General Game Playing (GGP)
Winter term 2013/2014

7. GIGA 1
## Outline

<table>
<thead>
<tr>
<th>Date</th>
<th>What will we do?</th>
</tr>
</thead>
<tbody>
<tr>
<td>22.10.2013</td>
<td>Introduction, Repetition propositional logic and FOL</td>
</tr>
<tr>
<td>29.10.2013</td>
<td>Repetition FOL / Datalog and Prolog</td>
</tr>
<tr>
<td>05.11.2013</td>
<td>Game Description Language</td>
</tr>
<tr>
<td>12.11.2013</td>
<td>Design of GDL games</td>
</tr>
<tr>
<td>19.11.2013</td>
<td>Search Algorithms 1</td>
</tr>
<tr>
<td>26.11.2013</td>
<td>No lecture</td>
</tr>
<tr>
<td>03.12.2013</td>
<td>Search Algorithms 2</td>
</tr>
<tr>
<td>10.12.2013</td>
<td>Real GGP 1</td>
</tr>
<tr>
<td>17.12.2013</td>
<td><strong>Midterm competition</strong></td>
</tr>
<tr>
<td>07.01.2014</td>
<td>GIGA 1 – State of the art 2009</td>
</tr>
<tr>
<td>14.01.2014</td>
<td>GIGA 2 – State of the art 2011</td>
</tr>
<tr>
<td>21.01.2014</td>
<td>GIGA 3 – State of the art 2013</td>
</tr>
<tr>
<td>28.01.2014</td>
<td>Summary and Outlook</td>
</tr>
<tr>
<td>04.02.2014</td>
<td><strong>Final Competition</strong></td>
</tr>
<tr>
<td>11.02.2014</td>
<td><strong>Exam</strong></td>
</tr>
</tbody>
</table>
Repetition: Monto-Carlo Tree Search

- **Four phases:**
  - In the *selection step* the tree is traversed from the root node until it selects a leaf node \( L \) that is not added to the tree yet.
  - Subsequently, the *expansion strategy* is called to add the leaf node \( L \) to the tree.
  - A *simulation strategy* plays moves in a self-play mode until the end of the game is reached. The result \( R \) of such a “simulated” game is +1 in case of a win for Black, 0 in case of a draw, and −1 in case of a win for White.
  - A *backpropagation strategy* propagates the results \( R \) through the tree, i.e., in each node traversed the average result of the simulations is computed.

- **Main advantage:** no need for a game-specific evaluation function!
A single simulation
Today

