Stuttering Mostly Speeds Up Solving Parity Games

Sjoerd Cranen, Jeroen J.A. Keiren, and Tim A.C. Willemse

Department of Mathematics and Computer Science,
Technische Universiteit Eindhoven,
P.O. Box 513, 5600 MB Eindhoven, The Netherlands

Abstract. We study the process theoretic notion of stuttering equivalence in the setting of parity games. We demonstrate that stuttering equivalent vertices have the same winner in the parity game. This means that solving a parity game can be accelerated by minimising the game graph with respect to stuttering equivalence. While, at the outset, it might not be clear that this strategy should pay off, our experiments using typical verification problems illustrate that stuttering equivalence speeds up solving parity games in many cases.

1 Introduction

Parity games [6,13,22] are played by two players (called *even* and *odd*) on a directed graph in which vertices have been assigned *priorities*. Every vertex in the graph belongs to exactly one of these two players. The game is played by moving a token along the edges in the graph indefinitely; the edge that is moved along is chosen by the player owning the vertex on which the token currently resides. Priorities that appear infinitely often along such infinite plays then determine the winner of the play.

Solving a parity game essentially boils down to computing the set of vertices that, if the token is initially placed on a vertex in this set, allows player *even* (resp. *odd*) to win. This problem is known to be in NP ∩ co-NP; it is still an open problem whether a polynomial time algorithm exists for the problem, but even in case such an algorithm is found, it may not be the most efficient algorithm in practice.

Parity games play a crucial role in verification; the model checking problem for the modal μ-calculus can be reduced to the problem of solving a given parity game. It is therefore worthwile to investigate methods by which these games can be solved efficiently in practice. In [7], Friedman and Lange describe a meta-algorithm that, combined with a set of heuristics, appears to have a positive impact on the time required to solve parity games. Fritz and Wilke consider more-or-less tried and tested techniques for *minimising* parity games using novel refinement and equivalence relations, see [9]. The delayed simulation they introduce, and its induced equivalence relation, however, are problematic for quotienting, which is why they go on to define two variations of delayed simulations that do not suffer from this problem. As stated in [8], however, "Experiments

M. Bobaru et al. (Eds.): NFM 2011, LNCS 6617, pp. 207–221, 2011.

indicate that simplifying parity games using our approach before solving them is not faster than solving them outright in practice".

Despite the somewhat unsatisfactory performance of the delayed simulation in practice, we follow a methodology similar to the one pursued by Fritz and Wilke. As a basis for our investigations, we consider *stuttering equivalence* [3], which originated in the setting of Kripke Structures. Stuttering equivalence has two qualities that make it an interesting candidate for minimising parity games. Firstly, vertices with the same player and priority are only distinguished on the basis of their future branching behaviour, allowing for a considerable compression. Secondly, stuttering equivalence has a very attractive worst-case time complexity of $\mathcal{O}(n \cdot m)$, for n vertices and m edges, which is in stark contrast to the far less favourable time complexity required for delayed simulation, which is $\mathcal{O}(n^3 \cdot m \cdot d^2)$, where d is the number of different priorities in the game. In addition to these, stuttering equivalence has several other traits that make it appealing: quotienting is straightforward, distributed algorithms for computing stuttering equivalence have been developed (see *e.g.* [2]), and it admits efficient, scalable implementations using BDD technology [21].

On the basis of the above qualities, stuttering equivalence is likely to significantly compress parity games that stem from typical model checking problems. Such games often have a rather limited number of priorities (typically at most three), and appear to have regular structures. We note that, as far as we have been able to trace, quotienting parity games using stuttering equivalence has never been shown to be sound. Thus, our contributions in this paper are twofold.

First, we show that stuttering equivalent vertices are won by the same player in the parity game. As a side result, given a winning strategy for a player for a particular vertex, we obtain winning strategies for all stuttering equivalent vertices. This is of particular interest in case one is seeking an explanation for the solution of the game, for instance as a means for diagnosing a failed verification.

Second, we experimentally show that computing and subsequently solving the stuttering quotient of a parity game is in many cases *faster* than solving the original game. In our comparison, we included several competitive implementations of algorithms for solving parity games, including several implementations of *Small Progress Measures* [11] and McNaughton's *recursive* algorithm [13]. Moreover, we also compare it to quotienting using *strong bisimulation* [15]. For an up-to-date overview of experiments we refer to [5], which we plan to keep updated with new results. While we do not claim that stuttering equivalence minimisation should always be performed prior to solving a parity game, we are optimistic about its effects in practical verification tasks.

Structure. The remainder of this paper is organised as follows. Section 2 briefly introduces the necessary background for parity games. In Section 3 we define both strong bisimilarity and stuttering equivalence in the setting of parity games; we show that both can be used for minimising parity games. Section 4 is devoted to describing our experiments, demonstrating the efficacy of stuttering equivalence minimisation on a large set of verification problems. In Section 5, we briefly discuss future work and open issues.

2 Preliminaries

We assume the reader has some familiarity with parity games; therefore, the main purpose of this section is to fix terminology and notation. For an in-depth treatment of these games, we refer to [13,22].

2.1 Parity Games

A parity game is a game played by players *even* (represented by the symbol 0) and *odd* (represented by the symbol 1). It is played on a total finite directed graph, the vertices of which can be won by either 0 or 1. The objective of the game is to find the partitioning that separates the vertices won by 0 from those won by 1. In the following text we formalise this definition, and we introduce some concepts that will make it easier to reason about parity games.

