General Game Playing (GGP)
Winter term 2013/2014

9. State-of-the-art

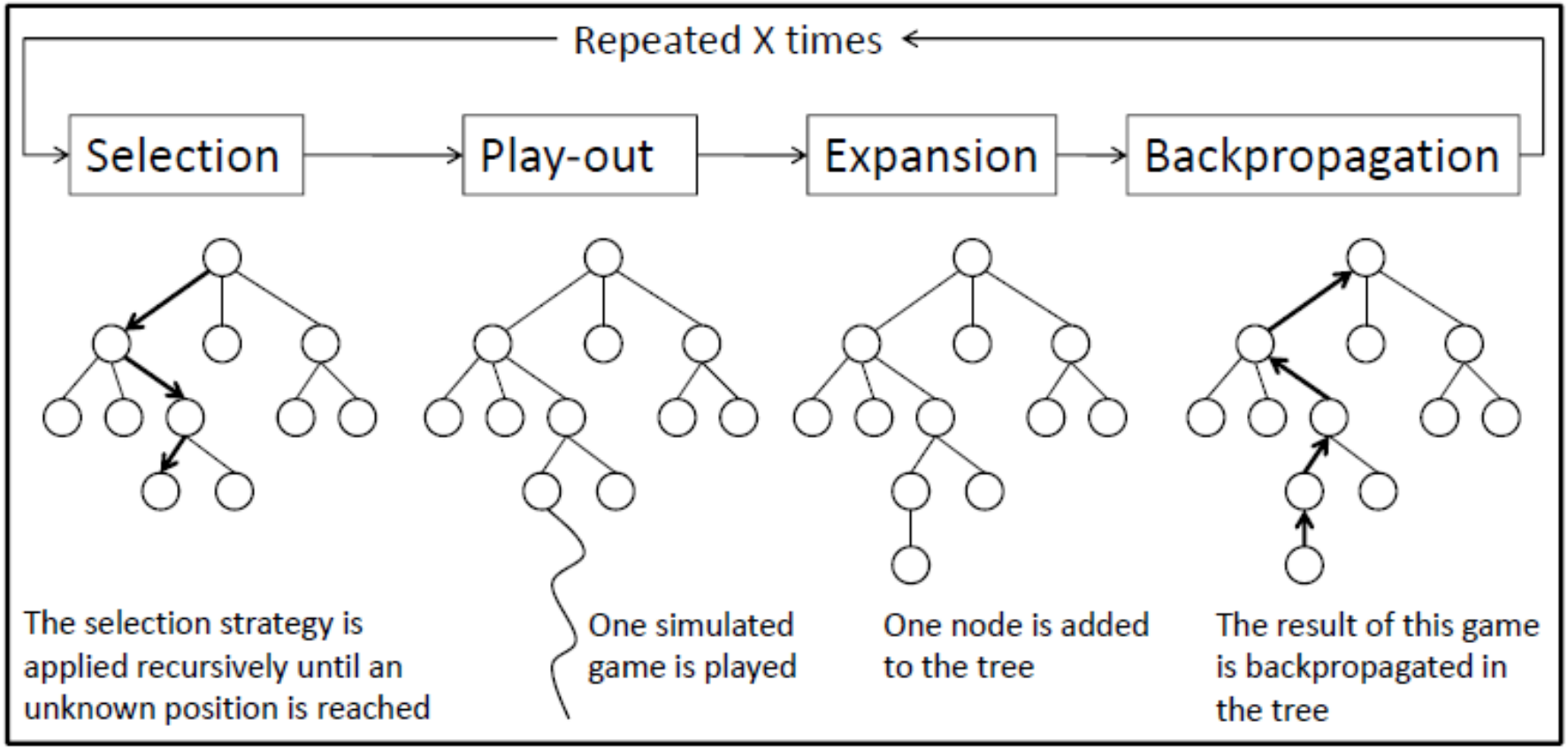
Sebastian Wandelt

WBI, Humboldt-Universität zu Berlin

Outline

Date	What will we do?
22.10.2013	Introduction, Repetition propositional logic and FOL
29.10.2013	Repetition FOL / Datalog and Prolog
05.11.2013	Game Description Language
12.11.2013	Design of GDL games
19.11.2013	Search Algorithms 1
26.11.2013	No lecture
03.12.2013	Search Algorithms 2
10.12.2013	Real GGP 1
17.12.2013	**Midterm competition**
07.01.2014	GIGA 1 – State of the art 2009
14.01.2014	GIGA 2 – State of the art 2011
21.01.2014	MCTS extensions , Summary and Outlook
28.01.2014	Please work on your implementation!
04.02.2014	**Final Competition**
11.02.2014	**Exam**

N-Grams and the Last-Good-Reply Policy


Repeated X times
Selection
Play-out
Expansion
Backpropagation
The selection strategy is applied recursively until an unknown position is reached
One simulated game is played
One node is added to the tree
The result of this game is backpropagated in the tree

Selection step

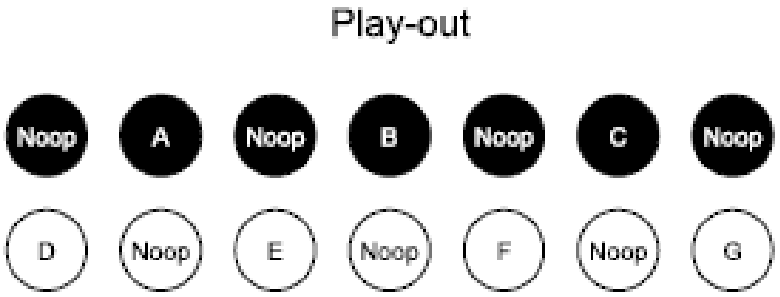
$$a^* = \underset{a \in A(s)}{\operatorname{argmax}} \left\{ Q(s,a) + C\sqrt{\frac{\ln N(s)}{N(s,a)}} \right\}$$

New: N-gram selection technique (NST)

