Playing General Structure Rewriting Games

Łukasz Kaiser
Joint work with Łukasz Stafiniak

Mathematische Grundlagen der Informatik
RWTH Aachen

AGI
Lugano, 2010

Motivation

A system is intelligent if it is more useful to talk about it in terms of goals than in terms of mechanisms.

Richard Sutton

Motivation

A system is intelligent if it is more useful to talk about it in terms of goals than in terms of mechanisms.

Richard Sutton

Mentioned Models of Computation (deterministic)

Turing Machines

- AIXI

Term Rewriting

- OpenCog

Graph Rewriting

- SOAR

Motivation

A system is intelligent if it is more useful to talk about it in terms of goals than in terms of mechanisms.

Richard Sutton

Mentioned Models of Computation (deterministic)

Turing Machines

- AIXI

Term Rewriting

- OpenCog

Graph Rewriting

- SOAR

General Structure Rewriting Games

- Non-determinism with **multiple players** (and probability)
- Hypergraph **rewriting** with constraints
- Continuous dynamics (by **ODE**) in the model

Motivation

> **A system is intelligent if it is more useful to talk about it in terms of goals than in terms of mechanisms.**
>
> *Richard Sutton*

Mentioned Models of Computation (deterministic)

Turing Machines

- AIXI

Term Rewriting

- OpenCog

Graph Rewriting

- SOAR

General Structure Rewriting Games

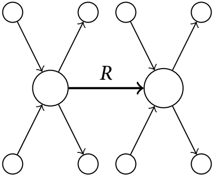
- Non-determinism with **multiple players** (and probability)
- Hypergraph **rewriting** with constraints
- Continuous dynamics (by **ODE**) in the model

Why a high-level model? Easier to program, debug, understand

Cf. programming languages, regular expressions, databases

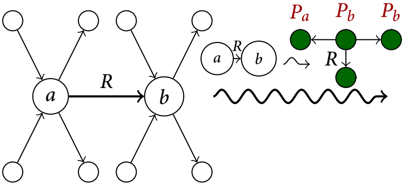
Structure Rewriting Rules

Rewriting Example



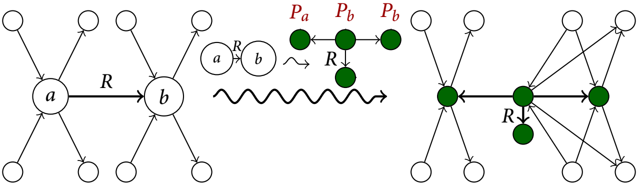
Structure Rewriting Rules

Rewriting Example



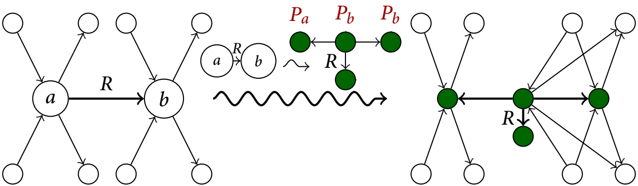
Structure Rewriting Rules

Rewriting Example



Structure Rewriting Rules

Rewriting Example



Embedding: σ is **injective** and $R_i^{\mathfrak{A}}(a_1, \dots, a_{r_i}) \Leftrightarrow R_i^{\mathfrak{B}}(\sigma(a_1), \dots, \sigma(a_{r_i}))$

$$\sigma \ : \quad \mathfrak{A} = (A, R_1^{\mathfrak{A}}, R_2^{\mathfrak{A}}, \dots, R_k^{\mathfrak{A}}) \quad \hookrightarrow \quad (B, R_1^{\mathfrak{B}}, R_2^{\mathfrak{B}}, \dots, R_k^{\mathfrak{B}}) = \mathfrak{B}$$

Rewriting: $\mathfrak{B} = \mathfrak{A}[\mathfrak{L} \to \mathfrak{R}/\sigma]$ iff $B = (A \setminus \sigma(L)) \dot\cup R$ and,