Definition 1. *A parity game $\mathcal{G}$ is a directed graph $(V, \rightarrow, \Omega, \mathcal{P})$, where*

- *V is a finite set of vertices,*
- *$\rightarrow \subseteq V \times V$ is a total edge relation (i.e., for each $v \in V$ there is at least one $w \in V$ such that $(v, w) \in \rightarrow$),*
- *$\Omega : V \rightarrow \mathbb{N}$ is a priority function that assigns priorities to vertices,*
- *$\mathcal{P} : V \rightarrow \{0, 1\}$ is a function assigning vertices to players.*

Instead of $(v, w) \in \rightarrow$ we will usually write $v \rightarrow w$. Note that, for the purpose of readability later in this text, our definition deviates from the conventional one: instead of requiring a partitioning of V into vertices owned by player even and vertices owned by player odd, we achieve the same through the function $\mathcal{P}$.

Paths. A sequence of vertices $v_1, \ldots, v_n$ for which $v_i \rightarrow v_{i+1}$ for all $1 \leq i < n$ is called a *path*, and may be denoted using angular brackets: $\langle v_1, \ldots, v_n \rangle$. The concatenation $p \cdot q$ of paths p and q is again a path. We use p_n to denote the n^{th} vertex in a path p. The set of paths of length n, for $n \geq 1$ starting in a vertex v is defined inductively as follows.

$$\begin{aligned} \Pi^1(v) &= \{\langle v \rangle\} \\ \Pi^{n+1}(v) &= \{\langle v_1, \ldots, v_n, v_{n+1} \rangle \mid \langle v_1, \ldots, v_n \rangle \in \Pi^n(v) \wedge v_n \rightarrow v_{n+1}\} \end{aligned}$$

We use $\Pi^\omega(v)$ to denote the set of infinite paths starting in v. The set of all paths starting in v, both finite and infinite is defined as follows:

$$\Pi(v) = \Pi^\omega(v) \cup \bigcup_{n \in \mathbb{N}} \Pi^n(v)$$

Winner. A game starting in a vertex $v \in V$ is played by placing a token on v, and then moving the token along the edges in the graph. Moves are taken indefinitely according to the following simple rule: if the token is on some vertex v, player $\mathcal{P}(v)$ moves the token to some vertex w such that $v \rightarrow w$. The result is an infinite path p in the game graph. The *parity* of the lowest priority that occurs infinitely often on p defines the *winner* of the path. If this priority is even, then player 0 wins, otherwise player 1 wins.

Strategies. A *strategy* for player i is a partial function $\phi : V^* \to V$, that for each path ending in a vertex owned by player i determines the next vertex to be played onto. A path p of length n is *consistent* with a strategy ϕ for player i, denoted $\phi \Vdash p$, if and only if for all $1 \leq j < n$ it is the case that $\langle p_1, \ldots, p_j \rangle \in \mathsf{dom}(\phi)$ and $\mathcal{P}(p_j) = i$ imply $p_{j+1} = \phi(\langle p_1, \ldots, p_j \rangle)$. The definition of consistency is extended to infinite paths in the obvious manner. We denote the set of paths that are consistent with a given strategy ϕ, starting in a vertex v by $\Pi_\phi(v)$; formally, we define:

$$\Pi_\phi(v) = \{p \in \Pi(v) \mid \phi \Vdash p\}$$

A strategy ϕ for player i is said to be a *winning strategy* from a vertex v if and only if i is the winner of every path that starts in v and that is consistent with ϕ. It is known from the literature that each vertex in the game is won by exactly one player; effectively, this induces a partitioning on the set of vertices V in those vertices won by player 0 and those vertices won by player 1.

Orderings. We assume that V is ordered by an arbitrary, total ordering $\sqsubset$. The minimal element of a non-empty set $U \subseteq V$ with respect to this ordering is denoted $\sqcap(U)$. Let $|v, u|$ denote the least number of edges required to move from vertex v to vertex u in the graph. We define $|v, u| = \infty$ if u is unreachable from v. For each vertex $u \in V$, we define an ordering $\prec_u \subseteq V \times V$ on vertices, that intuitively orders vertices based on their proximity to u, with a subjugate role for the vertex ordering $\sqsubset$:

$$v \prec_u v' \text{ iff } |v, u| < |v', u| \text{ or } (|v, u| = |v', u| \text{ and } v \sqsubset v')$$

Observe that $u \prec_u v$ for all $v \neq u$. The minimal element of $U \subseteq V$ with respect to $\prec_u$ is written $\lambda_u(U)$.

3 Strong Bisimilarity and Stuttering Equivalence

Process theory studies refinement and equivalence relations, characterising the differences between models of systems that are observable to entities with different observational powers. Most equivalence relations have been studied for their computational complexity, giving rise to effective procedures for deciding these equivalences. Prominent equivalences are *strong bisimilarity*, due to Park [15] and *stuttering equivalence* [3], proposed by Browne, Clarke and Grumberg.

Game graphs share many of the traits of the system models studied in process theory. As such, it is natural to study refinement and equivalence relations for such graphs, see *e.g.*, *delayed simulation* [9]. In the remainder of this section, we recast the bisimilarity and stuttering equivalence to the setting of parity games, and show that these are finer than *winner equivalence*, which we define as follows.

Definition 2. *Let $\mathcal{G} = (V, \to, \Omega, \mathcal{P})$ be a parity game. Two vertices $v, v' \in V$ are said to be* winner equivalent*, denoted $v \sim_w v'$ iff v and v' are won by the same player.*

Because every vertex is won by exactly one player (see, *e.g.*, [22]), winner equivalence partitions V into a subset won by player 0 and a subset won by player 1. Clearly, winner equivalence is therefore an equivalence relation on the set of vertices of a given parity game. The problem of deciding winner equivalence, is in NP ∩ co-NP; all currently known algorithms require time exponential in the number of priorities in the game.

