A Super-Polynomial Lower Bound for the Parity Game Strategy Improvement Algorithm as We Know it

Oliver Friedmann

Dept. of Computer Science, University of Munich (LMU)

February 26, 2009
Parity Games
Parity Games

parity game $G = (V, E, \Omega)$ with

- $(V, E)$ directed total graph
- $V = V_0 \cup V_1$ with $V_0 \cap V_1 = \emptyset$
- $\Omega : V \to \mathbb{N}$ injective priority function

interpretation:

- players 0 and 1 move token along edges
- player $i$ moves in $V_i$
- play $= \text{infinite sequence of nodes}$
- parity of greatest priority seen infinitely often determines winner
strategy for player $i$ is $\sigma_i : V_i \rightarrow V$ respecting edges

$v$-winning strategy is $\sigma_i$ s.t. player $i$ wins any play conforming to $\sigma_i$ starting in $v$
strategy for player $i$ is $\sigma_i : V_i \to V$ respecting edges

$v$-winning strategy is $\sigma_i$ s.t. player $i$ wins any play conforming to $\sigma_i$ starting in $v$

given parity game $G = (V, E, \Omega)$, define

$$W_i := \{ v \in V \mid \text{player } i \text{ has } v\text{-winning strategy starting in } v \}$$
strategy for player $i$ is $\sigma_i : V_i \rightarrow V$ respecting edges

$v$-winning strategy is $\sigma_i$ s.t. player $i$ wins any play conforming to $\sigma_i$ starting in $v$

given parity game $G = (V, E, \Omega)$, define

$$W_i := \{v \in V \mid \text{player } i \text{ has } v\text{-winning strategy starting in } v\}$$

decision problem “solving”: given $G = (V, E, \Omega)$ compute $W_0, W_1$ and associated winning strategies
The Complexity of Solving Parity Games

Theorem 1

Solving a parity game is in $NP \cap co-NP$. 
The Complexity of Solving Parity Games

Theorem 1

Solving a parity game is in $NP \cap co-NP$.

unlikely to be NP-complete, but is it in $P$?

widely believed; no proof so far
## The Complexity of Solving Parity Games

**Theorem 1**  
Solving a parity game is in \( NP \cap co-NP \).

unlikely to be \( NP \)-complete, but is it in \( P \)?

widely believed; no proof so far

**Theorem 2 (Jurdziński’98)**  
Solving a parity game is in \( UP \cap co-UP \).
Strategy Improvement
Introduction

origin Jurdziński/Vöge’00

Components

- **Initial strategy** $\sigma_{init}$ for player 0
Introduction

origin Jurdziński/Vöge’00

Components

- Initial strategy $\sigma_{init}$ for player 0
- Each player 0 strategy $\sigma$ induces a valuation $\Xi_\sigma$
Introduction

origin Jurdziński/Vöge’00

Components

- **Initial strategy** $\sigma_{init}$ for player 0
- Each player 0 strategy $\sigma$ induces a **valuation** $\Xi_\sigma$
- Partial ordering $\triangleleft$ on valuations
**Introduction**

origin Jurdziński/Vöge’00

**Components**

- **Initial strategy** $\sigma_{init}$ for player 0
- Each player 0 strategy $\sigma$ induces a **valuation** $\Xi_\sigma$
- **Partial ordering** $\prec$ on valuations
- **Improvement policy** $I_G$ selects a successor strategy for any $\sigma$
Introduction

origin Jurdziński/Vöge’00

Components

- **Initial strategy** $\sigma_{\text{init}}$ for player 0
- Each player 0 strategy $\sigma$ induces a valuation $\Xi_\sigma$
- Partial ordering $\triangleleft$ on valuations
- **Improvement policy** $\mathcal{I}_G$ selects a successor strategy for any $\sigma$
- Crucial: $\Xi_\sigma \trianglelefteq \Xi_{\mathcal{I}_G(\sigma)}$ for all $\sigma$. 
Introduction

origin Jurdziński/Vöge’00

Components

- **Initial strategy** $\sigma_{\text{init}}$ for player 0
- Each player 0 strategy $\sigma$ induces a valuation $\Xi_\sigma$
- Partial ordering $\prec$ on valuations
- **Improvement policy** $\mathcal{I}_G$ selects a successor strategy for any $\sigma$
- Crucial: $\Xi_\sigma \preceq \Xi_{\mathcal{I}_G(\sigma)}$ for all $\sigma$.
- Maximal strategy induces winning sets
The Algorithm