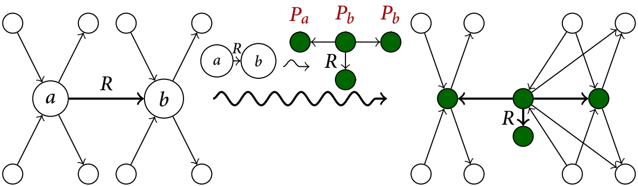
for $M = \{(r, a) \mid a = \sigma(l), r \in P_l^{\mathfrak{R}}$ for some $l \in L\} \cup \{(a, a) \mid a \in A\}$,

$(b_1, \dots, b_{r_i}) \in R_i^{\mathfrak{B}} \ \Leftrightarrow \ (b_1, \dots, b_{r_i}) \in R_i^{\mathfrak{R}}$ **or** $(b_1 M \times \dots \times b_{r_i} M) \cap R_i^{\mathfrak{A}} \neq \varnothing$.

(in the second case at least one $b_j \notin \mathfrak{A}$)

Structure Rewriting Rules

Rewriting Example



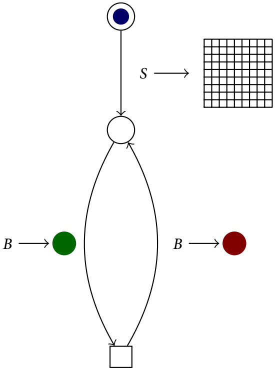
Embedding: σ is **injective** and $R_i^{\mathfrak{A}}(a_1, \ldots, a_{r_i}) \Leftrightarrow R_i^{\mathfrak{B}}(\sigma(a_1), \ldots, \sigma(a_{r_i}))$

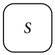
$$\sigma \ : \quad \mathfrak{A} = (A, R_1^{\mathfrak{A}}, R_2^{\mathfrak{A}}, \ldots, R_k^{\mathfrak{A}}) \quad \hookrightarrow \quad (B, R_1^{\mathfrak{B}}, R_2^{\mathfrak{B}}, \ldots, R_k^{\mathfrak{B}}) = \mathfrak{B}$$

Rewriting: $\mathfrak{B} = \mathfrak{A}[\mathfrak{L} \to \mathfrak{R}/\sigma]$ iff $B = (A \setminus \sigma(L)) \dot{\cup} R$ and,
for $M = \{(r, a) \mid a = \sigma(l), r \in P_l^{\mathfrak{R}} \text{ for some } l \in L\} \cup \{(a, a) \mid a \in A\}$,
$(b_1, \ldots, b_{r_i}) \in R_i^{\mathfrak{B}} \ \Leftrightarrow \ (b_1, \ldots, b_{r_i}) \in R_i^{\mathfrak{R}}$ **or** $(b_1 M \times \ldots \times b_{r_i} M) \cap R_i^{\mathfrak{A}} \neq \emptyset$.
(in the second case at least one $b_j \notin \mathfrak{A}$)

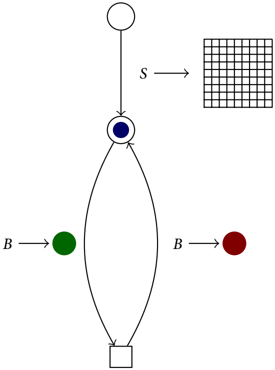
Motivation: many questions **naturally defined** as **structure rewriting games**:
constraint satisfaction, model checking, graph algorithms, games people play