We next define strong bisimilarity for parity games; basically, we interpret the priorities and players of vertices as state labellings.

Definition 3. *Let $\mathcal{G} = (V, \rightarrow, \Omega, \mathcal{P})$ be a parity game. A symmetric relation $R \subseteq V \times V$ is a* strong bisimulation *relation if $v\ R\ v'$ implies*

- *$\Omega(v) = \Omega(v')$ and $\mathcal{P}(v) = \mathcal{P}(v')$;*
- *for all $w \in V$ such that $v \rightarrow w$, there should be a $w' \in V$ such that $v' \rightarrow w'$ and $w\ R\ w'$.*

Vertices v and v' are said to be strongly bisimilar*, denoted $v \sim v'$, iff a strong bisimulation relation R exists such that $v\ R\ v'$.*

Strong bisimilarity is an equivalence relation on the vertices of a parity game; quotienting with respect to strong bisimilarity is straightforward. It is not hard to show that strong bisimilarity is strictly finer than winner equivalence. Moreover, quotienting can be done effectively with a worst-case time complexity of $\mathcal{O}(|V| \log |V|)$.

Strong bisimilarity quotienting prior to solving a parity game can in some cases be quite competitive. One of the drawbacks of strong bisimilarity, however, is its sensitivity to counting (in the sense that it will not identify vertices that require a different number of steps to reach a next equivalence class), preventing it from compressing the game graph any further.

Stuttering equivalence shares many of the characteristics of strong bisimilarity, and deciding it has only a slightly worse worst-case time complexity. However, it is insensitive to counting, and is therefore likely to lead to greater reductions. Given these observations, we hypothesise (and validate this hypothesis in Section 4) that stuttering equivalence outperforms strong bisimilarity and, in most instances, reduces the time required for deciding winner equivalence in parity games stemming from verification problems.

We first introduce stuttering bisimilarity [14], a coinductive alternative to the stuttering equivalence of Browne, Clarke and Grumberg; we shall use the terms stuttering bisimilarity and stuttering equivalence interchangeably. The remainder of this section is then devoted to showing that stuttering bisimilarity is coarser than strong bisimilarity, but still finer than winner equivalence. The latter result allows one to pre-process a parity game by quotienting it using stuttering equivalence.

Definition 4. *Let $\mathcal{G} = (V, \rightarrow, \Omega, \mathcal{P})$ be a parity game. Let $R \subseteq V \times V$. An infinite path p is R-divergent, denoted $\mathsf{div}_R(p)$ iff $p_1\ R\ p_i$ for all i. Vertex $v \in V$ allows for divergence, denoted $\mathsf{div}_R(v)$ iff there is a path p such that $p_1 = v$ and $\mathsf{div}_R(p)$.*

We generalise the transition relation $\rightarrow$ to its reflexive-transitive closure, denoted $\Longrightarrow$, taking a given relation R on vertices into account. The generalised transition relation is used to define stuttering bisimilarity. Let $\mathcal{G} = (V, \rightarrow, \Omega, \mathcal{P})$ be a parity game and let $R \subseteq V \times V$ be a relation on its vertices. Formally, we define the relations $\rightarrow_R \subseteq V \times V$ and $\Longrightarrow_R \subseteq V \times V$ through the following set of deduction rules.

$$\frac{v \rightarrow w \qquad v\ R\ w}{v \rightarrow_R w} \qquad \frac{}{v \Longrightarrow_R v} \qquad \frac{v \rightarrow_R w \qquad w \Longrightarrow_R v'}{v \Longrightarrow_R v'}$$

We extend this notation to paths: we sometimes write $\langle v_1, \ldots, v_n \rangle \rightarrow u$ if $v_n \rightarrow u$; similarly, we write $\langle v_1, \ldots, v_n \rangle \rightarrow_R u$ and $\langle v_1, \ldots, v_n \rangle \Longrightarrow_R u$.

Definition 5. *Let $\mathcal{G} = (V, \rightarrow, \Omega, \mathcal{P})$ be a parity game. Let $R \subseteq V \times V$ be a symmetric relation on vertices; R is a* stuttering bisimulation *if $v\ R\ v'$ implies*

- *$\Omega(v) = \Omega(v')$ and $\mathcal{P}(v) = \mathcal{P}(v')$;*
- *$\mathsf{div}_R(v)$ iff $\mathsf{div}_R(v')$;*
- *If $v \rightarrow u$, then either ($v\ R\ u \wedge u\ R\ v'$), or there are u', w, such that $v' \Longrightarrow_R w \rightarrow u'$ and $v\ R\ w$ and $u\ R\ u'$;*

Two states v and v' are said to be stuttering bisimilar*, denoted $v \approx v'$ iff there is a stuttering bisimulation relation R, such that $v\ R\ v'$.*

Note that stuttering bisimilarity is the largest stuttering bisimulation. Moreover, stuttering bisimilarity is an equivalence relation, see *e.g.* [14,3]. In addition, quotienting with respect to stuttering bisimilarity is straightforward.

Stuttering bisimilarity between vertices extends naturally to finite paths. Paths of length 1 are equivalent if the vertices they consist of are equivalent. If paths p and q are equivalent, then $p \cdot \langle v \rangle \approx q$ iff v is equivalent to the last vertex in q (and analogously for extensions of q), and $p \cdot \langle v \rangle \approx q \cdot \langle w \rangle$ iff $v \approx w$. An infinite path p is equivalent to a (possibly infinite) path q if for all finite prefixes of p there is an equivalent prefix of q and *vice versa*.