1: $\sigma \leftarrow \sigma_{\text{init}}$
2: while $\sigma$ is improvable do
3: $\sigma \leftarrow I_G(\sigma)$
4: end while
5: return induced winning sets
Node Orderings

Relevance Ordering

\[ v < w \iff \Omega(v) < \Omega(w) \]
Node Orderings

Relevance Ordering

\[ v < w \iff \Omega(v) < \Omega(w) \]

Reward Ordering

\[ v \prec w \iff rew(v) < rew(w) \]

where

- \( rew(v) = \Omega(v) \) for even \( \Omega(v) \)
- \( rew(v) = -\Omega(v) \) for odd \( \Omega(v) \)

Define \( V_\oplus := \{ v \mid rew(v) \geq 0 \} \) and \( V_\ominus := \{ v \mid rew(v) < 0 \} \).
Paths, Cycles

Loopless Path

Loopless Path of length \( k \): \( \pi = v_0 \ldots v_{k-1} \)

- conforming with \( E \)
- containing no cycle
## Paths, Cycles

### Loopless Path

Loopless Path of length $k$: $\pi = v_0 \ldots v_{k-1}$
- conforming with $E$
- containing no cycle

### Dominating Cycle Node

Node $w$ is dominating cycle node iff there is a cycle $c = v_0 \ldots v_{k-1}$ with $v_i \leq w$ for all $i$. 
###  Valuations

#### Node Valuation

Node valuation for node $v_0$:

$$(v_k, \{ v_i \mid v_k < v_i \}, k)$$

s.t.

- there is a loopless path $\pi = v_0 \ldots v_k$ and
Valuations

Node Valuation

Node valuation for node $v_0$:

$$(v_k, \{v_i \mid v_k < v_i\}, k)$$

s.t.

- there is a loopless path $\pi = v_0 \ldots v_k$ and
- $v_k$ is a dominating cycle node
Valuations

**Node Valuation**

Node valuation for node $v_0$:

$$(v_k, \{v_i \mid v_k < v_i\}, k)$$

s.t.

- there is a loopless path $\pi = v_0 \ldots v_k$ and
- $v_k$ is a dominating cycle node

**Game Valuation**

A game valuation is a map $\Xi$ assigning to each node $v$ a node valuation.
Comparing Node V.

Comparing Node Valuation

It holds that

\[(u, M, e) \prec (v, N, f)\]

if and only if one of the following cases applies:
Comparing Node V.

Comparing Node Valuation

It holds that

\[(u, M, e) \prec (v, N, f)\]

if and only if one of the following cases applies:

- \[u \prec v\]
Comparing Node V.

Comparing Node Valuation

It holds that

\[(u, M, e) \prec (v, N, f)\]

if and only if one of the following cases applies:

• \( u \prec v \)

• \( u = v \land M \triangle N \neq \emptyset \land \max_{<}(M \triangle N) \in N \cap V_{\oplus} \)
Comparing Node V.

Comparing Node Valuation

It holds that

\[(u, M, e) \prec (v, N, f)\]

if and only if one of the following cases applies:

- \(u \prec v\)
- \(u = v \land M \triangle N \neq \emptyset \land \max_{<} (M \triangle N) \in N \cap V_{\oplus}\)
- \(u = v \land M \triangle N \neq \emptyset \land \max_{<} (M \triangle N) \in M \cap V_{\ominus}\)
Comparing Node V.

