General Game Playing (GGP)
Winter term 2013/2014

10. Summary

Sebastian Wandelt

WBI, Humboldt-Universität zu Berlin

General Game Playing?

- General Game Players are systems
  - able to understand formal descriptions of arbitrary games
  - able to learn to play these games effectively.

Prolog as the foundation for GDL

General purpose logic programming language

- Facts/Rules
- Unification
- Reasoning
  - Bottom-up
  - Top-down

Prolog: What should you remember?

- Syntax/Semantics
- Describe a domain of interest in Prolog
- Unification of terms and expressions
- Apply reasoning techniques to deduce entailment

GDL

- Logical foundation, Herbrand base/models
- Game-independent vocabulary
  - Role, init, true, next, etc.
- Game-specific vocabulary
  - How to express common game patterns?
    - Boards
    - Marks
    - Counter
    - Turn-taking games
    - Etc.

What makes a general game player?

- A player typically consists of the following parts:
  - Communication
    - Each player is a basic HTTP server waiting for messages from the Game Master and sending its moves as a reply.
  - Reasoning
    - Since a game description is essentially a logic program, the player has to use automatic reasoning or a logic programming system (e.g., Prolog) to infer legal moves and successor states.
  - Strategy
    - You need the communication and reasoning parts to play games. To win you need a good strategy. There are reference players available (here or here) to get you started.

GDL: What should you remember?

- Syntax/Semantics
- What other means of representation are there? Advantages/disadvantages?
- How to express common game patterns?
  - Boards, Marks, Counter, Turn-taking games, etc.
- Interpret/analyze a small game description
- Formal properties of games
  - Winnability, Playability

Naïve Search

- A **search strategy** is defined by picking the order of node expansion
- Strategies are evaluated along the following dimensions:
  - **Completeness:** does it always find a solution if one exists?
  - **Optimality:** does it always find a least-cost solution?
  - **Time complexity:** number of nodes generated
  - **Space complexity:** maximum number of nodes in memory
- Time and space complexity are measured in terms of
  - *b:* maximum branching factor of the search tree
  - *d:* depth of the optimal solution
  - *m*: maximum length of any path in the state space (may be infinite)

Comparison of search strategies

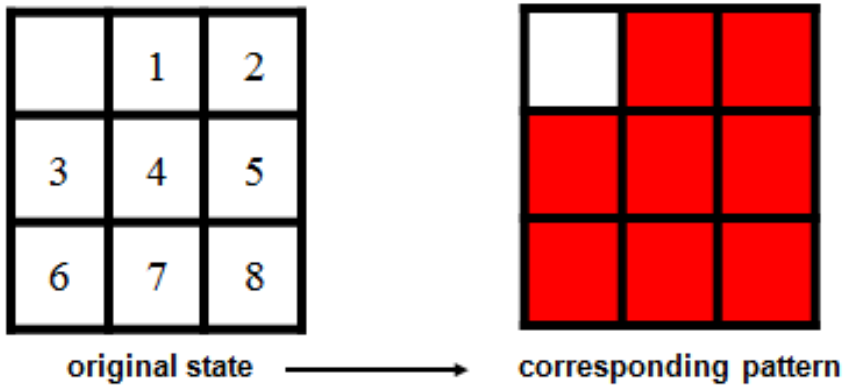
Algorithm	Complete?	Optimal?	Time complexity	Space complexity
BFS	Yes	If all step costs are equal	$O(b^d)$	$O(b^d)$
UCS	Yes	Yes	Number of nodes with $g(n) \leq C^*$	
DFS	No	No	$O(b^m)$	$O(bm)$
IDS	Yes	If all step costs are equal	$O(b^d)$	$O(bd)$
Greedy	No	No	Worst case: $O(b^m)$ Best case: $O(bd)$	
A*	Yes	Yes	Number of nodes with $g(n)+h(n) \leq C^*$	

Naïve Search: What should you remember?

- How do these naïve algorithms work?
- What is their advantages/disadvantages?
- Apply them to an example?