We next set out to prove that stuttering bisimilarity is finer than winner equivalence. Our proof strategy is as follows: given that there is a strategy ϕ for player i from a vertex v, we define a strategy for player i that from vertices equivalent to v schedules only paths that are stuttering bisimilar to a path starting in v that is consistent with ϕ.

If after a number of moves a path p has been played, and our strategy has to choose the next move, then it needs to know which successors for p will yield a path for which again there is a stuttering bisimilar path that is consistent with ϕ. To this end we introduce the set $\mathsf{reach}_{\phi,v}(p)$.

Let ϕ be an arbitrary strategy, v an arbitrary vertex owned by the player for which ϕ defines the strategy, and let p be an arbitrary path. We define $\mathsf{reach}_{\phi,v}(p)$ as the set of vertices in new classes, reachable by traversing ϕ-consistent paths that start in v and that are stuttering bisimilar to p.

$$\mathsf{reach}_{\phi,v}(p) = \{u \in V \mid \exists q \in \Pi_\phi(v) : p \approx q \wedge \phi \Vdash q \cdot \langle u \rangle \wedge q \cdot \langle u \rangle \not\approx q\}$$

Observe that not all vertices in $\mathsf{reach}_{\phi,v}(p)$ have to be in the same equivalence class, because it is not guaranteed that all paths $q \in \Pi_\phi(v)$, stuttering bisimilar to p, are extended by ϕ towards the same equivalence class.

Suppose the set $\mathsf{reach}_{\phi,v}(p)$ is non-empty; in this case, our strategy should select a *target class* to which p should be extended. Because stuttering bisimilar vertices can reach the same classes, it does not matter which class present in $\mathsf{reach}_{\phi,v}(p)$ is selected as the target class. We do however need to make a unique choice; to this end we use the total ordering $\sqsubset$ on vertices.

$$\mathsf{targetclass}_{\phi,v}(p) = \{u \in V \mid u \approx \sqcap(\mathsf{reach}_{\phi,v}(p))\}$$

Not all vertices in the target class need be reachable from p, but there must exist at least one vertex that is. We next determine a *target vertex*, by selecting a unique, reachable vertex from the target class. This target of p, given a strategy ϕ and a vertex v is denoted $\tau_{\phi,v}(p)$; note that the ordering $\sqsubset$ is again used to uniquely determine a vertex from the set of reachable vertices.

$$\tau_{\phi,v}(p) = \sqcap\{u \in \mathsf{targetclass}_{\phi,v}(p) \mid \exists w \in V : p \Longrightarrow_\approx w \to u\}$$

Definition 6. *We define a strategy* $\mathsf{mimick}_{\phi,v}$ *for player* i *that, given some strategy* ϕ *for player* i *and a vertex* v*, allows only paths to be scheduled that have a stuttering bisimilar path starting in* v *that is scheduled by* ϕ*. It is defined as follows.*

$$\mathsf{mimick}_{\phi,v}(p) = \begin{cases} \lambda_t\{u \in V \mid p \to_\approx u\}, & t = \tau_{\phi,v}(p) \\ & p \not\to \tau_{\phi,v}(p) \\ & \mathsf{reach}_{\phi,v}(p) \neq \emptyset \\ \tau_{\phi,v}(p) & p \to \tau_{\phi,v}(p) \\ & \mathsf{reach}_{\phi,v}(p) \neq \emptyset \\ \sqcap\{u \in V \mid p \to_\approx u\}, & \mathsf{reach}_{\phi,v}(p) = \emptyset \end{cases}$$

Lemma 1. *Let* ϕ *be a strategy for player* i *in an arbitrary parity game. Assume that* $v, w \in V$ *and* $v \approx w$*, and let* $\psi = \mathsf{mimick}_{\phi,v}$*. Then*

$$\forall l \in \mathbb{N} : \forall p \in \Pi_\psi^{l+1}(w) : \ \exists k \in \mathbb{N} : \exists q \in \Pi_\phi^k(v) : \ p \approx q$$

Proof. We proceed by induction on l. For $l = 0$, the desired implication follows immediately. For $l = n + 1$, assume that we have a path $p \in \Pi_\psi^{n+1}(w)$. Clearly, $\langle p_1, \ldots, p_n \rangle$ is also consistent with ψ. The induction hypothesis yields us a $q \in \Pi_\phi^k(v)$ for some $k \in \mathbb{N}$ such that $\langle p_1, \ldots, p_n \rangle \approx q$. Let q be such. We distinguish the following cases:

1. $p_n \approx p_{n+1}$. In this case, clearly $p \approx \langle p_1, \ldots, p_n \rangle \approx q$, which finishes this case.
2. $p_n \not\approx p_{n+1}$. We again distinguish two cases:
   (a) Case $\mathcal{P}(p_n) \neq i$. Since $p_n \approx q_k$, we find that there must be states $u, w \in V$ such that $q_k \Longrightarrow_\approx w \to u$ and $p_{n+1} \approx u$. So there must be a path r and vertex u such that $p \approx q \cdot r \cdot \langle u \rangle$, for which we know that $r \approx q_k$. Therefore,

all vertices in r are owned by $\mathcal{P}(q_k) = \mathcal{P}(p_n)$, so ϕ is not defined for the extensions of q along p. We can therefore conclude that $\phi \Vdash q \cdot r \cdot \langle u \rangle$.

(b) Case $\mathcal{P}(p_n) = i$. Then it must be the case that $p_{n+1} = \tau_{\phi,v}(\langle p_1, \ldots, p_n \rangle)$. By definition, that means that there is a ϕ-consistent path $r \in \Pi_\phi(v)$, such that $r \approx p$. □

In the following lemma we extend the above obtained result to infinite paths.

