrm{Subtree}^{\sigma}(v)$ to the sink, and if some vertex is not connected to the sink then its valuation will rise to ∞.

Proposition 14. *Let* (v, u) *be a profitable back edge in the strategy* σ*. There is some edge* (x, y) *in* $\mathrm{Esc}(\mathrm{Critical}^{\sigma}(v, u))$ *such that* $\mathrm{br}(\sigma[v \mapsto u])(x) = y$*.*

We can show that strategy improvement cannot terminate unless the current strategy is consistent with every snare that exists in the game. This is because every strategy that is not consistent with some snare must contain a profitable edge.

Proposition 15. *Let* σ *be a strategy that is not consistent with a snare* (W, χ)*. There is a profitable edge* (v, u) *in* σ *such that* $\chi(v) = u$*.*

These two propositions give us a new tool to study the process of strategy improvement. Instead of viewing strategy improvement as a process that tries to increase valuations, we can view it as a process that tries to force consistency with Max snares. Proposition 15 implies that this process can only terminate when the current strategy is consistent with every Max snare in the game. Therefore, the behaviour of strategy improvement on an example is strongly related with the snares that exist for the strategy improver in that example.

7 Using Snares to Guide Strategy Improvement

In the previous sections, we have shown the strong link between snares and strategy improvement. In this section we will show how this insight can be used to guide strategy improvement. We will give a procedure that takes a strategy that is inconsistent with some snare, and returns an improved strategy that is consistent with that snare. Since the procedure is guaranteed to produce an improved strategy, it can be used during strategy improvement as an alternative to switching a profitable edge. We call algorithms that make use of this procedure non-oblivious strategy improvement algorithms, and we give a reasonable example of such an algorithm.

To define our procedure we will use Proposition 15. Recall that this proposition implies that if a strategy σ is inconsistent with a snare (W, χ), then there is some profitable edge (v, u) in σ such that $\chi(v) = u$. Our procedure will actually be a strategy improvement switching policy. This policy will always choose to switch an edge that is chosen by χ but not by the current strategy. As long as the current strategy remains inconsistent with (W, χ) such an edge is guaranteed to exist, and the policy terminates once the current strategy is consistent with the snare. This procedure is shown as Algorithm 1.

Algorithm 1. FixSnare(σ, (W, χ))

```
while $\sigma$ is inconsistent with $(W, \chi)$ do
    $(v, w)$ := Some edge where $\chi(v) = w$ and $(v, w)$ is profitable in $\sigma$.
    $\sigma := \sigma[v \mapsto u]$
end while
return $\sigma$
```

In each iteration the switching policy switches one vertex v to an edge (v, u) with the property that $\chi(v) = u$, and it never switches a vertex at which the current strategy agrees with χ. It is therefore not difficult to see that if the algorithm has not terminated after $|W|$ iterations then the current strategy will agree with χ on every vertex in W. We can prove that such a strategy must be consistent with (W, χ), and therefore the switching policy must terminate after at most $|W|$ iterations.

Proposition 16. *Let σ be a strategy that is not consistent with a snare (W, χ). Algorithm 1 will arrive at a strategy σ' which is consistent with (W, χ) after at most $|W|$ iterations.*

Since FixSnare is implemented as a strategy improvement switching policy that switches only profitable edges, the strategy that is produced must be an improved strategy. Therefore, at any point during the execution of strategy improvement we can choose not to switch a subset of profitable edges and run FixSnare instead. Note that the strategy produced by FixSnare may not be reachable from the current strategy by switching a subset of profitable edges. This is because FixSnare switches a sequence of profitable edges, some of which may not have been profitable in the original strategy.

We propose a new class of strategy improvement algorithms that are aware of snares. These algorithms will record a snare for every profitable back edge that they encounter during their execution. In each iteration these algorithms can either switch a subset of profitable edges or run the procedure FixSnare on some recorded snare that the current strategy is inconsistent with. We call these algorithms non-oblivious strategy improvement algorithms, and the general schema that these algorithms follow is shown in Algorithm 2.

Algorithm 2. NonOblivious(σ)

```
$S := \emptyset$
**while** $\sigma$ has a profitable edge **do**
    $S := S \cup \{\text{Snare}^{\sigma}(v, u) \;:\; (v, u) \text{ is a profitable back edge in } \sigma\}$
    $\sigma := \text{Policy}(\sigma, S)$
**end while**
**return** $\sigma$
```

Recall that oblivious strategy improvement algorithms required a switching policy to specify which profitable edges should be switched in each iteration. Clearly, non-oblivious strategy improvement algorithms require a similar method to decide whether to apply the procedure FixSnare or to pick some subset of profitable edges to switch. Moreover, they must decide which snare should be used when the procedure FixSnare is applied. We do not claim to have the definitive non-oblivious switching policy, but in the rest of this section we will present one reasonable method of constructing a non-oblivious version of an oblivious switching policy. We will later show that our non-oblivious strategy improvement algorithms behave well on the examples that are known to cause exponential time behaviour for oblivious strategy improvement.