• We will look at a few papers presented at the 1\textsuperscript{st} workshop on General Game Playing, hosted at IJCAI 2009
  – “The aim of this workshop is bring together researchers from the above sub-fields of AI to discuss how best to address the challenges of and further advance the state-of-the-art of general game-playing systems and generic artificial intelligence.“
  – This is basically the state-of-the-art in year 2009
<table>
<thead>
<tr>
<th>Time</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>09:00 – 09:10</td>
<td>GIGA Inauguration Ceremony</td>
</tr>
</tbody>
</table>
| 09:10 – 10:00    | Session 1: Learning<br>  • Feature Learning Using State Differences  
|                  |  • Chess Revision: Acquiring the Rules of Chess Variants through Theory Revision from Examples  
|                  |  • Stephen Muggleton, Aline Paes, Vitor Santos Costa, Gerzon Zaverucha |
| 10:00 – 10:30    | Coffee Break                                                         |
| 10:30 – 12:10    | Session 2: Simulation<br>  • Meta Monte-Carlo Tree Search for Automatic Opening Book Generation  
|                  |  • Hilmar Finnsson, Yngvi Björnsson                                  |
|                  |  • Information Set Sampling for General Imperfect Information Positional Games  
|                  |  • Mohammad Shafiei, Nathan Sturtevant, Jonathan Schaeffer            |
| 12:10 – 1:20     | Lunch Break                                                          |
| 01:20 – 03:00    | Session 3: Analysis<br>  • Factoring General Games using Propositional Automata  
|                  |  • Martin Günther, Stephan Schiffel, Michael Thielscher               |
|                  |  • Instantiating General Games<br>  • Peter Kissmann, Stefan Edelkamp  
|                  |  • Symmetry Detection in General Game Playing<br>  • Stephan Schiffel   |
| 03:00 – 03:30    | Coffee Break                                                         |
| 03:30 – 04:20    | Session 4: GDL<br>  • From GDL to a Market Specification Language for General Trading Agents  
|                  |  • Michael Thielscher, Dongmo Zhang                                  |
|                  |  • Faster State Manipulation in General Games using Generated Code    
<p>|                  |  • Kevin Waugh                                                        |
| 04:30 – 05:30    | Panel Discussion: Future GGP Competitions&lt;br&gt;  • Michael Genesereth    |</p>
<table>
<thead>
<tr>
<th>Time</th>
<th>Session</th>
</tr>
</thead>
<tbody>
<tr>
<td>09:00 – 09:10</td>
<td>GIGA Inauguration Ceremony</td>
</tr>
<tr>
<td></td>
<td><strong>Session 1: Learning</strong></td>
</tr>
<tr>
<td></td>
<td>• Feature Learning Using State Differences</td>
</tr>
<tr>
<td></td>
<td><em>Mesut Kirci, Jonathan Schaeffer, Nathan Sturtevant</em></td>
</tr>
<tr>
<td></td>
<td>• Chess Revision: Acquiring the Rules of Chess Variants through Theory Revision from Examples</td>
</tr>
<tr>
<td></td>
<td><em>Stephen Muggleton, Aline Paes, Vitor Santos Costa, Gerzon Zaverucha</em></td>
</tr>
<tr>
<td>09:10 – 10:00</td>
<td>Coffee Break</td>
</tr>
<tr>
<td></td>
<td><strong>Session 2: Simulation</strong></td>
</tr>
<tr>
<td></td>
<td>• Meta Monte-Carlo Tree Search for Automatic Opening Book Generation</td>
</tr>
<tr>
<td></td>
<td><em>Guillaume Chaslot, Jean-Baptiste Hoock, Julien Perez, Arpad Rimmel, Olivier Teytaud, Mark Winands</em></td>
</tr>
<tr>
<td></td>
<td>• Simulation Control in General Game Playing Agents</td>
</tr>
<tr>
<td></td>
<td><em>Hilmar Finnsson, Yngvi Björnsson</em></td>
</tr>
<tr>
<td></td>
<td>• Information Set Sampling for General Imperfect Information Positional Games</td>
</tr>
<tr>
<td></td>
<td><em>Mark Richards, Eyal Amir</em></td>
</tr>
<tr>
<td></td>
<td>• Comparing UCT versus CFR in Simultaneous Games</td>
</tr>
<tr>
<td></td>
<td><em>Mohammad Shafiei, Nathan Sturtevant, Jonathan Schaeffer</em></td>
</tr>
<tr>
<td>12:10 – 01:20</td>
<td>Lunch Break</td>
</tr>
<tr>
<td></td>
<td><strong>Session 3: Analysis</strong></td>
</tr>
<tr>
<td></td>
<td>• Factoring General Games using Propositional Automata</td>
</tr>
<tr>
<td></td>
<td><em>Evan Cox, Eric Schaffra, Ryan Madsen, Michael Genesereth</em></td>
</tr>
<tr>
<td></td>
<td>• Factoring General Games</td>
</tr>
<tr>
<td></td>
<td><em>Martin Günther, Stephan Schiffel, Michael Thielscher</em></td>
</tr>
<tr>
<td></td>
<td>• Instantiating General Games</td>
</tr>
<tr>
<td></td>
<td><em>Peter Kissmann, Stefan Edelkamp</em></td>
</tr>
<tr>
<td></td>
<td>• Symmetry Detection in General Game Playing</td>
</tr>
<tr>
<td></td>
<td><em>Stephan Schiffel</em></td>
</tr>
<tr>
<td>01:20 – 03:00</td>
<td>Coffee Break</td>
</tr>
<tr>
<td></td>
<td><strong>Session 4: GDL</strong></td>
</tr>
<tr>
<td></td>
<td>• From GDL to a Market Specification Language for General Trading Agents</td>
</tr>
<tr>
<td></td>
<td><em>Michael Thielscher, Dongmo Zhang</em></td>
</tr>
<tr>
<td></td>
<td>• Faster State Manipulation in General Games using Generated Code</td>
</tr>
<tr>
<td></td>
<td><em>Kevin Waugh</em></td>
</tr>
<tr>
<td>03:00 – 03:30</td>
<td></td>
</tr>
<tr>
<td>03:30 – 04:20</td>
<td>Panel Discussion: Future GGP Competitions</td>
</tr>
<tr>
<td></td>
<td><em>Michael Genesereth</em></td>
</tr>
</tbody>
</table>
Topic 1:
Feature Learning using State Differences
Feature Learning Using State Differences

• Existing work:
  – UCT: no need for evaluation function
    • CadiaPlayer
  – Alpha/beta: evaluation function based on
    • Degree of truth using fuzzy logic (FluxPlayer)
    • Automatically extracted features (Cluneplayer)

• Proposes “Game Independent Feature Learning” (GIFL)

• GIFL consists of two parts:
  – Learning the features
  – Using the features in UCT search
Learning the features

• So far, feature were a subset of a state
  – Here it also contains moves!