### Comparing Node Valuation

It holds that

\[(u, M, e) \prec (v, N, f)\]

if and only if one of the following cases applies:

- \[u \prec v\]
- \[u = v \land M \triangle N \neq \emptyset \land \max_{<}(M \triangle N) \in N \cap V_{\oplus}\]
- \[u = v \land M \triangle N \neq \emptyset \land \max_{<}(M \triangle N) \in M \cap V_{\ominus}\]
- \[u = v \land M = N \land e < f \land u \in V_{\ominus}\]
Comparing Node V.

### Comparing Node Valuation

It holds that

\[(u, M, e) \prec (v, N, f)\]

if and only if one of the following cases applies:

- \(u \prec v\)
- \(u = v \land M \triangle N \neq \emptyset \land \max_{<} (M \triangle N) \in N \cap V_{\oplus}\)
- \(u = v \land M \triangle N \neq \emptyset \land \max_{<} (M \triangle N) \in M \cap V_{\ominus}\)
- \(u = v \land M = N \land e < f \land u \in V_{\ominus}\)
- \(u = v \land M = N \land e > f \land u \in V_{\oplus}\)
Comparing Game Valuations

\[ \Xi \triangleleft \Xi' : \iff (\forall v \in V : \Xi(v) \leq \Xi'(v)) \land (\Xi \neq \Xi') \]
Valuating Strategies

**Induced Subgame**

A strategy $\sigma$ induces the *strategy subgame* $G|_\sigma := (V, V_0, V_1, E|_\sigma, \Omega)$ where

$$E|_\sigma := \{(u, v) \in E \mid u \in \text{dom}(\sigma) \Rightarrow \sigma(u) = v\}$$
Valuating Strategies

Induced Subgame

A strategy $\sigma$ induces the strategy subgame $G|_\sigma := (V, V_0, V_1, E|_\sigma, \Omega)$ where

$$E|_\sigma := \{(u, v) \in E \mid u \in \text{dom}(\sigma) \Rightarrow \sigma(u) = v\}$$

Induced Valuation

The game valuation $\Xi_\sigma$ is the $\triangleleft$-worst game valuation associated with $G|_\sigma$. 
Improving Strategies

Improvement Policy

An improvement policy maps a given player 0 strategy $\sigma$ to a successor strategy $\sigma'$ satisfying

$$\Xi_\sigma(\sigma(v)) \preceq \Xi_\sigma(\sigma'(v)) \quad \text{for all} \ v \in V_0$$

That is, an improvement policy is only allowed to select improvement edges.
Improving Strategies

Improvement Policy

An improvement policy maps a given player 0 strategy $\sigma$ to a successor strategy $\sigma'$ satisfying

$$\Xi_\sigma(\sigma(v)) \preceq \Xi_\sigma(\sigma'(v)) \quad \text{for all } v \in V_0$$

That is, an improvement policy is only allowed to select improvement edges.
Additionally, if there is at least one proper improvement edge, an improvement policy has to select at least one of them.
Improving Strategies

Improvement Policy

An improvement policy maps a given player 0 strategy $\sigma$ to a successor strategy $\sigma'$ satisfying

$$\Xi_\sigma(\sigma(v)) \preceq \Xi_\sigma(\sigma'(v)) \quad \text{for all } v \in V_0$$

That is, an improvement policy is only allowed to select improvement edges.

Additionally, if there is at least one proper improvement edge, an improvement policy has to select at least one of them.

Theorem 3

Let $G$ be a parity game, $\sigma$ be an improvable strategy and $\mathcal{I}_G$ be an improvement policy. Let $\sigma' = \mathcal{I}_G(\sigma)$. Then $\Xi_\sigma \triangleleft \Xi_{\sigma'}$. 
Policies

Locally Optimizing Policy

The locally optimizing policy selects a successor strategy \( \sigma' \) s.t.

\[
\Xi_{\sigma}(u) \leq \Xi_{\sigma}(\sigma'(v)) \quad \text{for all } v \in V_0 \text{ and all } (v, u) \in E
\]

origin Jurdziński/Vöge’00
Policies

**Locally Optimizing Policy**

The locally optimizing policy selects a successor strategy $\sigma'$ s.t.

$$\Xi_{\sigma}(u) \preceq \Xi_{\sigma}(\sigma'(v)) \quad \text{for all } v \in V_0 \text{ and all } (v, u) \in E$$

origin Jurdziński/Vöge’00

**Globally Optimizing Policy**

The globally optimizing policy selects a successor strategy $\sigma'$ s.t.

for all possible successor strategies $\sigma''$ it holds that

$$\Xi_{\sigma''} \preceq \Xi_{\sigma'}$$

origin Schewe’08
Initial Strategy

Heuristic Approach
E.g. select the successor with the best reward.
Initial Strategy

Heuristic Approach
E.g. select the successor with the best reward.