We intend to take an oblivious switching policy α as the base of our non-oblivious switching policy. This means that when we do not choose to use the procedure FixSnare, we will switch the subset of profitable edges that would be chosen by α. Our goal is to only use FixSnare when doing so is guaranteed to yield a larger increase in valuation than applying α. Clearly, in order to achieve this we must know how much the valuations increase when α is applied and how much the valuations increase when FixSnare is applied.

Determining the increase in valuation that is produced by applying an oblivious switching policy is easy. Since every iteration of oblivious strategy improvement takes polynomial time, We can simply switch the edges and measure the difference between the current strategy and the one that would be produced. Let σ be a strategy and let P be the set of edges that are profitable in σ. For an oblivious switching policy α the increase of applying α is defined to be:

$$\text{Increase}(\alpha, \sigma) = \sum_{v \in V} (\text{Val}^{\sigma[\alpha(P)]}(v) - \text{Val}^{\sigma}(v))$$

We now give a lower bound on the increase in valuation that an application of FixSnare produces. Let (W, χ) be a snare and suppose that the current strategy σ is inconsistent with this snare. Our lower bound is based on the fact that FixSnare will produce a strategy that is consistent with the snare. This means that Min's best response is not currently choosing an escape from the snare, but it will be forced to do so after FixSnare has been applied. It is easy to see that forcing the best response to use a different edge will cause an increase in valuation, since otherwise the best response would already be using that edge. Therefore, we can use the increase in valuation that will be obtained when Min is forced to use and escape. We define:

$$\text{SnareIncrease}^{\sigma}(W, \chi) = \min\{(\text{Val}^{\sigma}(y) + w(x)) - \text{Val}^{\sigma}(x) \ : \ (x, y) \in \text{Esc}(W)\}$$

This expression gives the smallest possible increase in valuation that can happen when Min is forced to use an edge in $\text{Esc}(W)$. We can prove that applying FixSnare will cause an increase in valuation of at least this amount.

Proposition 17. *Let σ be a strategy that is not consistent with a snare (W, χ), and let σ' be the result of* FixSnare$(\sigma, (W, \chi))$. *We have:*

$$\sum_{v \in V} (\text{Val}^{\sigma'}(v) - \text{Val}^{\sigma}(v)) \geq \text{SnareIncrease}^{\sigma}(W, \chi)$$

We now have the tools necessary to construct our proposed augmentation scheme, which is shown as Algorithm 3. The idea is to compare the increase obtained by applying α and the increase obtained by applying FixSnare with the best snare that has been previously recorded, and then to only apply FixSnare when it is guaranteed to yield a larger increase in valuation.

Algorithm 3. $(\text{Augment}(\alpha))(\sigma, S)$

```
$(W, \chi) := \text{argmax}_{(X,\mu) \in S} \text{SnareIncrease}^{\sigma}(X, \mu)$
if $\text{Increase}(\alpha, \sigma) > \text{SnareIncrease}^{\sigma}(W, \chi)$ then
    $P := \{(v, u) \ : \ (v, u) \text{ is profitable in } \sigma\}$
    $\sigma := \sigma[\alpha(P)]$
else
    $\sigma := \text{FixSnare}(\sigma, (W, \chi))$
end if
return $\sigma$
```

8 Comparison with Oblivious Strategy Improvement

In this section we will demonstrate how non-oblivious strategy improvement can behave well in situations where oblivious strategy improvement has exponential time behaviour. Unfortunately, there is only one source of examples with such properties in the literature, and that is the family of examples given by Friedmann. In fact, Friedmann gives two slightly different families of hard examples. The first type is the family that that forces exponential behaviour for the all-switches policy [5], and the second type is the family that forces exponential behaviour for both all-switches and optimal switching policies [6]. Although our algorithm performs well on both families, we will focus on the example that was designed for optimal switching policies because it is the most interesting of the two.

This section is split into two parts. In the first half of this section we will study a component part of Friedmann's example upon which the procedure FixSnare can out perform an optimal switching policy. This implies that there are situations in which our augmentation scheme will choose to use FixSnare. In the second half, we will show how the good performance on the component part is the key property that allows our non-oblivious strategy improvement algorithms to terminate quickly on Friedmann's examples.