• Terminal predicates:
  – predicates that are required for a goal state

• State predicates:
  – All predicates in a state

• Offensive feature:
  – Moves/predicates that (help to) add terminal predicates to a state

• Defensive feature
  – Moves/predicates that (help to) avoid enabling offensive features
2-ply game tree at the end of the game

- **Root state**: 1
- **Middle state**: 2
  - Branch a: 1
  - Branch b: 2
- **Leaf state**:
  - Terminal state: Player 2 wins
  - 1

Diagram shows a 2-ply game tree with branches leading to a leaf state where player 2 wins.
Example for Tic-Tac-Toe

How do offensive/defensive features look like for Tic-Tac-Toe?
Example for Tic-Tac-Toe
Example for Tic-Tac-Toe

How do we identify terminal predicates?
Terminal predicate identification

After we remove the first predicate of (a), the state is not terminal. Therefore, the predicate (cell 1 1 x) is a terminal predicate. In (c), the state is still terminal and (cell 2 1 o) is not a terminal predicate. In the end, terminal predicates are (cell 1 1 x), (cell 2 2 x) and (cell 3 3 x).
Offensive feature learning

- Focus on the last 2-ply of the game sequence
- If the player who made the move at the middle state won the game, an offensive feature is learned from the 2-ply game tree. The move which led to a win (and the satisfaction of the leaf conditions) is considered good and is part of an offensive feature.
- The offensive-feature predicates are required predicates in the middle state to satisfy the leaf conditions after the offensive-feature move is made.
Offensive feature learning

In the end, the predicates (cell 1 1 x) and (cell 3 3 x) are found to be offensive-feature predicates along with the offensive-feature move (mark 2 2 x).
Defensive feature learning

• Again, only look at 2-ply tree
• Idea:
  – The defensive feature either makes the offensive-feature move illegal or makes the offensive-feature predicates false in the middle state
  – The algorithm looks if there are possible moves at the root of the 2-ply game tree that can be counted as defensive-feature moves.
  – Finally, if no defensive feature move can be found, the algorithm backtracks 2 plies in the game tree
Defensive feature learning

(a) 

(b) 

(c)
## Results

<table>
<thead>
<tr>
<th>Game Name</th>
<th>Number of Simulations</th>
<th>Number of Games</th>
<th>Learning-UCT</th>
<th>Win Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>(2008 competition) game2-1000-20</td>
<td>193-7</td>
<td></td>
<td>97.0 %</td>
<td></td>
</tr>
<tr>
<td>knightthrough-1000-20</td>
<td>184-16</td>
<td></td>
<td>92.0 %</td>
<td></td>
</tr>
<tr>
<td>(2008 competition) game1-1000-20</td>
<td>170-30</td>
<td></td>
<td>85.0 %</td>
<td></td>
</tr>
<tr>
<td>breakthrough-1000-20</td>
<td>165-35</td>
<td></td>
<td>82.5 %</td>
<td></td>
</tr>
<tr>
<td>checkers-150-20</td>
<td>156-44</td>
<td></td>
<td>78.0 %</td>
<td></td>
</tr>
<tr>
<td>connect4-1000-100</td>
<td>115-85</td>
<td></td>
<td>57.5 %</td>
<td></td>
</tr>
<tr>
<td>chess-25-40</td>
<td>102-84</td>
<td></td>
<td>56.0 %</td>
<td></td>
</tr>
<tr>
<td>(2008 competition) game5-1000-40</td>
<td>111-94</td>
<td></td>
<td>55.0 %</td>
<td></td>
</tr>
<tr>
<td>pentago-1000-100</td>
<td>100-100</td>
<td></td>
<td>50.0 %</td>
<td></td>
</tr>
<tr>
<td>quarto-1000-100</td>
<td>98-102</td>
<td></td>
<td>49.0 %</td>
<td></td>
</tr>
<tr>
<td>(2008 competition) game6-1000-100</td>
<td>96-104</td>
<td></td>
<td>48.0 %</td>
<td></td>
</tr>
<tr>
<td>(2008 competition) game3-1000-100</td>
<td>91-109</td>
<td></td>
<td>45.0 %</td>
<td></td>
</tr>
<tr>
<td>checkersbarrelnokings-1000-100</td>
<td>61-139</td>
<td></td>
<td>30.0 %</td>
<td></td>
</tr>
<tr>
<td>(2008 competition) game4-1000-40</td>
<td>100-100</td>
<td></td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>
## Results