Lemma 2. *Let ϕ be a strategy for player i in an arbitrary parity game. Assume that $v, w \in V$ and $v \approx w$, and let $\psi = \mathsf{mimick}_{\phi,v}$. Then*

$$\forall p \in \Pi^\omega_\psi(w) : \exists q \in \Pi^\omega_\phi(v) : p \approx q.$$

Proof. Suppose we have an infinite path $p \in \Pi^\omega_\psi(w)$. Using Lemma 1 we can obtain a path q starting in v that is stuttering bisimilar, and that is consistent with ϕ. The lemma does not guarantee, however, that q is of infinite length. We show that if q is finite, it can always be extended to an infinite path that is still consistent with ϕ.

Notice that paths can be partitioned into subsequences of vertices from the same equivalence class, and that two stuttering bisimilar paths must have the same number of partitions. This also follows from the original definition of stuttering equivalence given in [3].

Suppose now that q is of finite length, say $k + 1$. Then p must contain such a partition that has infinite size. In particular, there must be some $n \in \mathbb{N}$ such that $p_{n+j} \approx p_{n+j+1}$ for all $0 \leq j \leq |V|$. We distinguish two cases.

1. $\mathcal{P}(p_n) = i$. We show that then also $\mathsf{reach}_{\phi,v}(\langle p_0, p_1, \ldots p_n \rangle) = \emptyset$. Suppose this is not the case. Then we find that for some $u \in V$, $u = \tau_{\phi,v}(p)$ exists, and therefore $p_{n+j} \prec_u p_{n+j+1}$ for all $j \leq |V|$. Since $\prec_u$ is total, this means that the longest chain is of length $|V|$, which contradicts our assumptions. So, necessarily $\mathsf{reach}_{\phi,v}(\langle p_0, p_1, \ldots p_n \rangle) = \emptyset$, meaning that no path that is consistent with ϕ leaves the class of p_n. But this means that the infinite path that stays in the class of p_n is also consistent with ϕ.
2. $\mathcal{P}(p_n) \neq i$. Since $p_n \approx q_k$, also $\mathcal{P}(q_k) \neq i$. Since $p_n \approx p_{n+j}$ for all $j \leq |V|+1$, this means that there is a state u, such that $u = p_{n+l} = p_{n+l'}$. But this means that u is divergent. Since $\mathcal{P}(u) \neq i$, and $u \approx q_k$, we find that also q_k is divergent. Therefore, there is an infinite path with prefix q that is consistent with ϕ and that is stuttering bisimilar to p. □

Theorem 1. *Stuttering bisimilarity is strictly finer than winner equivalence, i.e., $\approx \subseteq \sim_w$.*

Proof. The claim follows immediately from Lemma 2 and the fact that two stuttering bisimilar infinite paths have the same infinitely occurring priorities. Strictness is immediate. □

Note that strong bisimilarity is strictly finer than stuttering bisimilarity; as a result, it immediately follows that strong bisimilarity is finer than winner equivalence, too.

As an aside, we point out that our proof of the above theorem relies on the construction of the strategy $\mathsf{mimick}_{\phi,v}$; its purpose, however, exceeds that of the proof. If, by solving the stuttering bisimilar quotient of a given parity game $\mathcal{G}$, one obtains a winning strategy ϕ for a given player, $\mathsf{mimick}_{\phi,v}$ defines the winning strategies for that player in $\mathcal{G}$. This is of particular importance in case an explanation of the solution of the game is required, for instance when the game encodes a verification problem for which a strategy helps explain the outcome of the verification (see *e.g.* [18]). It is not immediately obvious how a similar feature could be obtained in the setting of, say, the delayed simulations of Fritz and Wilke [9], because vertices that belong to different players and that have different priorities can be identified through such simulations.

4 Experiments

We next study the effect that stuttering equivalence minimisation has in a practical setting. We do this by solving parity games that originate from three different sources (we will explain more later) using three different methods: direct solving, solving after bisimulation reduction and solving after stuttering equivalence reduction. Parity games are solved using a number of different algorithms, *viz.* a naive C++ implementation of the *small progress measures* algorithm [11] due to Jurdziński, and the optimised and unoptimised variants that are implemented in the PGSolver tool [7] of the small progress measures algorithm, the recursive algorithm due to McNaughton [13], the bigstep algorithm due to Schewe [16] and a strategy improvement algorithm due to Vöge [20]. We compare the time needed by these methods to solve the parity games, and we compare the sizes of the parity games that are sent to the solving algorithms.

To efficiently compute bisimulation and stuttering equivalence for parity games we adapted a single-threaded implementation of the corresponding reduction algorithms by Blom and Orzan [2] for labelled transition systems.

All experiments were conducted on a machine consisting of 28 Intel® Xeon® E5520 Processors running at 2.27GHz, with 1TB of shared main memory, running a 64-bit Linux distribution using kernel version 2.6.27. None of our experiments employ multi-core features.

4.1 Test Sets

The parity games that were used for our experiments are partitioned into three test sets, of which we give a brief description below.

Test set 1. Our main interest is in the practical implications of stuttering equivalence reduction on solving model checking problems, so a number of typical model checking problems have been selected and encoded into parity games.

Five properties of the Firewire Link-Layer protocol (1394) [12] were considered, as they are described in [17]. They are numbered I–V in the order in which they can be found in that document.

Four properties are checked on the specification of a lift in [10]; a liveness property (I), a property that expresses the absence of deadlock (II) and two safety

properties (III and IV). These typical model checking properties are expressed as alternation-free µ-calculus formulae.