Pattern Databases

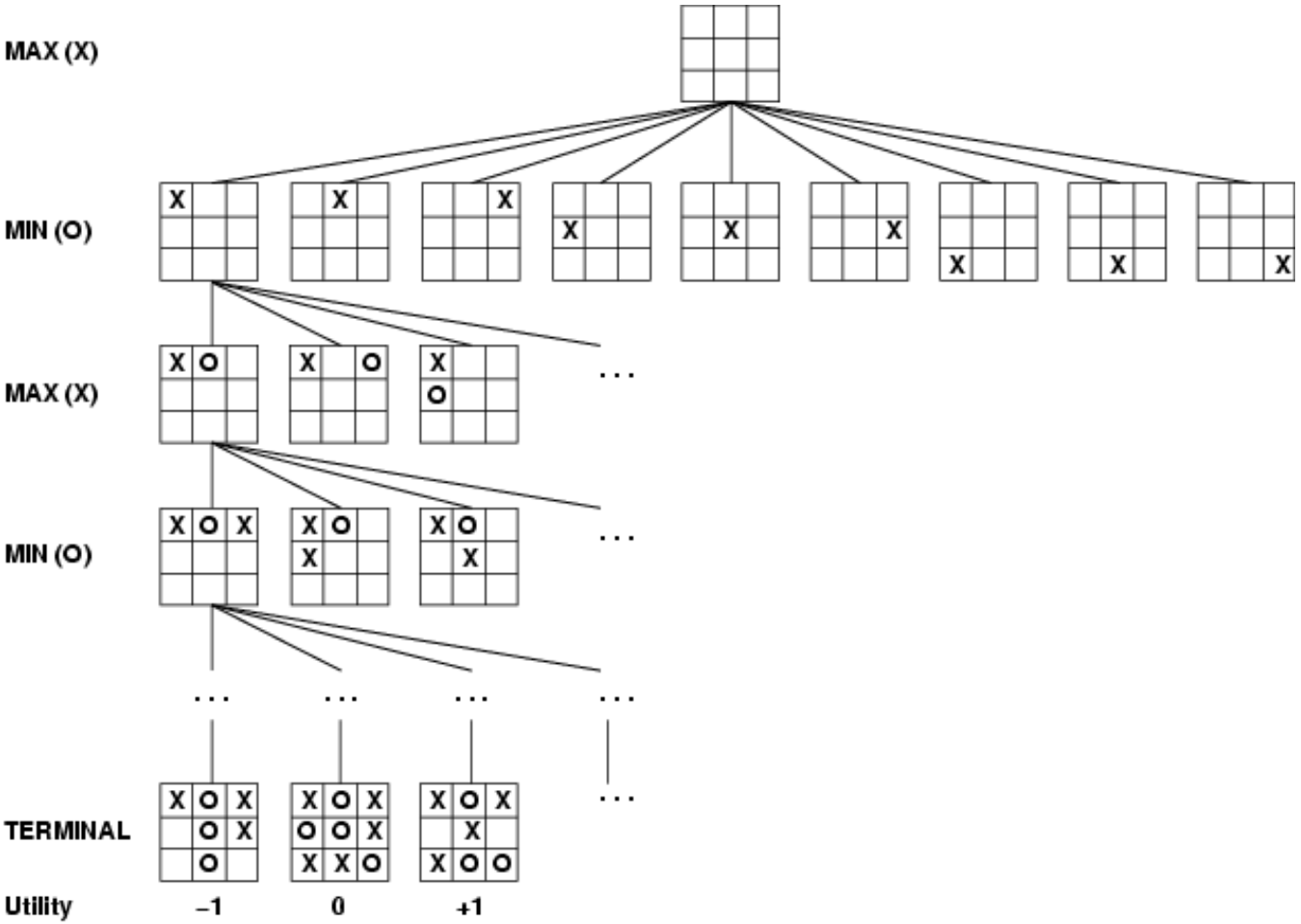
- Mapping of real state space into abstract state space
- Linear conflict heuristic
- Maxsort-heuristic



Pattern Databases: What should you remember?

- General idea, benefit?
- Compressed pattern databases?
- How to exploit pattern databases in GGP?

MinMax/alpha-beta-pruning



MinMax/a-b-pruning: remember?