<table>
<thead>
<tr>
<th>Games from Stanford GGP repository and 2008 GGP Competition</th>
<th>n. of simulations (learner/uct)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(2008 competition) game2</td>
<td>46 %</td>
</tr>
<tr>
<td>knightthrough</td>
<td>93 %</td>
</tr>
<tr>
<td>(2008 competition) game1</td>
<td>104 %</td>
</tr>
<tr>
<td>breakthrough</td>
<td>79 %</td>
</tr>
<tr>
<td>checkers</td>
<td>36 %</td>
</tr>
<tr>
<td>connect4</td>
<td>20 %</td>
</tr>
<tr>
<td>chess</td>
<td>74 %</td>
</tr>
<tr>
<td>(2008 competition) game5</td>
<td>32 %</td>
</tr>
<tr>
<td>pentago</td>
<td>156 %</td>
</tr>
<tr>
<td>quarto</td>
<td>34 %</td>
</tr>
<tr>
<td>(2008 competition) game6</td>
<td>58 %</td>
</tr>
<tr>
<td>(2008 competition) game3</td>
<td>99 %</td>
</tr>
<tr>
<td>checkersbarrelnokings</td>
<td>38 %</td>
</tr>
<tr>
<td>(2008 competition) game4</td>
<td>47 %</td>
</tr>
</tbody>
</table>
Major drawback?

- Only works if the winning party makes the last move
Topic 2:

**Meta Monte-Carlo Tree Search for Automatic Opening Book Generation**
Problem

• Generating opening book for Monte-Carlo Tree Search algorithms

• Definition: An "opening book“, in chess, is made by taking thousands (or hundreds of thousands) of GM games and creating a database of opening moves. Then, when your opponent plays a move you have in your database, you have a move to play in response, instantly. These moves are generally considered to be "good" because they were played by grandmaster players, and if they are played often enough, with good results, they are considered to be "opening theory" and are played whenever possible. (http://www.cis.uab.edu/hyatt/learning.html)
Solution: Meta-MCTS

• Weak simulation strategy at the lower part of the search is replaced by an entire MCTS program
• Standard UCT algorithm requires an exploration constant to be tuned.
  – Tuning such a constant for a two-level MCTS would take quite an amount of time.
• Therefore, two alternatives are proposed:
  – Quasi Best-First and
  – Beta Distribution Sampling (not covered here)
A single simulation
Quasi-Best-First Search

• Main observation:
  – The exploration constant, when optimized, is often set close to zero. A small exploration is given to every move using a specific strategy. The consequence of using such a small exploration is that, after a few games, a move is further analyzed as long as it is the move with the highest winning rate. Hence, most MCTS programs can be qualified as being greedy.

• Solution:
  – Program usually selects the child with the highest winning rate. However, if a move’s winning rate drops below a certain threshold $K$, QBF will ask real UCT player to chose a move.
Quasi-Best-First Search

Algorithm 1 The “Quasi Best-First” (QBF) algorithm. \( \lambda \) is the number of machines available. \( K \) is a constant. \( g \) is a game, defined as a sequence of game states. The function “MoGoChoice” asks MoGo to choose a move.

\[
\text{QBF}(K, \lambda)
\]

\[
\text{while True do}
\]
\[
\text{for } l = 1..\lambda, \text{ do}
\]
\[
\text{s = initial state: } g = \{s\}.
\]
\[
\text{while } s \text{ is not a final state do}
\]
\[
\text{bestScore} = K
\]
\[
\text{bestMove} = \text{Null}
\]
\[
\text{for } m \text{ in the set of possible moves in } s \text{ do}
\]
\[
\text{score} = \text{percentage of won games by playing the move } m \text{ in } s
\]
\[
\text{if } \text{score} > \text{bestScore} \text{ then}
\]
\[
\text{bestScore} = \text{score}
\]
\[
\text{bestMove} = m
\]
\[
\text{end if}
\]
\[
\text{end for}
\]
\[
\text{if } \text{bestMove} = \text{Null} \text{ then}
\]
\[
\text{bestMove} = \text{MoGoChoice}(s) \text{ // lower level MCTS}
\]
\[
\text{end if}
\]
\[
\text{s = playMove(s, bestMove)}
\]
\[
\text{g = concat(g, s)}
\]
\[
\text{end while}
\]
\[
\text{Add } g \text{ and the result of the game in the book.}
\]
\[
\text{end for}
\]
\[
\text{end while}
\]
Topic 3:
Simulation Control in General Game Playing Agents
Simulation Control in General Game Playing Agents