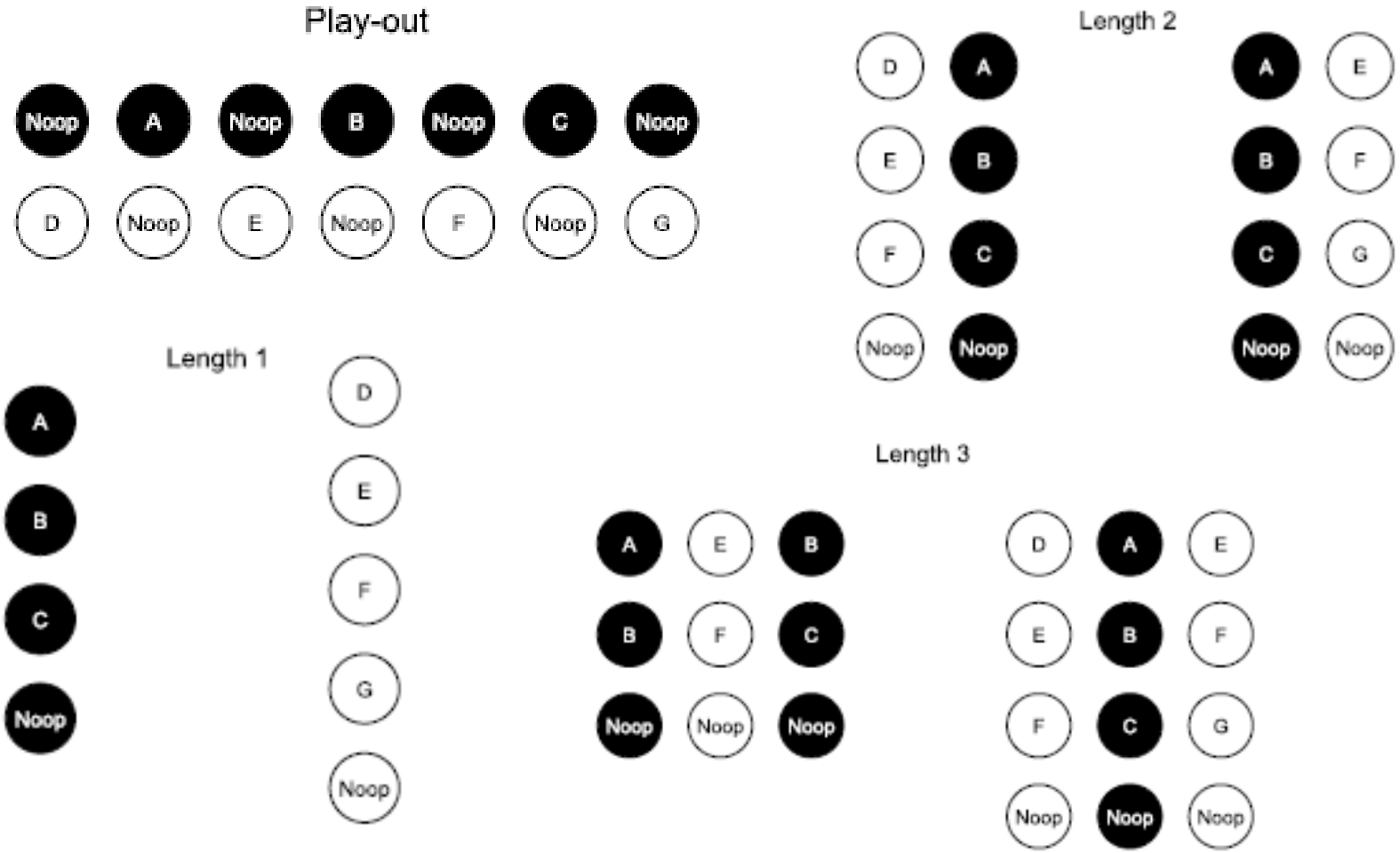
Idea:

- Keep track of move sequences, instead of single moves
- Similar to n-grams in statistical language processing
- Move sequences of length 1,2, and 3

- The N-Grams are formed as follows (for games with n players, current player index is i):
  - A move sequence of length 1 consists of just one move of the current player
  - A sequence of length 2 starts with a move of player with role (i + n - 1) mod n and ends with a move of the current player.
  - A sequence of length 3 starts with a move of player with role (i + n - 2) mod n, followed by a move of player with role (i + n - 1) mod n, and ends with a move of the current player.
- In truly simultaneous games, only one move per step is considered
- Scores for move sequences are updated as usual, i.e. using back propagation



- Which move sequences are extracted?


Play-out
Noop
A
Noop
B
Noop
C
Noop
D
Noop
E
Noop
F
Noop
G
Length 2
D
A
E
B
F
C
Noop
Noop
A
E
B
F
C
G
Noop
Noop
Length 1
A
B
C
Noop
D
E
F
G
Noop
Length 3
A
E
B
B
F
C
Noop
Noop
Noop
D
A
E
E
B
F
F
C
G
Noop
Noop
Noop

Exploitation during playout

- For each legal move, the player determines which sequence of length 1, which sequence of length 2 and which sequence of length 3, would occur when that move is played.
- The player then calculates a score for a move by taking the average of the *R(s) values stored for* these sequences.
- The *R(s) values for the* move sequences of length 2 and length 3 are only taken into account if they are visited at least *k times.*
- *For non-played sequences, the score is set to 100*

New: Last-Good-Reply Policy (LGRP)