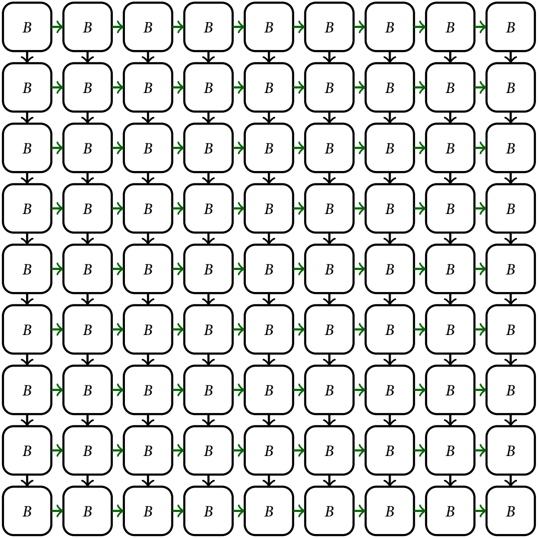
Example System: Gomoku (Connect–5)



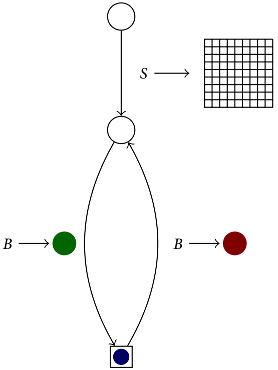


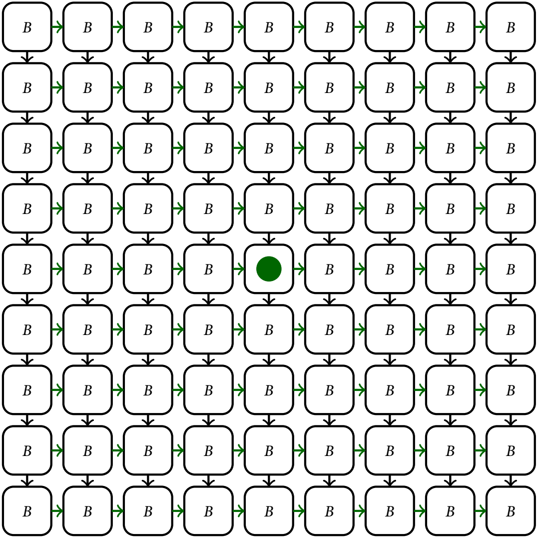
Example System: Gomoku (Connect–5)



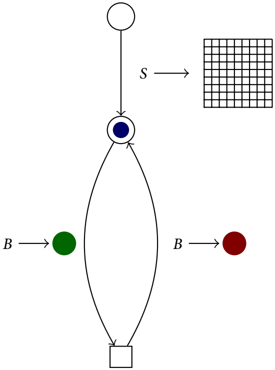


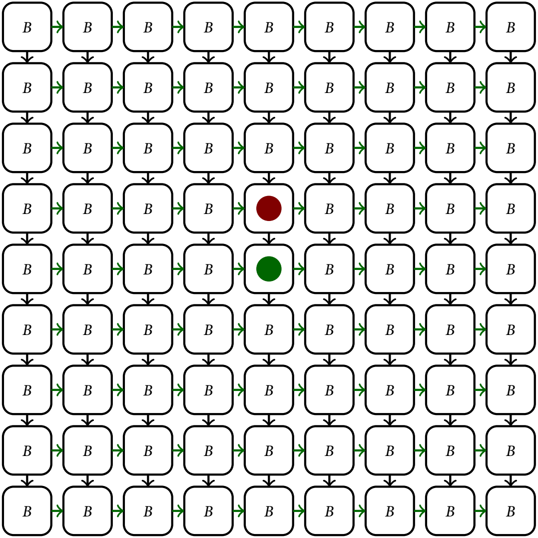
Example System: Gomoku (Connect–5)