8.1 Optimal Switching Policies

We have claimed that the procedure FixSnare can cause a greater increase in valuation than switching any subset of profitable edges. We will now give an example upon which this property holds. The example that we will consider is shown in Figure 3, and it is one of the component parts of Friedmann's family of examples that force optimal policies to take an exponential number of steps [6].

The diagram shows a strategy for Max as a set of dashed edges. It also shows Min's best response to this strategy as a dotted edge. Even though this example

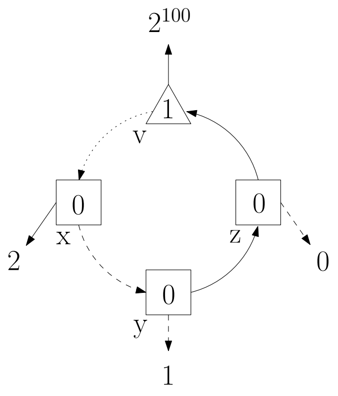

Fig. 3. A component of Friedmann's exponential time example

could be embedded in an arbitrary game, we can reason about the behaviour of strategy improvement by specifying, for each edge that leaves the example, the valuation of the successor vertex that the edge leads to. These valuations are shown as numbers at the end of each edge that leaves the example.

In order to understand how strategy improvement behaves we must determine the set of edges that are profitable for our strategy. There are two edges that are profitable: the edge (z, v) is profitable because the valuation of v is 2 which is greater than 0, and the edge at x that leaves the example is profitable because leaving the example gives a valuation of 2 and the valuation of y is 1. The edge (y, z) is not profitable because the valuation of z is 0, which is smaller than the valuation of 1 obtained by leaving the example at y.

For the purposes of demonstration, we will assume that no other edge is profitable in the game into which the example is embedded. Furthermore, we will assume that no matter what profitable edges are chosen to be switched, the valuation of every vertex not contained in the example will remain constant. Therefore, the all-switches policy will switch the edges (z, v) and the edge leading away from the example at the vertex x. It can easily be verified that this is also the optimal subset of profitable edges, and so the all-switches and the optimal policies make the same decisions for this strategy. After switching the edges chosen by the two policies, the valuation of x will rise to 2, the valuation of z will rise to 3, and the valuation of y remain at 1.

By contrast, we will now argue that non-oblivious strategy improvement would raise the valuations of x, y, and z to $2^{100} + 1$. Firstly, it is critical to note that the example is a snare. If we set $W = \{v, x, y, z\}$ and choose χ to be the partial strategy for Max that chooses the edges (x, y), (y, z), and (z, v), then (W, χ) will be a snare in every game into which the example is embedded. This is because there is only one cycle in the subgame induced by W when Max plays χ, and this cycle has positive weight.

Now, if the non-oblivious strategy improvement algorithm was aware of the snare (W, χ) then the lower bound given by Proposition 17 would be 2^{100}. This is because closing the cycle forces Min's best response to use escape edge to avoid losing the game. Since 2^{100} is much larger than the increase obtained by the optimal switching policy, the policies Augment(All) and Augment(Optimal) will choose to run FixSnare on the snare (W, χ). Once consequence of this is that the policy Optimal is no longer optimal in the non-oblivious setting.

8.2 Friedmann's Exponential Time Examples

The example that we gave in the previous subsection may appear to be trivial. After all, if the valuations outside the example remain constant then both the all-switches and optimal switching policies will close the cycle in two iterations. A problem arises, however, when the valuations can change. Note that when we applied the oblivious policies to the example, no progress was made towards closing the cycle. We started with a strategy that chose to close the cycle at only one vertex, and we produced a strategy that chose to close the cycle at only one vertex. When the assumption that valuations outside the example are constant is removed, it becomes possible for a well designed game to delay the closing of the cycle for an arbitrarily large number of iterations simply by repeating the pattern of valuations that is shown in Figure 3.