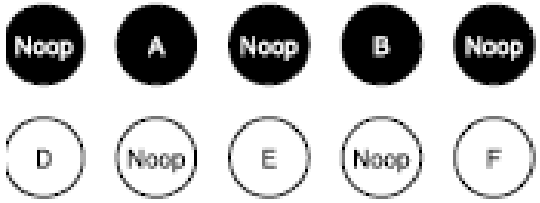
- Successful replies that occurred in the play-outs are kept in memory
- For each player i, two separate tables are created:
  - the best replies to a previous move of the player with role number (*i + n - 1) mod n are stored*
  - the best replies to the previous two moves are stored. This sequence of two moves starts with a move made by the player with role number (*i + n - 2) mod n and ends with a move* played by the player with role number (*i + n - 1) mod n.*

LGRP update

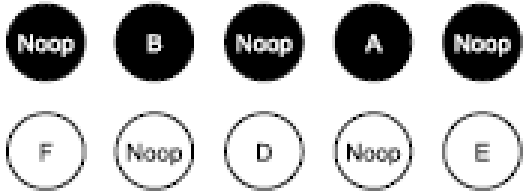
- Tables are updated after each simulation
  - If the reward obtained by the player is at least as high as the highest reward obtained by any of the players then the moves made by the player are stored in the table as being a best reply to their immediate predecessor(s)
  - If the reward of the player is lower than that of any of the opponents then the moves of the player are removed from the table if they were stored as best reply to their immediate predecessor(s).