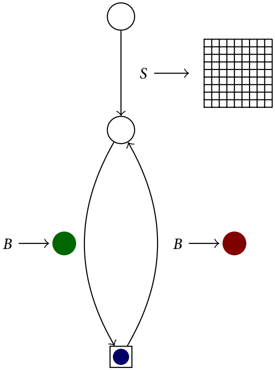


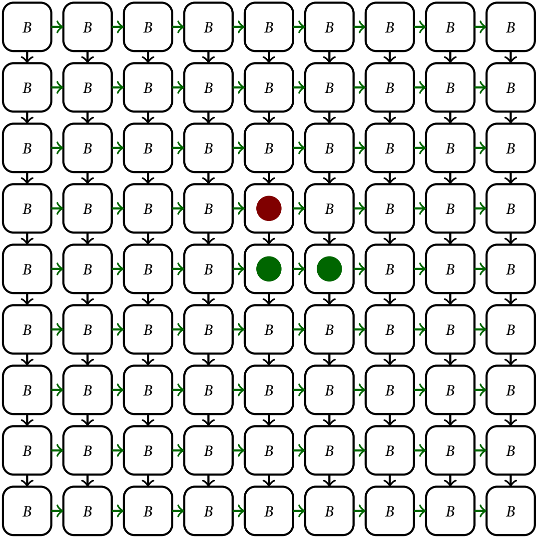
Example System: Gomoku (Connect–5)



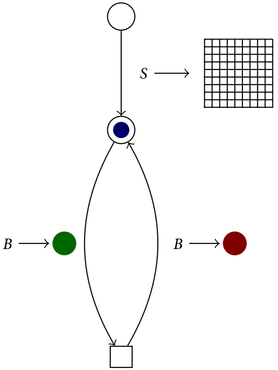


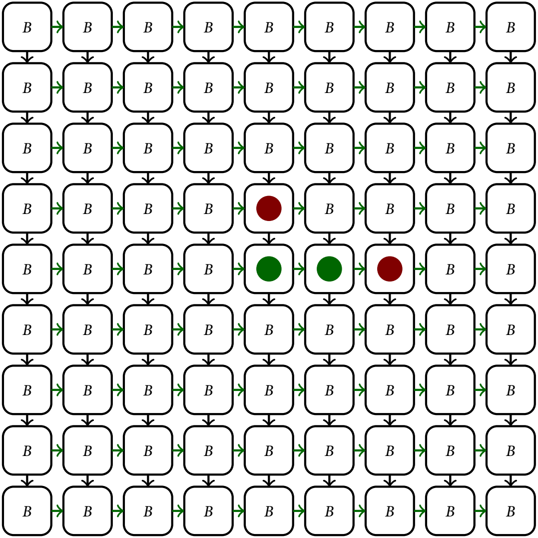
Example System: Gomoku (Connect–5)



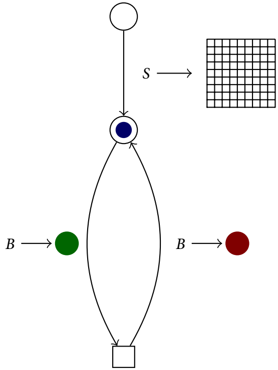


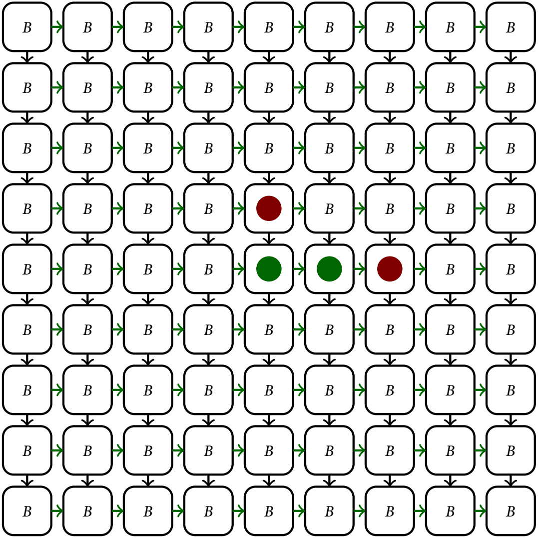
Example System: Gomoku (Connect–5)