- Describes four search-control mechanisms for guiding simulation runs
  - Move-average sampling technique
  - Tree-Only MST
  - Predicate-Average Sampling Technique
  - Rapid Action Value Estimation
Move-Average Sampling Technique

- Learns search-control information during the back-propagations step
- More specifically, when a return value of a simulation is backed up from T to S, then for each action a on the path a global (over all simulations) average for the action a, $Q_h(a)$, is incrementally calculated and kept in a lookup table.

$$P(a) = \frac{e^{Q_h(a)/\tau}}{\sum_{b=1}^{n} e^{Q_h(b)/\tau}}$$
Tree-Only MAST

- Variation of MAST
- Instead of updating the Qh(a) for an entire simulation episode, it does so only for the part within the game tree (from state N back to S).
- More selective
- Prefers quality of data over sample quality
Predicate-Average Sampling Technique

• Predicates in encountered states are used for generalization
• Compared to MAST, now $Q_p(p,a)$ is maintained
• During the back-propagation, in a state $s$ where action $a$ was taken, $Q_p(p, a)$ is updated for all $p$ in $P(s)$ where $P(s)$ is the set of predicates that are true in state $s$.
• PAST can distinguish moves that are only good in a given context
• Apply threshold to ignore values with high variance
Rapid Action Value Estimation

- When backing up the value of a simulation, RAVE updates in the tree not only the value for the action taken, $Q(s, a)$, but also sibling action values, $Q_{RAVE}(s, a_0)$, if and only if action $a_0$ occurs further down the path being backed up ($s$ to $T$).
- Should only be used when sampled data is still unreliable
- $Q(s,a)$ should be trusted more in mature stage of the game
Empirical evaluation

Table 1: Tournament against the MCTS agent.

<table>
<thead>
<tr>
<th>Game</th>
<th>MAST win %</th>
<th>TO-MAST win %</th>
<th>PAST win %</th>
<th>RAVE win %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Checkers</td>
<td>54.83 (± 5.42)</td>
<td>80.67 (± 4.23)</td>
<td>61.33 (± 5.20)</td>
<td>61.50 (± 5.27)</td>
</tr>
<tr>
<td>Othello</td>
<td>58.67 (± 5.48)</td>
<td>54.00 (± 5.55)</td>
<td>61.33 (± 5.42)</td>
<td>57.17 (± 5.50)</td>
</tr>
<tr>
<td>Breakthrough</td>
<td>88.67 (± 3.59)</td>
<td>86.67 (± 3.85)</td>
<td>89.67 (± 3.45)</td>
<td>60.67 (± 5.54)</td>
</tr>
</tbody>
</table>

Table 2: Tournament against the MAST agent.

<table>
<thead>
<tr>
<th>Game</th>
<th>TO-MAST win %</th>
<th>PAST win %</th>
<th>RAVE win %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Checkers</td>
<td>74.83 (± 4.64)</td>
<td>57.67 (± 5.27)</td>
<td>61.33 (± 5.16)</td>
</tr>
<tr>
<td>Othello</td>
<td>37.00 (± 5.35)</td>
<td>49.67 (± 5.51)</td>
<td>46.50 (± 5.51)</td>
</tr>
<tr>
<td>Breakthrough</td>
<td>49.33 (± 5.67)</td>
<td>42.33 (± 5.60)</td>
<td>13.00 (± 3.81)</td>
</tr>
</tbody>
</table>

Table 3: Tournament between MCTS and RAVE/MAST.

<table>
<thead>
<tr>
<th>Game</th>
<th>RAVE/MAST win %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Checkers</td>
<td>62.83 (± 5.25)</td>
</tr>
<tr>
<td>Othello</td>
<td>66.83 (± 5.29)</td>
</tr>
<tr>
<td>Breakthrough</td>
<td>89.00 (± 3.55)</td>
</tr>
</tbody>
</table>

Table 4: Tournament between MAST and RAVE/MAST.