LGRP example

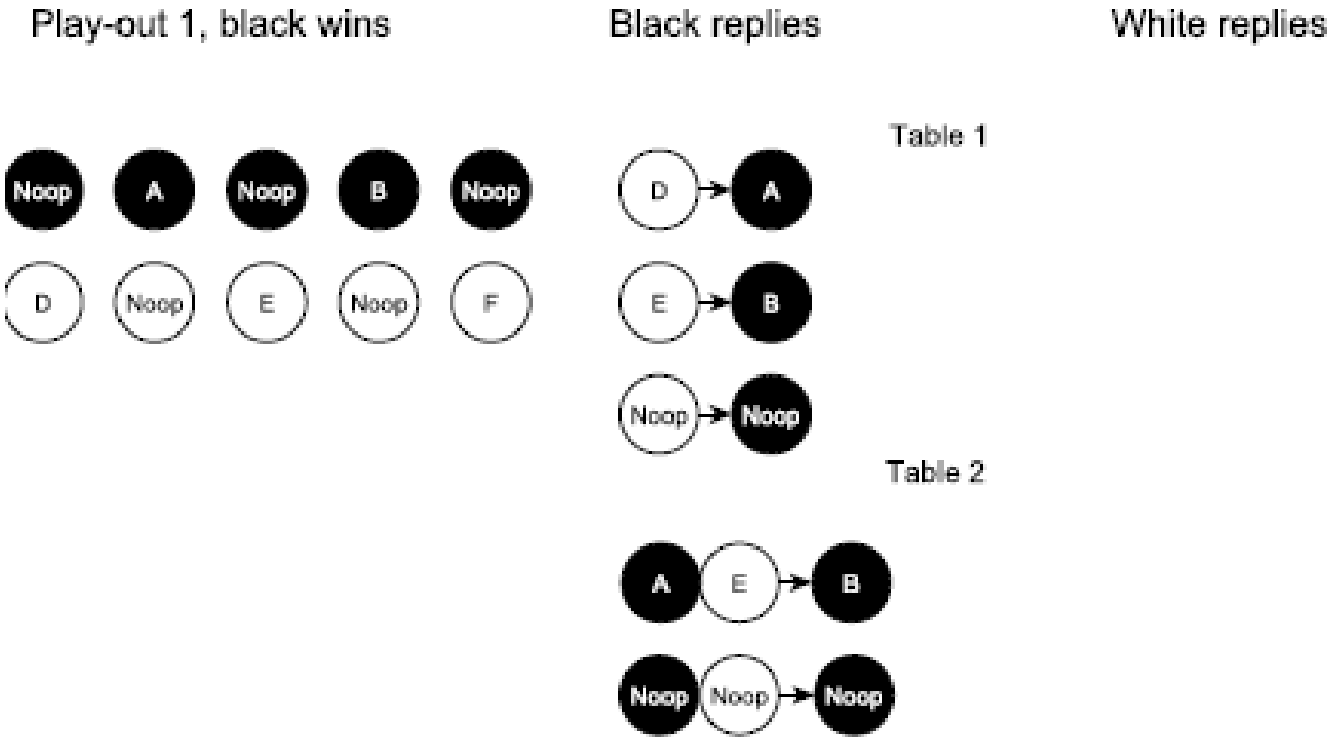
Play-out 1, black wins



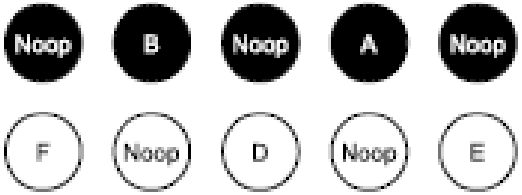
Play-out 2, white wins



LGRP example



Play-out 2, white wins



Black replies

Table 1

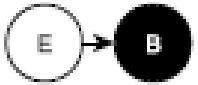
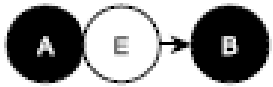
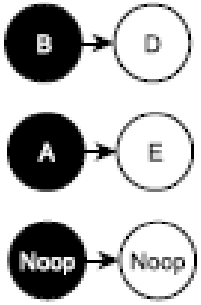
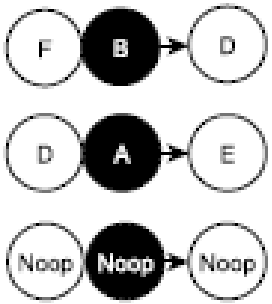


Table 2



White replies





Evaluation of NST and LGRP

- Both optimizations implemented in CadiaPlayer

TABLE X
WIN % AGAINST CP_{UCT}, STARTCLOCK=60S, PLAYCLOCK=30S

Game	CP_{MAST}	CP_{NST}	$CP_{LGR\text{-}MASTG}$	$CP_{LGR\text{-}NST}$
Connect5	66.5 (±5.34)	85.8 (±3.95)	84.0 (±4.15)	**91.3** (±3.18)
Checkers	56.7 (±5.61)	71.0 (±5.11)	**76.9** (±4.69)	69.6 (±5.18)