Probabilistic Approach
E.g. select successor randomly.
An Unsuccessful Approach
General Assumption

Conjecture

Strategy Improvement probably requires linearly many iterations in the worst case. And if not so, it is very likely that it requires polynomially many iterations in the worst case.

origin Jurkziński/Vöge’00, Schewe’08
General Assumption

Conjecture

Strategy Improvement probably requires linearly many iterations in the worst case. And if not so, it is very likely that it requires polynomially many iterations in the worst case.

origin Jurdziński/Vöge’00, Schewe’08

Known Instances

There were no known examples on which the strategy iteration required more than linearly many iterations (at least to our knowledge).
Searching Hard Instances

**Procedure**

Fix a game size $n$ and a lower bound $i$ on the number of iterations. It is possible to construct a predicate $P(n, i)$ in propositional logic that basically simulates a run of the strategy iteration on a game of size $n$ with at least $i$ iterations.
Searching Hard Instances

Procedure

Fix a game size $n$ and a lower bound $i$ on the number of iterations. It is possible to construct a predicate $P(n, i)$ in propositional logic that basically simulates a run of the strategy iteration on a game of size $n$ with at least $i$ iterations. Use a SAT solver to solve $P(n, i)$ and draw the following conclusions:

- **SAT**: There is a game of size $n$ that requires at least $i$ iterations. Such a game can be extracted using the returned variable assignment for $P(n, i)$.
- **UNSAT**: There is no game of size $n$ that requires at least $i$ iterations.
Searching Hard Instances (cont.)

Results

Up to \( n < 11 \) it holds that there are no games of size \( n \) that require more than \( n \) iterations.
Results

Up to $n < 11$ it holds that there are no games of size $n$ that require more than $n$ iterations.
For $n \geq 11$ and $i \geq n$ it holds that the SAT solver just didn’t terminate within in a reasonable amount of time.
Searching Hard Instances (cont.)

Results

Up to $n < 11$ it holds that there are no games of size $n$ that require more than $n$ iterations.

For $n \geq 11$ and $i \geq n$ it holds that the SAT solver just didn’t terminate within in a reasonable amount of time.

Since we are about to present a family of games on which the strategy iteration requires super-polynomially many iterations:
Searching Hard Instances (cont.)

Results

Up to $n < 11$ it holds that there are no games of size $n$ that require more than $n$ iterations.
For $n \geq 11$ and $i \geq n$ it holds that the SAT solver just didn’t terminate within in a reasonable amount of time.

Since we are about to present a family of games on which the strategy iteration requires super-polynomially many iterations:

Corollary to Murphy’s Law

As always, it is dangerous to base conclusions on empirical results.
Building Tough Games
Enforcing super-polynomially many iterations

Combinatorial Point Of View

A game valuation consists of linearly many node valuations. A node valuation consists of three components:

$$(v, M, e)$$
Enforcing super-polynomially many iterations

Combinatorial Point Of View

A game valuation consists of *linearly* many node valuations. A node valuation consists of three components:

\[(v, M, e)\]

- A node \(v\): *Linearly* many.
Enforcing super-polynomially many iterations

Combinatorial Point Of View

A game valuation consists of linearly many node valuations. A node valuation consists of three components:

\[(v, M, e)\]

- A node \(v\): Linearly many.
- A set of nodes \(M\): Exponentially many.
Enforcing super-polynomially many iterations

Combinatorial Point Of View

A game valuation consists of \textit{linearly} many node valuations. A node valuation consists of three components:

\[(v, M, e)\]

- A node \(v\): \textit{Linearly} many.
- A set of nodes \(M\): \textit{Exponentially} many.
- A number \(e < |V|\): \textit{Linearly} many.
Enforcing super-polynomially many iterations

Combinatorial Point Of View

A game valuation consists of **linearly** many node valuations. A node valuation consists of three components:

\[(v, M, e)\]

- A node \(v\): **Linearly** many.
- A set of nodes \(M\): **Exponentially** many.
- A number \(e < |V|\): **Linearly** many.