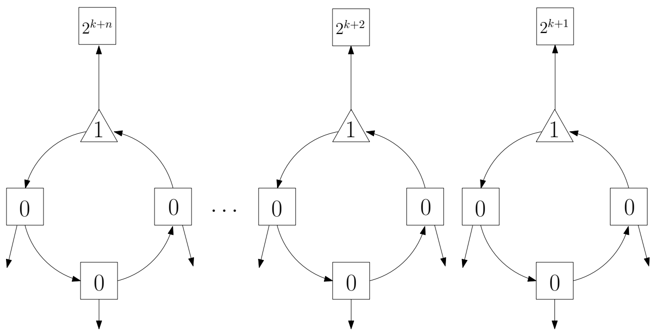

Fig. 4. The bits of a binary counter

Friedmann's family of examples exploits this property to build a binary counter, which uses the subgame shown in Figure 3 to represent the bits. The general idea of this approach is shown in Figure 4. Friedmann's example uses n instances of the cycle, indexed 1 through n. These bits are interconnected in a way that enforces two properties on both the all-switches and the optimal switching policies. Firstly, the ability to prevent a cycle from closing that we have described is used to ensure that the cycle with index i can only be closed after every cycle with index smaller than i has been closed. Secondly, when the cycle with index i is closed, every cycle with index smaller than i is forced to open. Finally, every cycle is closed in the optimal strategy for the example. Now, if the initial strategy is chosen so that every cycle is open, then these three properties are sufficient to force both switching policies to take at least 2^n steps before terminating.

The example works by forcing the oblivious switching policy to make the same mistakes repeatedly. To see this, consider the cycle with index $n-1$. When the cycle with index n is closed for the first time, this cycle is forced open. The oblivious optimal switching policy will then not close it again for at least another 2^{n-1} steps. By contrast, the policies Augment(All) and Augment(Optimal) would close the cycle again after a single iteration. This breaks the exponential time behaviour, and it turns out that both of our policies terminate in polynomial time on Friedmann's examples.

Of course, for Friedmann's examples we can tell by inspection that Max always wants to keep the cycle closed. It is not difficult, however, to imagine an example which replaces the four vertex cycle with a complicated subgame, for which Max had a winning strategy and Min's only escape is to play to the vertex with a large weight. This would still be a snare, but the fact that it is a snare would only become apparent during the execution of strategy improvement. Nevertheless, as long as the subgame can be solved in polynomial time by non-oblivious strategy improvement, the whole game will also be solved in polynomial time. This holds for exactly the same reason as the polynomial behaviour on Friedmann's examples: once the snare representing the subgame has been recorded then consistency with that snare can be enforced in the future.

9 Conclusions and Further Work

This paper has uncovered and formalized a strong link between the snares that exist in a game and the behaviour of strategy improvement on that game. We have shown how this link can be used to guide the process of strategy improvement. With our augmentation procedure we gave one reasonable method of incorporating non-oblivious techniques into traditional strategy improvement, and we have demonstrated how these techniques give rise to good behaviour on the known exponential time examples.

It must be stressed that we are not claiming that simply terminating in polynomial time on Friedmann's examples is a major step forward. After all, the randomized switching policies of Björklund and Vorobyov [2] have the same property. What is important is that our strategy improvement algorithms are polynomial because they have a better understanding of the underlying structure of strategy improvement. Friedmann's examples provide an excellent cautionary tale that shows how ignorance of this underlying structure can lead to exponential time behaviour.

There are a wide variety of questions that are raised by this work. Firstly, we have the structure of snares in parity and mean-payoff games. Theorem 4 implies that all algorithms that find winning strategies for parity and mean payoff games must, at least implicitly, consider snares. We therefore propose that a thorough and complete understanding of how snares arise in a game is a necessary condition for devising a polynomial time algorithm for these games.

It is not currently clear how the snares in a game affect the difficulty of solving that game. It is not difficult, for example, to construct a game in which there an exponential number of Max snares: in a game in which every weight is positive there will be a snare for every connected subset of vertices. However, games with only positive weights have been shown to be very easy to solve [9]. Clearly, the first challenge is to give a clear formulation of how the structure of the snares in a given game affects the difficulty of solving it.

In our attempts to construct intelligent non-oblivious strategy improvement algorithms we have continually had problems with examples in which Max and Min snares overlap. By this we mean that the set of vertices that define the subgames of the snares have a non empty intersection. We therefore think that studying how complex the overlapping of snares can be in a game may lead to further insight. There are reasons to believe that these overlappings cannot be totally arbitrary, since they arise from the structure of the game graph and the weights assigned to the vertices.

We have presented a non-oblivious strategy improvement algorithm that passively records the snares that are discovered by an oblivious switching policy, and then uses those snares when doing so is guaranteed to lead to a larger increase in valuations. While we have shown that this approach can clearly outperform traditional strategy improvement, it does not appear to immediately lead to a proof of polynomial time termination. It would be interesting to find an exponential time example for the augmented versions of the all-switches policy or of the optimal policy. This may be significantly more difficult since it is no longer possible to trick strategy improvement into making slow progress by forcing it to repeatedly close a small number of snares.