<table>
<thead>
<tr>
<th>Game</th>
<th>RAVE/MAST win %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Checkers</td>
<td>74.50 (± 4.80)</td>
</tr>
<tr>
<td>Othello</td>
<td>60.33 (± 5.43)</td>
</tr>
<tr>
<td>Breakthrough</td>
<td>46.33 (± 5.65)</td>
</tr>
</tbody>
</table>
Topic 4: Factoring General Games using Propositional Automata
Factoring general games

- Problem:
  - Games often/may consist of sub-games
  - For instance TicTacToeParallel
  - Size of the game tree grows exponentially with each added game
    - Single TicTacToe: \( n=255,168 \) states
    - Double TicTacToe: \( n^2=65 \) billion states
  - Minimax/MonteCarlo have a difficult time to perform an effective evaluation with limited time play clock
- Identifying independences of sub games leads to only \( 2n \) states
Hodgepodge

Hodgepodge = Chess + Othello

Branching factor: \( a \)

Branching factor: \( b \)

Analysis of joint game:

Branching factor as given to players: \( a * b \)

Fringe of tree at depth \( n \) as given: \( (a * b)^n \)

Fringe of tree at depth \( n \) factored: \( a^n + b^n \)
Factoring general games

- The class of simultaneous independent games are defined as those games in which each agent takes an action in each independent game on each time step.

- The goal for an agent is to win every independent game.
State-Machine

Usual state-machine view of a game
Propositional nets

- A propositional net is a graph in which propositions and actions are nodes rather than states and where these nodes are interleaved with nodes representing logical connectives and transitions, as suggested by the example shown below.
Propositional Net Components

Propositions

\[ p \quad q \quad r \]

Connectives

Transitions
Propositional Net

Input Proposition

Proposition

Base Proposition

View
Markings

A *marking* for a propositional net is a function from the propositions $P$ to boolean values.

\[ m: P \rightarrow \{true, false\} \]

A marking $m$ is *partial* if and only if $m$ is a partial function. Otherwise, it is *total*.

Think of a marking as a state of a game.
Acceptability

A marking is *acceptable* if and only if it obeys the logical properties of all connectives.

Negation with input \( x \) and output \( y \):

\[
m(x) = \text{false} \iff m(y) = \text{true}
\]

Conjunction with inputs \( x \) and \( y \) and output \( z \):

\[
m(x) = \text{true} \land m(y) = \text{true} \iff m(z) = \text{true}
\]

Disjunction with inputs \( x \) and \( y \) and output \( z \):

\[
m(x) = \text{true} \lor m(y) = \text{true} \iff m(z) = \text{true}
\]
Definitions

A transition is *enabled* by a marking $m$ if and only if all of its inputs are marked *true*.

The *transitional marking* for $m$ is the partial marking that assigns *true* to the base propositions that are outputs of transitions enabled by $m$ and *false* to the outputs of all other base propositions.

An *input marking* for a propositional net is a marking for the input propositions.
Consider the propnet shown above. Assume a base marking that assigns $s$ the value 0, and consider an input marking that assigns $a$ the value 0 and $b$ the value 1.

- What is the marking of $p$ on this step?
- What is the marking of $q$ on this step?
- What is the marking of $r$ on this step?
- What is the marking of $s$ on this step?
- What is the marking of $s$ on the next step?
Buttons and Lights

Pressing button $a$ toggles $p$.
Pressing button $b$ interchanges $p$ and $q$.

What does a propositional net for B&L look like?
Propositional Net for Buttons and Lights
Tic-Tac-Toe
Partial Propositional Net for Tic-Tac-Toe
Propositional Net Fragment
How to factor propositional nets?
Factorable Example
Solution

• Identify disconnected components (factors) in the propositional network of the complete game
  – be careful about goal/terminal criteria
  – => Components can be played separately

• Identify equivalent graphs (or subgraphs) via homomorphism
  – Only need to “learn” how to play for one equivalent factor
Example
Example
Topic 5: Factoring General Games
Factoring General Games

• Same problem as before, but slightly different solution

**Definition 1.** Let $F$ be a GDL formula. The call graph $G = (V, E)$ for $F$ is the smallest graph with the following properties:

1. if $t$ is an atomic formula occurring in $F$, then $t \in V$
2. if there is a game rule $r$ with head $h$, and $t \in V$ and $h$ are unifiable, then
   - all atomic formulae $b_1, \ldots, b_n$ occurring in the body of $r$ are elements of $E$, and
   - $\{\langle t, b_1 \rangle, \ldots, \langle t, b_n \rangle\} \subseteq E$
Incredible