Example System: Gomoku (Connect–5)





$$\exists x_1 \dots x_5 \Big(\bigwedge_{1\le i\le 5} G(x_i) \wedge \Big(\bigwedge_{1\le i\le 5} R(x_i, x_{i+1}) \vee \bigwedge_{1\le i\le 5} C(x_i, x_{i+1}) \vee$$

$$\bigwedge_{1\le i\le 5} \exists y (R(x_i, y) \wedge C(y, x_{i+1})) \vee \bigwedge_{1\le i\le 5} \exists y (R(x_i, y) \wedge C(x_{i+1}, y))\Big)\Big)$$

Continuous Dynamics

$\mathbb{R}$-structures: $\mathfrak{A} = (A, R_1, \ldots, R_k, f_1, \ldots, f_l)$ with $f_i : A \to \mathbb{R}$

Additional Parameters to a Rule:

- **dynamics**: system of **ordinary differential equations**
- **updates**: equations assigning **values on the right-hand side**
- **constraints**: precondition, invariant, postcondition

Continuous Dynamics

$\mathbb{R}$**-structures**: $\mathfrak{A} = (A, R_1, \ldots, R_k, f_1, \ldots, f_l)$ with $f_i : A \to \mathbb{R}$

Additional Parameters to a Rule:

- **dynamics**: system of **ordinary differential equations**
- **updates**: equations assigning **values on the right-hand side**
- **constraints**: precondition, invariant, postcondition

Logic

- **Monadic Second-Order Logic** (MSO):
$$\forall X(x \in X \wedge (\forall z, v(z \in X \wedge R(z,v) \to v \in X)) \to y \in X)$$
- **Real-valued terms with counting**: $2 \cdot \chi\big(\exists y(P(y) \wedge R(x,y))\big) + f(x)$
- **Real quantification**: $\exists a \in \mathbb{R}(a^2 \cdot f(x) + a - 1 = 0) \wedge (f(x) > 2)$

Continuous Dynamics

$\mathbb{R}$**-structures**: $\mathfrak{A} = (A, R_1, \ldots, R_k, f_1, \ldots, f_l)$ with $f_i : A \to \mathbb{R}$

Additional Parameters to a Rule:

- **dynamics**: system of **ordinary differential equations**
- **updates**: equations assigning **values on the right-hand side**
- **constraints**: precondition, invariant, postcondition

Logic

- **Monadic Second-Order Logic** (MSO):
$$\forall X(x \in X \wedge (\forall z, v(z \in X \wedge R(z,v) \to v \in X)) \to y \in X)$$
- **Real-valued terms with counting**: $2 \cdot \chi\big(\exists y(P(y) \wedge R(x,y))\big) + f(x)$
- **Real quantification**: $\exists a \in \mathbb{R}(a^2 \cdot f(x) + a - 1 = 0) \wedge (f(x) > 2)$

Semantics of Application

(1) All dynamics applies **concurrently**

(2) Rules with **minimal time** fire

(3) Discrete rewriting **after** continuous evolution

Running the System

How do the players play? Monte Carlo with UCT and Learned Playout

Running the System

How do the players play? Monte Carlo with UCT and Learned Playout

Monte Carlo

- **Both players** play **randomly** a large number of times
- Calculate the **ratio of wins** of each player

Running the System

How do the players play? Monte Carlo with UCT and Learned Playout

Monte Carlo

- **Both players** play **randomly** a large number of times
- Calculate the **ratio of wins** of each player

UCT: Building a Tree during Random Plays

- **Idea:** memorise first random moves, play **minimax** there
- **History:** encouraged by **MoGo** success

Running the System

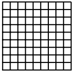
How do the players play? Monte Carlo with UCT and Learned Playout

Monte Carlo

- **Both players** play **randomly** a large number of times
- Calculate the **ratio of wins** of each player

UCT: Building a Tree during Random Plays

- **Idea:** memorise first random moves, play **minimax** there
- **History:** encouraged by **MoGo** success