There is no inherent reason why strategy improvement algorithms should be obsessed with trying to increase valuations as much as possible in each iteration. Friedmann's exponential time example for the optimal policy demonstrates that doing so in no way guarantees that the algorithm will always make good progress. Our work uncovers an alternate objective that strategy improvement algorithms can use to measure their progress. Strategy improvement algorithms could actively try to discover the snares that exist in the game, or they could try and maintain consistency with as many snares as possible, for example. There is much scope for an intelligent snare based strategy improvement algorithm.

We have had some limited success in designing intelligent snare based strategy improvement algorithms for parity games. We have developed a non-oblivious strategy improvement algorithm which, when given a list of known snares in the game, either solves the game or finds a snare that is not in the list of known snares. This gives the rather weak result of a strategy improvement algorithm whose running time is polynomial in $|V|$ and k, where k is the number of Max snares that exist in the game. This is clearly unsatisfactory since we have already argued that k could be exponential in the number of vertices. However, this is one example of how snares can be applied to obtain new bounds for strategy improvement. As an aside, the techniques that we used to obtain this algorithm do not generalize to mean-payoff games. Finding a way to accomplish this task for mean-payoff games is an obvious starting point for designing intelligent snare based algorithms for this type of game.

Acknowledgements. I am indebted to Marcin Jurdziński for his guidance, support, and encouragement during the preparation of this paper.

References

1. Björklund, H., Sandberg, S., Vorobyov, S.: A discrete subexponential algorithm for parity games. In: Alt, H., Habib, M. (eds.) STACS 2003. LNCS, vol. 2607, pp. 663–674. Springer, Heidelberg (2003)
2. Björklund, H., Vorobyov, S.: A combinatorial strongly subexponential strategy improvement algorithm for mean payoff games. Discrete Applied Mathematics 155(2), 210–229 (2007)
3. Condon, A.: On algorithms for simple stochastic games. In: Cai, J.-Y. (ed.) Advances in Computational Complexity Theory. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 13, pp. 51–73. American Mathematical Society, Providence (1993)

4. Emerson, E.A., Jutla, C.S., Sistla, A.P.: On model-checking for fragments of μ-calculus. In: Courcoubetis, C. (ed.) CAV 1993. LNCS, vol. 697, pp. 385–396. Springer, Heidelberg (1993)
5. Friedmann, O.: A super-polynomial lower bound for the parity game strategy improvement algorithm as we know it. In: Logic in Computer Science (LICS). IEEE, Los Alamitos (2009)
6. Friedmann, O.: A super-polynomial lower bound for the parity game strategy improvement algorithm as we know it (January 2009) (preprint)
7. Howard, R.: Dynamic Programming and Markov Processes. Technology Press and Wiley (1960)
8. Jurdziński, M., Paterson, M., Zwick, U.: A deterministic subexponential algorithm for solving parity games. In: Proceedings of ACM-SIAM Symposium on Discrete Algorithms, SODA 2006, pp. 117–123. ACM/SIAM (2006)
9. Khachiyan, L., Gurvich, V., Zhao, J.: Extending dijkstras algorithm to maximize the shortest path by node-wise limited arc interdiction. In: Grigoriev, D., Harrison, J., Hirsch, E.A. (eds.) CSR 2006. LNCS, vol. 3967, pp. 221–234. Springer, Heidelberg (2006)
10. Liggett, T.M., Lippman, S.A.: Stochastic games with perfect information and time average payoff. SIAM Review 11(4), 604–607 (1969)
11. Mansour, Y., Singh, S.P.: On the complexity of policy iteration. In: Laskey, K.B., Prade, H. (eds.) UAI 1999: Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence, pp. 401–408. Morgan Kaufmann, San Francisco (1999)
12. Puri, A.: Theory of Hybrid Systems and Discrete Event Systems. PhD thesis, University of California, Berkeley (1995)
13. Schewe, S.: An optimal strategy improvement algorithm for solving parity and payoff games. In: Kaminski, M., Martini, S. (eds.) CSL 2008. LNCS, vol. 5213, pp. 369–384. Springer, Heidelberg (2008)
14. Stirling, C.: Local model checking games (extended abstract). In: Lee, I., Smolka, S.A. (eds.) CONCUR 1995. LNCS, vol. 962, pp. 1–11. Springer, Heidelberg (1995)
15. Vöge, J., Jurdziński, M.: A discrete strategy improvement algorithm for solving parity games (Extended abstract). In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, pp. 202–215. Springer, Heidelberg (2000)
16. Zwick, U., Paterson, M.: The complexity of mean payoff games on graphs. Theoretical Computer Science 158(1-2), 343–359 (1996)