On a model of the sliding window protocol [1], a fairness property (I) and a safety property (II) are verified, as well as 7 other fairness, liveness and safety properties.

Note that some of the properties are described by alternation free µ-calculus formulae, whereas others use alternation. The parity games induced by the alternation free µ-calculus formulae have different numbers of priorities, but the priorities along the paths in the parity games are ascending. In contrast, the paths in the parity games induced by alternating properties have no such property and are therefore computationally more challenging. Note that the parity games generated for these problems only have limited alternations between vertices owned by player 0 and 1 in the paths of the parity games.

Test set 2. The second test set was taken from [7] and consists of several instances of the elevator problem and the Hanoi towers problem described in that paper. For the latter, a different encoding was devised and added to the test set.

Test set 3. This test set consists of a number of equivalence checking problems encoded into parity games as described in [4].

The problems taken from [7], as well as some of the equivalence checking problems, give rise to parity games with alternations between both players and priorities.

4.2 Results

To analyse the performance of stuttering equivalence reduction, we measured the number of vertices and the number of edges in the original parity games, the bisimulation-reduced parity games and the stuttering-reduced parity games. Some of the results for test set 1 are shown in Table 1. For the Elevator model from [7], the results are shown in Table 2.

Figure 1.a compares these sizes (and those not shown in the tables) graphically; each plot point represents a parity game, of which the position along the y-axis is determined by its stuttering-reduced size, and the position along the x-axis by its original size and its bisimulation-reduced size, respectively. The plotted sizes are the sum of the number of vertices and the number of edges.

In addition to these results, we measured the time needed to reduce and to solve the parity games. The time needed to solve a parity game using stuttering equivalence or bisimulation reduction is computed as the time needed to reduce the parity game, plus the time needed to solve the reduced game. Also, the time needed to solve these games directly was measured. The solving time for a game is the time that the *fastest* of the solving algorithms needs to solve it. The results are plotted in Figure 1.b. Again, every data point is a parity game, of which the solving times determine the position in the scatter plot.

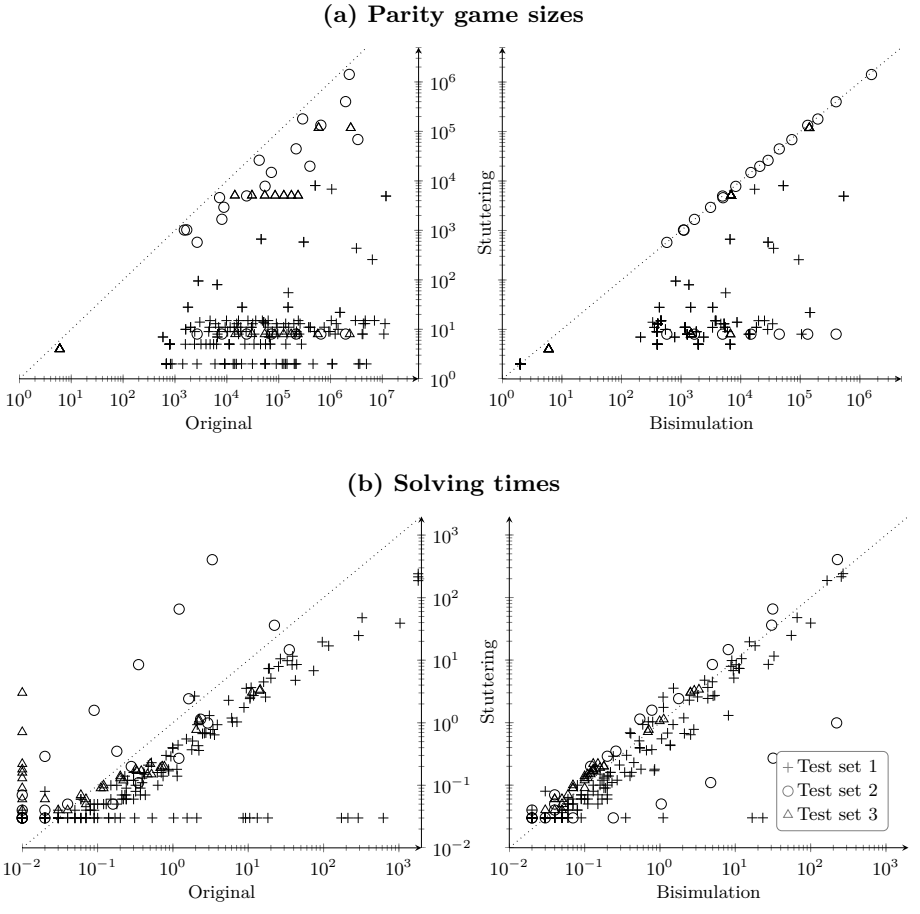

Fig. 1. Sizes and solving times (in seconds) of the stuttering-reduced parity games set out against sizes and solving times of the original games and of the bisimulation-reduced games. The vertical axis is shared between the plots in each subfigure. The dotted line is defined as $x = y$ and serves as a reference. Note that axes are in log scale.

4.3 Discussion

At a glance, stuttering reduction seems a big improvement on bisimulation reduction in terms of size reduction. Figure 1.a shows clearly that stuttering equivalence gives a better size reduction than bisimulation equivalence in the majority of cases. The difference is often somewhere between a factor ten and a factor thousand. Looking at solving times, the results also seem promising. In Figure 1.b we see that in most cases reducing the game and then solving it costs significantly less time. We will discuss the results in more detail for each test set separately.