Therefore, a tough game needs to focus on the second component of node valuations in order to exploit the potentially exponentially many different sets of nodes.
Critical Graphs
A Deceleration Lane

Structure Properties

- Takes \textit{linearly} many iterations to be non-improvable
A Deceleration Lane

Structure Properties

- Takes *linearly* many iterations to be non-improvable
- Comprises a new *best-valued node* in each iteration
A Deceleration Lane

Structure Properties

- Takes **linearly** many iterations to be non-improvable
- Comprises a new **best-valued node** in each iteration
- Structure can be **reset** in one single iteration
A Deceleration Lane
A Deceleration Lane

Initial strategy maps each player 0 nodes to $x$. 
A Deceleration Lane

Improve $b_0$ to $d$. 
A Deceleration Lane

$a_0$ becomes new best-valued entry node.
A Deceleration Lane

Improve $b_1$ to $b_0$.
A Deceleration Lane

\[ \begin{align*}
& a_3 : 10 \\
& a_2 : 8 \\
& a_1 : 6 \\
& a_0 : 4 \\
& c : 11
\end{align*} \]

\[ \begin{align*}
& b_3 : 9 \\
& b_2 : 7 \\
& b_1 : 5 \\
& b_0 : 3 \\
& d : 12
\end{align*} \]

\[ \begin{align*}
& x : 14 \\
& t : 1 \\
& s : 0
\end{align*} \]

\[ a_1 \text{ becomes new best-valued entry node.} \]
A Deceleration Lane

Improve $b_2$ to $b_1$. 
A Deceleration Lane

\[ a_2 \] becomes new best-valued entry node.
A Deceleration Lane

Improve $b_3$ to $b_2$. 

\[ \begin{align*}
  a_3 &: 10 \\
  a_2 &: 8 \\
  a_1 &: 6 \\
  a_0 &: 4 \\
  c  &: 11 \\
  b_3 &: 9 \\
  b_2 &: 7 \\
  b_1 &: 5 \\
  b_0 &: 3 \\
  d  &: 12 \\
  x  &: 14 \\
  t  &: 1 \\
  s  &: 0
\end{align*} \]
A Deceleration Lane

\[ a_3 : 10 \]
\[ a_2 : 8 \]
\[ a_1 : 6 \]
\[ a_0 : 4 \]
\[ c : 11 \]
\[ b_3 : 9 \]
\[ b_2 : 7 \]
\[ b_1 : 5 \]
\[ b_0 : 3 \]
\[ d : 12 \]
\[ x : 14 \]
\[ t : 1 \]
\[ s : 0 \]

\( a_3 \) becomes new best-valued entry node; external event values \( s \) better than \( x \).
A Deceleration Lane

Improve all possible player 0 nodes to $s$. 
A Deceleration Lane

\[ a_3 : 10 \]
\[ a_2 : 8 \]
\[ a_1 : 6 \]
\[ a_0 : 4 \]
\[ c : 11 \]
\[ b_3 : 9 \]
\[ b_2 : 7 \]
\[ b_1 : 5 \]
\[ b_0 : 3 \]
\[ d : 12 \]
\[ x : 14 \]
\[ t : 1 \]
\[ s : 0 \]

\( a_3 \) remains best-valued entry node.
A Deceleration Lane

Improve $d$ to $t$. 
A Deceleration Lane

\[ a_3 : 10 \]
\[ a_2 : 8 \]
\[ a_1 : 6 \]
\[ a_0 : 4 \]
\[ c : 11 \]

\[ b_3 : 9 \]
\[ b_2 : 7 \]
\[ b_1 : 5 \]
\[ b_0 : 3 \]
\[ d : 12 \]

\[ x : 14 \]
\[ t : 1 \]
\[ s : 0 \]

\( c \) becomes new best-valued entry node; external event values \( x \) better than \( s \).
A Deceleration Lane

Improve all player 0 nodes to $x$. 
A Deceleration Lane

Initial setting is reached.
A Deceleration Lane (cont.)