Running the System

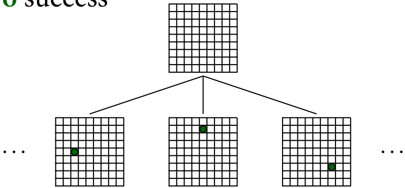
How do the players play? Monte Carlo with UCT and Learned Playout

Monte Carlo

- **Both players** play **randomly** a large number of times
- Calculate the **ratio of wins** of each player

UCT: Building a Tree during Random Plays

- **Idea:** memorise first random moves, play **minimax** there
- **History:** encouraged by **MoGo** success



Running the System

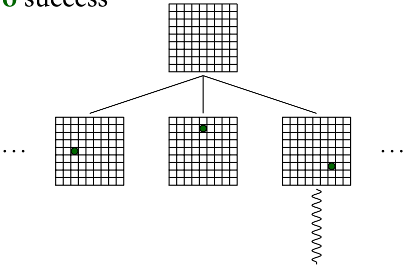
How do the players play? Monte Carlo with UCT and Learned Playout

Monte Carlo

- **Both players** play **randomly** a large number of times
- Calculate the **ratio of wins** of each player

UCT: Building a Tree during Random Plays

- **Idea:** memorise first random moves, play **minimax** there
- **History:** encouraged by **MoGo** success



Running the System

How do the players play? Monte Carlo with UCT and Learned Playout

Monte Carlo

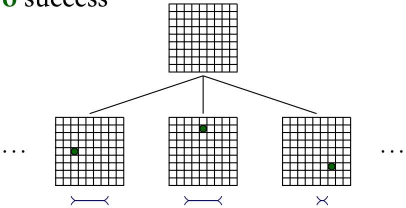
- **Both players** play **randomly** a large number of times
- Calculate the **ratio of wins** of each player

UCT: Building a Tree during Random Plays

- **Idea:** memorise first random moves, play **minimax** there
- **History:** encouraged by **MoGo** success

Pick Max Upper Confidence

$$C \cdot \sqrt{\frac{\ln(n(v)+1)}{n(w)+1}}$$



Running the System

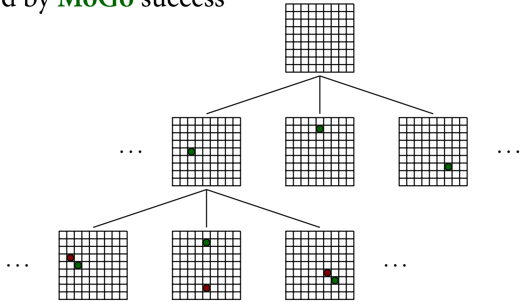
How do the players play? Monte Carlo with UCT and Learned Playout

Monte Carlo

- **Both players** play **randomly** a large number of times
- Calculate the **ratio of wins** of each player

UCT: Building a Tree during Random Plays

- **Idea:** memorise first random moves, play **minimax** there
- **History:** encouraged by **MoGo** success



Running the System

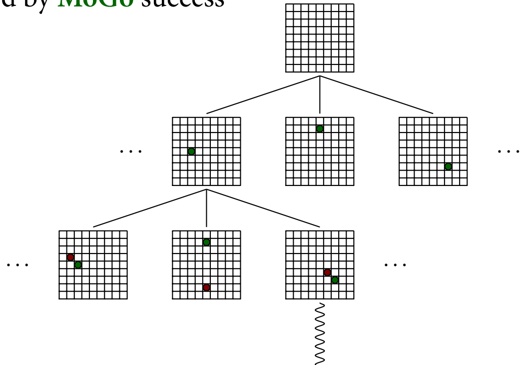
How do the players play? Monte Carlo with UCT and Learned Playout

Monte Carlo

- **Both players** play **randomly** a large number of times
- Calculate the **ratio of wins** of each player

UCT: Building a Tree during Random Plays

- **Idea:** memorise first random moves, play **minimax** there
- **History:** encouraged by **MoGo** success