- How to apply both heuristics to a game tree
- Advantages/disadvantages
- Horizon effect

Automated Feature Extraction

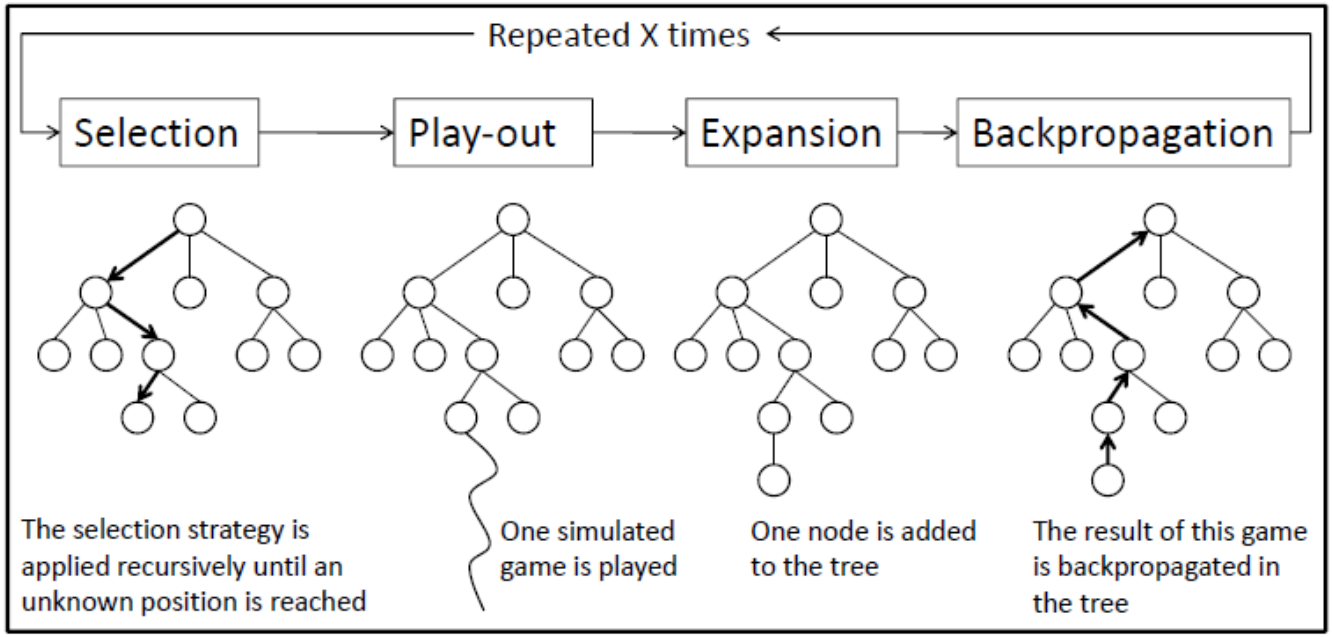
- Cluneplayer
  - Candidate expressions => interpretations => relevant features
  - Different interpretation functions
  - Notion of variance/stability
  - Abstract model for search

Parameter	Meaning
$P : \Omega \rightarrow [0, 100]$	Approximates payoff function.
$M : \Omega \rightarrow [-1, 1]$	Relative mobility.
$T : \Omega \rightarrow [0, 1]$	Proximity to termination.
$S_P : [0, 1]$	Relative stability of P.
$S_M : [0, 1]$	Relative stability of M.

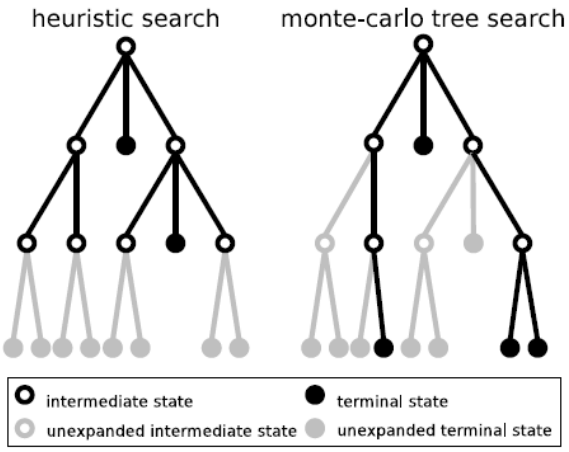
Automated Feature Extraction

- OGRE
  - (Syntactic) Extraction of turn counters
  - Feature extraction by variance
  - Motion detection by comparing consecutive game states
  - Different evaluation functions
    - Specific: Distance-initial, distance-to-target, counter pieces, …
    - General: Depth, number of moves