Breakthrough	86.0 (±3.93)	96.4 (±2.10)	88.0 (±3.68)	**97.3** (±1.82)
Othello	70.1 (±5.15)	**82.8** (±4.27)	70.0 (±5.19)	73.6 (±4.93)
Skirmish	45.0 (±5.63)	**71.5** (±5.11)	56.2 (±5.61)	69.2 (±5.22)
Clobber	52.0 (±5.65)	54.3 (±5.64)	**55.6** (±5.21)	48.7 (±5.64)
Sheep and Wolf	51.3 (±5.66)	65.3 (±5.39)	55.3 (±5.63)	**71.0** (±5.13)
TTCC4	60.6 (±5.05)	**66.0** (±4.89)	44.7 (±5.14)	45.7 (±5.15)
Chinese Checkers	65.1 (±4.85)	**73.0** (±4.22)	59.7 (±4.98)	59.4 (±4.99)
Chinook	79.3 (±4.56)	**90.0** (±3.39)	84.2 (±4.13)	88.7 (±3.59)
Runners	**54.2** (±5.64)	31.3 (±5.25)	37.5 (±5.48)	28.7 (±5.12)
Fighter	49.3 (±5.66)	51.9 (±5.46)	51.5 (±5.66)	**52.8** (±5.48)
Pawn Whopping	**74.3** (±4.91)	74.0 (±4.93)	71.4 (±5.08)	73.2 (±4.93)