• The game Incredible consists of two subgames:
  – The well-known “blocks world” and “maze” and
  – another very simple single-player game in which a robot has to find a piece of gold.
Dependency graph for “Incredible”
Algorithm for subgame-detection

• Simple: find connected components
• Result:

\[ \sigma_{maze} = (\{cell, gold\}, \{move, grab, drop\}) \]
\[ \sigma_{blocks} = (\{on, clear, table\}, \{stack, unstack\}) \]
\[ \sigma_{step} = (\{step\}, \emptyset) \]
How to exploit plans for sub-games?

• Naïve approach:
  • Search each subgame as if it were a separate game (this is possible because the features and actions of different subgames cannot influence each other), yielding a local plan (a sequence of actions that leads to a desired goal state).
  • Then, execute the resulting local plans one after another.
• Two problems:
  – Goal predicate is usually only defined on complete states
  – Concatenating local plans may fail, since the game could terminate before all plans have been executed

• Solution:
  – Path interleaving (for details see the paper)
### Evaluation

<table>
<thead>
<tr>
<th></th>
<th># computed states</th>
<th>computation time [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decomposition Search</td>
<td>3,212</td>
<td>45.331</td>
</tr>
<tr>
<td>Standard Fluxplayer</td>
<td>41,191,436</td>
<td>8510.648</td>
</tr>
</tbody>
</table>
Topic 6:
Instantiating General Games
Instantiating General Games

• Most existing players use Prolog to reason about the games states/actions/axioms
• This limits the search speed (number of expanded nodes/s)
• Idea:
  – Eliminate variables, so-called instantiation
Example game

(role robot)
(init (cell a)) (init (gold c)) (init (step 1))
(adjacent a b) (adjacent b c) (adjacent c d) (adjacent d a)
(<= timeout (true (step 10)))
(<= (legal robot grab) (true (cell ?x)) (true (gold ?x)))
(<= (legal robot move))
(<= (next (cell ?y))
(does robot move) (true (cell ?x)) (adjacent ?x ?y))
(<= terminal timeout)
(<= terminal (true (gold a)))
(<= (goal robot 100) (true (gold a)))
(<= (goal robot 0) (true (gold ?x)) (distinct ?x a))
Algorithm

• 1\textsuperscript{st} step: normalization
  – Basically: Eliminate disjunction from rules
Step 2: Supersets (Variant 1)

• Fix-point computation in Prolog
  – Start with initial state in KB
  – Loop until KB does not change
    • Compute legal actions
    • Compute all next states of all legal actions
    • Add all next states to KB

• Result is a superset of all reachable axioms
Step 2: Supersets (Variant 2)

- Using dependency graphs

```
(init (step 0))
(<= (next (step ?s2))
  (true (step ?s1))
  (succ ?s1 ?s2))
(succ 0 1) (succ 1 2) ... (succ 9 10)
```
Algorithm

- Start with predicates $p_1, \ldots, p_n$ which depend only on constants
  - Compute their instantiations
- Instantiate all predicates which depend directly on $p_1, \ldots, p_n$ only
- And so on…

Problem?
Algorithm

• Start with predicates $p_1, \ldots, p_n$ which depend only on constants
  – Compute their instantiations
• Instantiate all predicates which depend directly on $p_1, \ldots, p_n$ only
• And so on…

Problem?
The superset can be really large, often much larger than for Variant 1 (Prolog)
### 3rd step: Rule Instantiation

<table>
<thead>
<tr>
<th>?y ?x</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>(next(cell ?y))</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
</tr>
<tr>
<td>(true(cell ?x))</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
</tr>
<tr>
<td>(adjacent ?x ?y)</td>
<td>b</td>
<td>a</td>
<td>c</td>
<td>b</td>
</tr>
</tbody>
</table>

- \( <= (\text{next(cell b)}) \)
- \( \text{(does robot move)} \)
- \( \text{(true(cell a))} \)
- \( \text{(adjacent a b)} \)
- \( \text{(next(cell d))} \)
- \( \text{(does robot move)} \)
- \( \text{(true(cell d))} \)
- \( \text{(adjacent c d)} \)

Fig. 1. Instantiated \text{next} formula from the Maze example.
Evaluation

Number of expanded states using pure Monte-Carlo search with Prolog and with instantiated input. The timeout was set to 10 seconds. All game descriptions are taken from Dresden’s GGP server.