Table 1. Statistics for the parity games for experiments from test set 1. In the Lift case, N denotes the number of distributed lifts; in the case of SWP, N denotes the size of the window. The number of priorities in the original (and minimised) parity games is listed under Priorities.

IEEE 1394		**original**		$\simeq$		$\sim$	
Property	**Priorities**	$\|V\|$	$\|\rightarrow\|$	$\|V\|$	$\|\rightarrow\|$	$\|V\|$	$\|\rightarrow\|$
I	1	346 173	722 422	1	1	1	1
II	1	377 028	679 157	3 730	3 086	5 990	11 180
III	4	1 190 395	2 025 022	102	334	13 551	22 166
IV	2	524 968	875 296	4	6	10 814	17 590
V	1	1 295 249	2 150 590	1	1	1	1

Lift			**original**		$\simeq$		$\sim$	
Property	**Priorities**	**N**	$\|V\|$	$\|\rightarrow\|$	$\|V\|$	$\|\rightarrow\|$	$\|V\|$	$\|\rightarrow\|$
I	4	2	1 691	4 825	22	58	333	1 021
I	4	3	63 907	240 612	131	450	5 148	23 703
I	4	4	1 997 579	9 752 561	929	4 006	74 059	462 713
II	2	2	846	2 172	5	9	94	240
II	2	3	31 954	121 625	16	39	1 092	4 514
II	2	4	998 790	5 412 890	64	193	14 353	80 043
III	1	2	763	1 903	1	1	1	1
III	1	3	26 996	99 348	1	1	1	1
III	1	4	788 879	4 146 139	1	1	1	1
IV	2	2	486	1 126	4	6	151	396
IV	2	3	11 977	39 577	5	9	1 741	6 951
IV	2	4	267 378	1 257 302	7	15	23 526	122 230

SWP			**original**		$\simeq$		$\sim$	
Property	**Priorities**	**N**	$\|V\|$	$\|\rightarrow\|$	$\|V\|$	$\|\rightarrow\|$	$\|V\|$	$\|\rightarrow\|$
I	3	1	1 250	3 391	4	7	314	849
I	3	2	14 882	47 387	4	7	1 322	4 127
I	3	3	84 866	291 879	4	7	4 190	14 153
I	3	4	346 562	1 246 803	4	7	11 414	40 557
II	2	1	1 370	4 714	5	8	90	316
II	2	2	54 322	203 914	5	8	848	3 789
II	2	3	944 090	3 685 946	5	8	5 704	28 606
II	2	4	11 488 274	45 840 722	5	8	34 359	183 895

Test set 1. For these cases, we see that the size reduction is always better than that of bisimulation reduction, unless bisimulation already compressed the parity game to a single state. Solving times using stuttering equivalence are in general better than those of direct solving.

The experiments indicate that minimising parity games using stuttering equivalence before solving the reduced parity games is at least as fast as directly solving the original games.

Table 2. Statistics for the parity games for the FIFO and LIFO Elevator models taken from [7]. **Floors** indicates the number of floors.

Elevator Models			**original**		$\eqsim$		$\sim$	
Model	**Floors**	**Priorities**	$\|V\|$	$\|\rightarrow\|$	$\|V\|$	$\|\rightarrow\|$	$\|V\|$	$\|\rightarrow\|$
FIFO	3	3	564	950	351	661	403	713
FIFO	4	3	2 688	4 544	1 588	2 988	1 823	3 223
FIFO	5	3	15 684	26 354	9 077	16 989	10 423	18 335
FIFO	6	3	108 336	180 898	62 280	116 044	71 563	125 327
FIFO	7	3	861 780	1 431 610	495 061	919 985	569 203	994 127
LIFO	3	3	588	1 096	326	695	363	732
LIFO	4	3	2 832	5 924	866	2 054	963	2 151
LIFO	5	3	16 356	38 194	2 162	5 609	2 403	5 850
LIFO	6	3	111 456	287 964	5 186	14 540	5 763	15 117
LIFO	7	3	876 780	2 484 252	16 706	51 637	18 563	53494

The second observation we make is that stuttering equivalence reduces the size quite well for this test set, when compared to the other sets. This may be explained by the way in which the parity games were generated. As they encode a μ-calculus formula together with a state space, repetitive and deterministic parts of the state space are likely to generate fragments within the parity game that can be easily compressed using stuttering reduction.

Lastly, we observe that solving times using bisimulation reduction are not in general much worse than those using stuttering reduction. The explanation is simple: both reductions compress the original parity game to such an extent that the resulting game is small enough for the solvers to solve it in less than a tenth of a second.

Test set 2. Both stuttering equivalence and strong bisimulation reduction perform poorly on a reachability property for the Hanoi towers experiment, with the reduction times vastly exceeding the times required for solving the parity games directly. A closer inspection reveals that this is caused by an unfortunate choice for a new priority for vertices induced by a fixpoint-free subformula. As a result, all paths in the parity game have alternating priorities with very short stretches of the same priorities, because of which hardly any reduction is possible. We included an encoding of the same problem which does not contain the unfortunate choice, and indeed observe that in that case stuttering equivalence does speed up the solving process.

The LIFO Elevator problem shows results similar to those of the other model checking problems. The performance with respect to the FIFO Elevator however is rather poor. This seems to be due to three main factors: the relatively large number of alternating fixed point signs, the alternations between vertices owned by player 0 and vertices owned by player 1, and the low average branching degree in the parity game. This indicates that for alternating μ-calculus formulae with nested conjunctive and disjunctive subformulae, stuttering equivalence reduction generally performs suboptimal. This should not come as a surprise, as stuttering

equivalence only allows one to compress sequences of vertices with equal priorities and owned by the same player.