Nested Monte-Carlo Search

Idea

- Perform a principal playout with a bias on the selection of each move based on the results of a Monte-Carlo tree search
- Extension of Ary

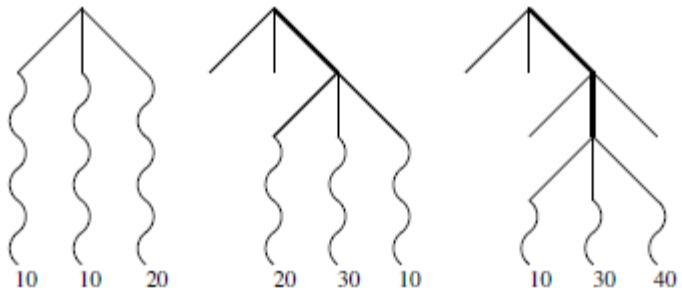


Figure 1. At each step of the principal playout shown here with a bold line, an NMC of level n performs a NMC of level $n-1$ (shown with wavy lines) for each available move and selects the best one. At level 0, a simple pseudo-random playout is used.

Algorithm 1 Nested Monte-Carlo search

```
nested (level,node)
if level==0 then
  ply ← 0
  seq ← {}
  while num_children(node) > 0 do
    CHOOSE seq[ply] ← child i with probability 1/num_children(node)
    node ← child(node,seq[ply])
    ply ← ply+1
  end while
  RETURN (score(node),seq)
else
  ply ← 0
  seq ←{}
  best_score ← ∞
  while num_children(node) > 0 do
    for children i of node do
      temp ← child(node,i)
      (results,new) ← nested(level-1,temp)
      if results<best_score then
        best_score ← results
        seq[ply]=i
        seq[ply+1...]=new
      end if
    end for
    node=child(node,seq[ply])
    ply ← ply+1
  end while
  RETURN (best_score,seq)
end if
```