Usage Benefits

- Able to absorb the update activity of other graph structures
A Deceleration Lane (cont.)

Usage Benefits

- Able to **absorb** the update activity of other graph structures
- **Reusable** due to its ability to be reset
Usage Example

to the Deceleration Lane
Usage Example

Initial strategy maps player 0 node to the first lane node.

to the Deceleration Lane
Usage Example

to the Deceleration Lane

Improve node to the next lane node.
Usage Example

to the Deceleration Lane
Usage Example

1

to the Deceleration Lane

2

Improve node to the next lane node.
Usage Example

to the Deceleration Lane
Usage Example

to the Deceleration Lane

Improve node to the next lane node.
Usage Example

to the Deceleration Lane
**Usage Example**

![Diagram of a graph with nodes 1 and 2 connected by arrows]

- **1** to the Deceleration Lane
- Improve node to move to the cycle.
Usage Example

Player 1 is forced to move out of the cycle.

to the Deceleration Lane
## Motivation

- We want to postpone the closing of a cycle
## A Stubborn Cycle

### Motivation

- We want to postpone the closing of a cycle
- A simple cycle can be postponed when applying the *locally* optimizing policy
A Stubborn Cycle

Motivation

- We want to postpone the closing of a cycle
- A simple cycle can be postponed when applying the locally optimizing policy
- Applying the globally optimizing policy, a simple cycle closes immediately
A Stubborn Cycle

Motivation

- We want to postpone the closing of a cycle
- A simple cycle can be postponed when applying the locally optimizing policy
- Applying the globally optimizing policy, a simple cycle closes immediately

Solution

Use a cycle consisting of more nodes instead s.t. there is always at least one edge belonging to the cycle that is not allowed to be chosen in one iteration.
A Stubborn Cycle
A Stubborn Cycle

Initial strategy maps two player 0 nodes out of the cycle.
One node is improved to move into, another is improved to move out of the cycle.
A Stubborn Cycle

Two player 0 nodes move out of the cycle.
A Stubborn Cycle

One node is improved to move into, another is improved to move out of the cycle.
A Stubborn Cycle

Two player 0 nodes move out of the cycle.
A Stubborn Cycle

One node is improved to move into, another is improved to move out of the cycle.
A Stubborn Cycle

Two player 0 nodes move out of the cycle.
A Stubborn Cycle

One node is improved to move into, another is improved to move out of the cycle.
A Stubborn Cycle

Two player 0 nodes move out of the cycle.
A Stubborn Cycle

$f$ is improved to move into the cycle.
A Stubborn Cycle

Only $g$ moves out of the cycle.
A Stubborn Cycle

$g$ is improved to move into the cycle.
A Stubborn Cycle

Player 1 is forced to move out of the cycle.
Lower Bound
Basic Idea

The Game $G_n$

- Implement a binary counter with $n$ bits: Each bit is represented by a stubborn cycle
Basic Idea

The Game $G_n$

- Implement a binary counter with $n$ bits: Each bit is represented by a stubborn cycle
- One single deceleration lane is used to absorb the update activity of open stubborn cycles
**Basic Idea**

#### The Game $G_n$

- Implement a binary counter with $n$ bits: Each bit is represented by a stubborn cycle.
- One single deceleration lane is used to absorb the update activity of open stubborn cycles.
- Stubborn cycles associated with lower bits have strictly less edges leading to the deceleration lane.
The Game $G_n$