Test set 3. The results for these experiments indicate that reduction using stuttering equivalence sometimes performs poorly. The subset where performance is especially poor is an encoding of branching bisimilarity, which gives rise to parity games with alternations both between different priorities as well as different players. As a result, little reduction is possible.

5 Conclusions

We have adapted the notion of stuttering bisimilarity to the setting of parity games, and proven that this equivalence relation can be safely used to minimise a parity game before solving the reduced game.

Experiments were conducted to investigate the effect of quotienting stuttering bisimilarity on parity games originating from model checking problems. In many practical cases this reduction leads to an improvement in solving time, however in cases where the parity games involved have many alternations between odd and even vertices, stuttering bisimilarity reduction performs only marginally better than strong bisimilarity reduction. Although we did compare our techniques against a number of competitive parity game solvers, using other solving algorithms, or even other implementations of the same algorithms, may give different results.

The fact that stuttering bisimilarity does not deal at all well with alternation leads us to believe that weaker notions of bisimilarity, in which vertices with different players can be related under certain circumstances, may resolve the most severe performance problems that we saw in our experiments. We regard the investigation of such weaker relations as future work.

Stuttering bisimilarity has been previously studied in a distributed setting [2]. It would be interesting to compare its performance to a distributed implementation of the known solving algorithms for parity games. However, we are only aware of a multi-core implementation of the *Small Progress Measures* algorithm [19].

References

1. Badban, B., Fokkink, W., Groote, J.F., Pang, J., van de Pol, J.: Verification of a sliding window protocol in μCRL and PVS. Formal Aspects of Computing 17, 342–388 (2005)
2. Blom, S., Orzan, S.: Distributed branching bisimulation reduction of state spaces. ENTCS 89(1) (2003)
3. Browne, M.C., Clarke, E.M., Grumberg, O.: Characterizing finite Kripke structures in propositional temporal logic. Theor. Comput. Sci. 59, 115–131 (1988)
4. Chen, T., Ploeger, B., van de Pol, J., Willemse, T.A.C.: Equivalence checking for infinite systems using parameterized boolean equation systems. In: Caires, L., Vasconcelos, V.T. (eds.) CONCUR 2007. LNCS, vol. 4703, pp. 120–135. Springer, Heidelberg (2007)

5. Cranen, S., Keiren, J.J.A., Willemse, T.A.C.: Stuttering equivalence for parity games, arXiv:1102.2366 [cs.LO] (2011)
6. Emerson, E.A., Jutla, C.S.: Tree automata, mu-calculus and determinacy. In: SFCS 1991, Washington, DC, USA, pp. 368–377. IEEE Computer Society, Los Alamitos (1991)
7. Friedmann, O., Lange, M.: Solving parity games in practice. In: Liu, Z., Ravn, A.P. (eds.) ATVA 2009. LNCS, vol. 5799, pp. 182–196. Springer, Heidelberg (2009)
8. Fritz, C.: Simulation-Based Simplification of omega-Automata. PhD thesis, Christian-Albrechts-Universität zu Kiel (2005)
9. Fritz, C., Wilke, T.: Simulation relations for alternating parity automata and parity games. In: Ibarra, O.H., Dang, Z. (eds.) DLT 2006. LNCS, vol. 4036, pp. 59–70. Springer, Heidelberg (2006)
10. Groote, J.F., Pang, J., Wouters, A.G.: Analysis of a distributed system for lifting trucks. J. Log. Algebr. Program 55, 21–56 (2003)
11. Jurdziński, M.: Small progress measures for solving parity games. In: Reichel, H., Tison, S. (eds.) STACS 2000. LNCS, vol. 1770, pp. 290–301. Springer, Heidelberg (2000)
12. Luttik, S.P.: Description and formal specification of the link layer of P1394. In: Workshop on Applied Formal Methods in System Design, pp. 43–56 (1997)
13. McNaughton, R.: Infinite games played on finite graphs. Annals of Pure and Applied Logic 65(2), 149–184 (1993)
14. De Nicola, R., Vaandrager, F.W.: Three logics for branching bisimulation. J. ACM 42(2), 458–487 (1995)
15. Park, D.: Concurrency and automata on infinite sequences. Theor. Comput. Sci. 104, 167–183 (1981)
16. Schewe, S.: Solving parity games in big steps. In: Arvind, V., Prasad, S. (eds.) FSTTCS 2007. LNCS, vol. 4855, pp. 449–460. Springer, Heidelberg (2007)
17. Sighireanu, M., Mateescu, R.: Verification of the Link Layer Protocol of the IEEE-1394 Serial Bus (FireWire): An Experiment with E-LOTOS. STTT 2(1), 68–88 (1998)
18. Stevens, P., Stirling, C.: Practical model checking using games. In: Steffen, B. (ed.) TACAS 1998. LNCS, vol. 1384, pp. 85–101. Springer, Heidelberg (1998)
19. van de Pol, J., Weber, M.: A multi-core solver for parity games. ENTCS 220(2), 19–34 (2008)
20. Vöge, J., Jurdziński, M.: A discrete strategy improvement algorithm for solving parity games. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, pp. 202–215. Springer, Heidelberg (2000)
21. Wimmer, R., Herbstritt, M., Hermanns, H., Strampp, K., Becker, B.: Sigref— a symbolic bisimulation tool box. In: Graf, S., Zhang, W. (eds.) ATVA 2006. LNCS, vol. 4218, pp. 477–492. Springer, Heidelberg (2006)
22. Zielonka, W.: Infinite games on finitely coloured graphs with applications to automata on infinite trees. Theor. Comp. Sci. 200(1-2), 135–183 (1998)