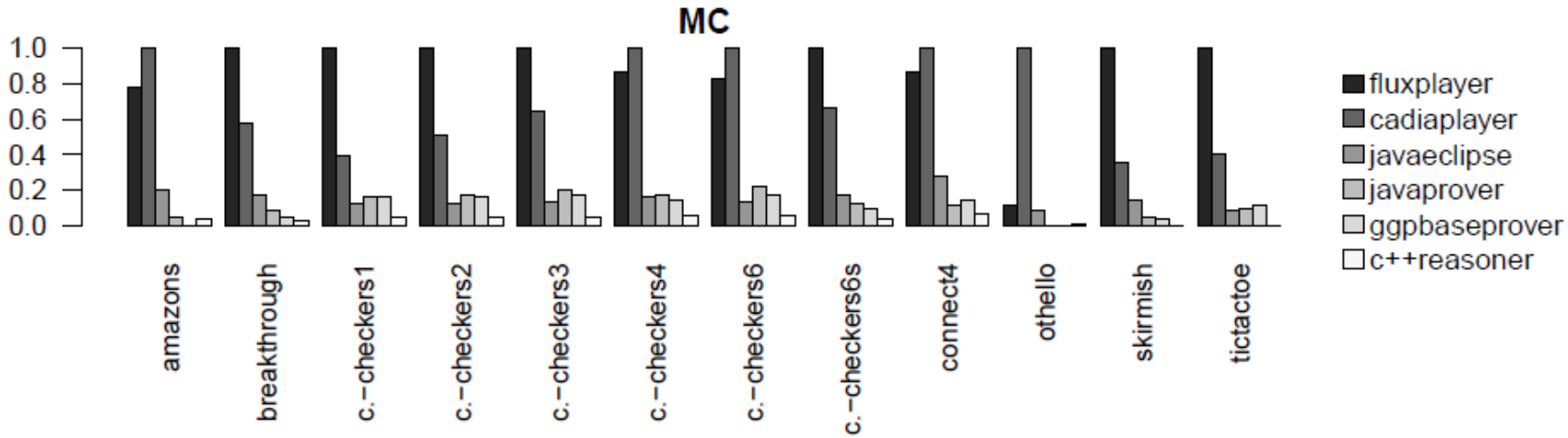
Evaluation

	MAX2		MAX1		NMC		UCT+T		UCT-T		# experiences
asteroidsparallel	71	±9	67	±11	68	±10	60	±12	59	±12	156/233/157/157/157/
asteroidsserial	85	±12	84	±12	85	±12	87	±12	86	±12	157/233/157/157/157/
duplicatestatelarge	67	±11	67	±8	51	±9	62	±17	42	±10	156/231/157/157/157/
duplicatestatemedium	95	±5	96	±5	84	±11	93	±12	53	±14	156/233/156/156/156/
hanoi	80	±0	80	±0	80	±0	80	±0	80	±2	156/232/156/156/156/
hanoi7 bugfix	70	±9	70	±9	58	±5	70	±10	49	±9	156/231/156/156/156/
incredible	75	±11	74	±15	71	±14	73	±11	50	±11	156/232/156/156/156/
knightmove	97	±4	97	±4	97	±4	95	±5	95	±5	157/232/157/157/156/
knightstour	100	±0	99	±1	96	±3	100	±1	100	±1	157/232/157/157/157/
lightsout	6	±23	2	±13	2	±13	3	±15	4	±19	155/232/156/156/156/
lightsout2	8	±26	3	±17	1	±11	5	±21	4	±20	157/232/157/157/157/
max knights	95	±8	87	±13	95	±9	68	±18	67	±18	156/232/642/619/620/
pancakes6	89	±1	89	±1	81	±4	89	±1	87	±3	156/232/156/156/156/
pancakes88	59	±9	54	±7	50	±4	61	±9	57	±9	157/231/157/157/157/
peg bugfixed	92	±3	91	±2	91	±2	90	±3	90	±3	156/232/156/156/156/
queens	76	±9	74	±9	79	±8	68	±10	70	±11	157/232/157/157/157/
statespacelarge	52	±8	50	±9	51	±8	42	±10	42	±11	156/232/156/156/156/
statespacemedium	84	±9	81	±11	83	±11	52	±12	50	±10	157/232/157/157/157/
sudoku simple	40	±7	36	±7	39	±7	32	±7	32	±7	157/231/530/467/467/
tpeg	99	±3	97	±4	98	±3	95	±7	94	±6	156/232/156/156/156/
total	1440		1398		1360		1325		1211		

Comparison of GDL reasoners

Competitors

- CadiaPlayer
- FluxPlayer
- JavaProver
- JavaEclipse
- C++Reasoner
- GGPBaseProver


MC
1.0
0.8
0.6
0.4
0.2
0.0
amazons
breakthrough
c.−checkers1
c.−checkers2
c.−checkers3
c.−checkers4
c.−checkers6
c.−checkers6s
connect4
othello
skirmish
tictactoe
fluxplayer
cadiaplayer
javaeclipse
javaprover
ggpbaseprover
c++reasoner

Conclusions

- The GDL reasoners are at least two to three orders of magnitude slower than their game-specific counterparts
- Of the GDL reasoners tested, FLUXPLAYER and CADIAPLAYER are by far the most efficient, both using Prolog engines as their backends
- The relative performance of the GDL reasoners can be quite game dependent.

Acknowledgements

- Tak et.el.: N-Grams and the Last-Good-Reply Policy applied in General Game Playing, IEEE Transactions on Computational Intelligence, 2012
- Mehat et.al.: Combining UCT and Nested Monte-Carlo Search for Single-Player General Game Playing, IEEE Transactions on Computational Intelligence, 2009
- Björnsson et.al.: Comparison of GDL Reasoners, GIGA 2010