- Implement a binary counter with $n$ bits: Each bit is represented by a stubborn cycle
- One single deceleration lane is used to absorb the update activity of open stubborn cycles
- Stubborn cycles associated with lower bits have strictly less edges leading to the deceleration lane
- Backbone structure ensures resetting the deceleration lane after closing a stubborn cycle
### The Game $G_n$

- Implement a binary counter with $n$ bits: Each bit is represented by a stubborn cycle.
- One single deceleration lane is used to absorb the update activity of open stubborn cycles.
- Stubborn cycles associated with lower bits have strictly less edges leading to the deceleration lane.
- Backbone structure ensures resetting the deceleration lane after closing a stubborn cycle.
- Backbone structure ensures reopening all lower cycles after closing a stubborn cycle.
The Game $G_2$

Whole graph consists of a simple cycle,
The Game $G_2$

A deceleration lane,
The Game $G_2$

stubborn cycles
The Game $G_2$

stubborn cycles that are connected to the lane,
The Game $G_2$

cycle associated structures,
The Game $G_2$

and additional access structure.
The Game $G_2$

Strategy: First stubborn cycle is closed, lane is reset.
Lane improves iteratively, second stubborn cycle is occupied thereby.
The Game $G_2$

Lane improves iteratively, second stubborn cycle is occupied thereby.
Lane improves iteratively, second stubborn cycle is occupied thereby.
The Game $G_2$

Lane improves iteratively, second stubborn cycle is occupied thereby.
A Super-Polynomial Lower Bound for the Parity Game Strategy Iteration

The Game $G_2$

Lane improves iteratively, second stubborn cycle is occupied thereby.
The Game $G_2$

Lane improves iteratively, second stubborn cycle is occupied thereby.
The Game $G_2$

Lane improves iteratively, second stubborn cycle is occupied thereby.
The Game $G_2$

Lane improves iteratively, second stubborn cycle is occupied thereby.
The Game $G_2$

Lane improves iteratively, second stubborn cycle is occupied thereby.
The Game $G_2$

Lane improves iteratively, second stubborn cycle is occupied thereby.
The Game $G_2$

Lane improves iteratively, second stubborn cycle is occupied thereby.
The Game $G_2$

Lane improves iteratively, second stubborn cycle is occupied thereby.
The Game $G_2$

Lane improves iteratively, second stubborn cycle is occupied thereby.
Lane improves iteratively, second stubborn cycle is occupied thereby.
Lane improves iteratively, second stubborn cycle is occupied thereby.
The Game $G_2$

Lane improves iteratively, second stubborn cycle is occupied thereby.
The Game $G_2$

Lane improves iteratively, second stubborn cycle is occupied thereby.
The Game $G_2$

Lane improves iteratively, second stubborn cycle is occupied thereby.
The Game $G_2$

Second cycle cannot improve furthermore to the lane.
Second cycles closes, forcing player 1 to leave it.
Now it is profitable to move to $k_1$. 

The Game $G_2$
The Game $G_2$

Now it is profitable to move to $k_1$. 
The lane resets, the first stubborn cycle opens.
The Game $G_2$

The lane resets, the first stubborn cycle opens.
The Game $G_2$

Lane improves iteratively, first stubborn cycle is occupied thereby.
The Game $G_2$

Lane improves iteratively, first stubborn cycle is occupied thereby.
Lane improves iteratively, first stubborn cycle is occupied thereby.
Lane improves iteratively, first stubborn cycle is occupied thereby.
The Game \( G_2 \)

Lane improves iteratively, first stubborn cycle is occupied thereby.
Lane improves iteratively, first stubborn cycle is occupied thereby.
Lane improves iteratively, first stubborn cycle is occupied thereby.
The Game $G_2$

Lane improves iteratively, first stubborn cycle is occupied thereby.
The Game $G_2$

Lane improves iteratively, first stubborn cycle is occupied thereby.
The Game $G_2$

Lane improves iteratively, first stubborn cycle is occupied thereby.
Benchmarks
Benchmarks
Fixing Strategy Improvement?
Fixing Strategy Improvement?

Possible Solution

Enrich improvement policies by reapplying sub-strategies that were profitable before.
The End