Running the System

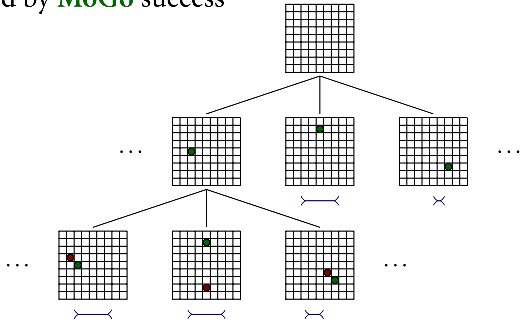
How do the players play? Monte Carlo with UCT and Learned Playout

Monte Carlo

- **Both players** play **randomly** a large number of times
- Calculate the **ratio of wins** of each player

UCT: Building a Tree during Random Plays

- **Idea:** memorise first random moves, play **minimax** there
- **History:** encouraged by **MoGo** success



UCT for Structure Rewriting Games

Improving Playout Policy

- **Hints: formulas** which separate **good and bad** states (**moves**)
- **Example (Gomoku):** connected group of stones is good

UCT for Structure Rewriting Games

Improving Playout Policy

- **Hints:** **formulas** which separate **good and bad** states (**moves**)
- **Example (Gomoku):** connected group of stones is good

Learning Hints

- Make a **play** between two **UCT players without hints**
- Collect high confidence **states appearing during the play**
- Formula should **separate** winning from loosing states
- Search only for **positive existential first-order formulas** (**Apriori**)

UCT for Structure Rewriting Games

Improving Playout Policy

- **Hints: formulas** which separate **good and bad** states (**moves**)
- **Example (Gomoku):** connected group of stones is good

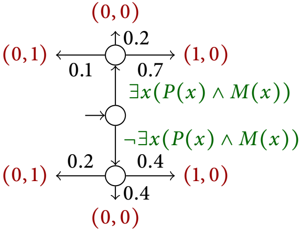
Learning Hints

- Make a **play** between two **UCT players without hints**
- Collect high confidence **states appearing during the play**
- Formula should **separate** winning from loosing states
- Search only for **positive existential first-order formulas** (**Apriori**)

Evaluation Games

Improvements vs. UCT with random

- **Breakthrough**: **beat if possible**
  ca. 70% improvement
- **Gomoku**: **play near your stone**
  ca. 80% improvement



Conclusions

Structure Rewriting Games

- **General** model of games with **structured states**
- Establishing the winner is **decidable** for **certain subclasses**
- **Simulation** can be used to **play** the games
- Possible to **learn formulas** from simulated plays

Conclusions

Structure Rewriting Games

- **General** model of games with **structured states**
- Establishing the winner is **decidable** for **certain subclasses**
- **Simulation** can be used to **play** the games
- Possible to **learn formulas** from simulated plays

Future Work

- **Types of structures** (based on bounded clique-width)
- **Imperfect Information** for players
- **Hierarchical modelling** (objects?)
- **Efficiency improvements**

Conclusions

Structure Rewriting Games

- **General** model of games with **structured states**
- Establishing the winner is **decidable** for **certain subclasses**
- **Simulation** can be used to **play** the games
- Possible to **learn formulas** from simulated plays

Future Work

- **Types of structures** (based on bounded clique-width)
- **Imperfect Information** for players
- **Hierarchical modelling** (objects?)
- **Efficiency improvements**

`toss.sourceforge.net`

Conclusions

Structure Rewriting Games

- **General** model of games with **structured states**
- Establishing the winner is **decidable** for **certain subclasses**
- **Simulation** can be used to **play** the games
- Possible to **learn formulas** from simulated plays

Future Work

- **Types of structures** (based on bounded clique-width)
- **Imperfect Information** for players
- **Hierarchical modelling** (objects?)
- **Efficiency improvements**

`toss.sourceforge.net`

Thank You