<table>
<thead>
<tr>
<th>Game</th>
<th>Exp&lt;sub&gt;Prolog&lt;/sub&gt;</th>
<th>Exp&lt;sub&gt;Inst.&lt;/sub&gt;</th>
<th>Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>asteroidsserial</td>
<td>59,364</td>
<td>219,575</td>
<td>3.70</td>
</tr>
<tr>
<td>beatmania</td>
<td>28,680</td>
<td>3,129,300</td>
<td>109.11</td>
</tr>
<tr>
<td>chomp</td>
<td>22,020</td>
<td>1,526,445</td>
<td>69.32</td>
</tr>
<tr>
<td>connectfour</td>
<td>44,449</td>
<td>2,020,006</td>
<td>45.45</td>
</tr>
<tr>
<td>hanoi</td>
<td>84,785</td>
<td>7,927,847</td>
<td>93.51</td>
</tr>
<tr>
<td>lightsout</td>
<td>28,800</td>
<td>7,230,080</td>
<td>251.04</td>
</tr>
<tr>
<td>pancakes6</td>
<td>154,219</td>
<td>2,092,308</td>
<td>13.57</td>
</tr>
<tr>
<td>peg_bugfixed</td>
<td>19,951</td>
<td>1,966,075</td>
<td>98.55</td>
</tr>
<tr>
<td>sheep_and_wolf</td>
<td>20,448</td>
<td>882,738</td>
<td>43.17</td>
</tr>
<tr>
<td>tictactoe</td>
<td>65,864</td>
<td>5,654,553</td>
<td>85.85</td>
</tr>
</tbody>
</table>
The GGP competition also took place at GIGA!

<table>
<thead>
<tr>
<th>Place</th>
<th>Player</th>
<th>Team</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Ary</td>
<td>Jean Méhat (Université de Paris, France)</td>
</tr>
<tr>
<td>2</td>
<td>Fluxplayer</td>
<td>Stephan Schiffel, Michael Thielischer (Dresden University of Technology, Germany)</td>
</tr>
<tr>
<td>3</td>
<td>Maligne</td>
<td>Nathan Sturtevant, Neil Burch, Jonathan Schaeffer, Mesut Kirci, Mohammad Shafiei, Kevin Waugh, Richard Valenzano, Mehdi Samadi (University of Alberta, Canada)</td>
</tr>
<tr>
<td>4</td>
<td>Centurio</td>
<td>Maximilian Möller, Marius Schneider, Martin Wegner (University of Potsdam, Germany)</td>
</tr>
<tr>
<td>5</td>
<td>Ethos</td>
<td>Ethan Petrick Dreyfuss (Stanford University, USA)</td>
</tr>
<tr>
<td>6</td>
<td>CadiaPlayer</td>
<td>Hilmar Finnsson, Yngvi Björnsson (Reykjavik University, Iceland)</td>
</tr>
<tr>
<td>7</td>
<td>Gamer</td>
<td>Peter Kissmann, Stefan Edelkamp (TU Dortmund / TZI Bremen, Germany)</td>
</tr>
<tr>
<td>8</td>
<td>TurboTurtle</td>
<td>Sam Schreiber, Steven Bills, Mike Mintz (Stanford University, USA)</td>
</tr>
</tbody>
</table>

Details can be found here:

How does Ary work?

• Translation of GDL into Prolog!
• First: Random moves only => in the end move with best mean reward is being played
• Later changed to MCTS with UCT
• Some work on root-parallelization of MCTS
  – See
Outlook

• During the next two lectures we will look at
  – GIGA 11
  – GIGA 13
Acknowledgements

Published at GIGA’09 (1\textsuperscript{st} workshop on General Game Playing):

- Mesut Kirci et.al.: Feature Learning Using State Differences
- Guillaume Chaselot et.al.: Meta Monte-Carlo Tree Search for Automatic Opening Book Generation
- Finnsson et.al.: Simulation Control in General Game Playing Agents
- Evan Cox et.al.: Factoring General Games using Propositional Automata
- Günther et.al.: Factoring General Games
- Kissmann: Instantiating General Games

Information on Ary:

- Jean Méhat at. al.: Ary, a general game playing program

Material for Propositional nets:

- [http://arrogant.stanford.edu/ggp/chapters/chapter_09.html](http://arrogant.stanford.edu/ggp/chapters/chapter_09.html)
- Slides for lecture 6/7 of “General Game Playing” at Stanford (Michael Genesereth)