Monte-Carlo Tree Search



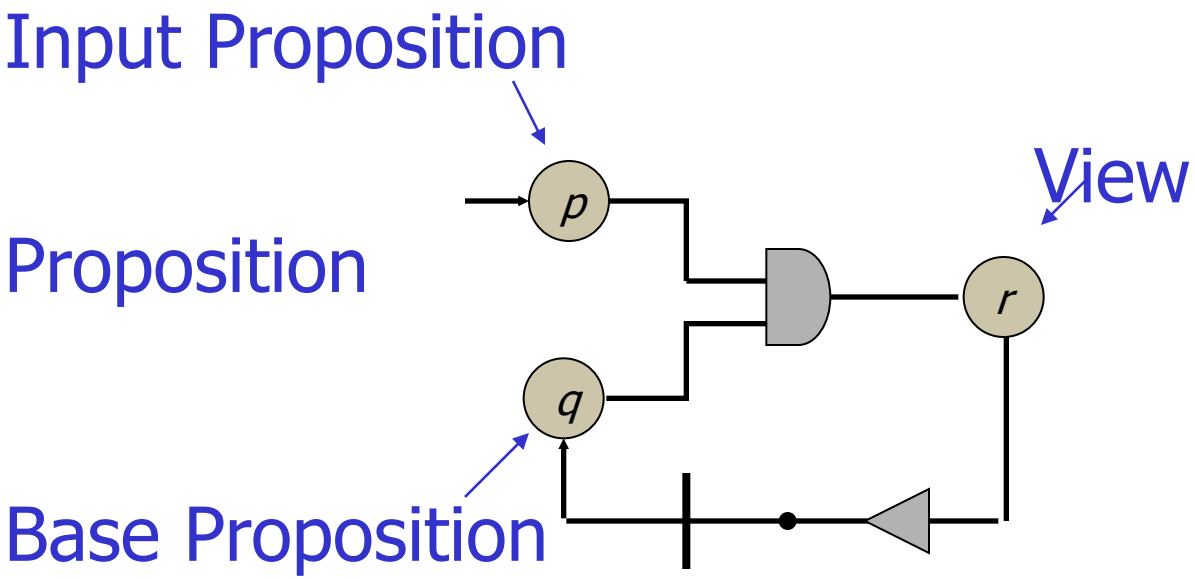
$$a_t = argmax_{a \in A_t} \left\{ Q(s,a) + C_p \sqrt{\frac{\ln N(s)}{N(s,a)}} \right\}$$



Extensions to MCTS

- Move-average sampling technique
- Tree-Only
- Predicate-Average Sampling Technique
- Rapid Action Value Estimation
- N-Grams and the Last-Good-Reply Policy
- Nested MCTS

Propositional Networks



- Syntax/Semantic
- Application: factoring of games, improve speed for small games
- Problem: grounding

FluxPlayer

- Compute degree of truth of GDL formulae using Fuzzy Logic
  - Generating state evaluation functions
  - Identifying common game structures using semantics
  - Distance estimation between states
  - Fluent graphs
  - Proving properties of games using induction

Additional techniques

- Game independent feature learning
  - Look at 2-ply trees only (identification of offensive/defensive features)
- Game instantiation (grounding of GDL)
  - Speed up computation
  - Exponential blow-up possible

Conclusions

- State-of-the-art in General Game Playing
- General game playing is mainly about search techniques
- Most techniques work only well for some classes of game
- Early techniques relied on the GDL syntax
- Most recent competitors use variants of MCTS
- Today, the major bottleneck is the reasoning component!
  - Limits the number of nodes you can unfold per second

- If you are interest in doing a thesis on GGP, let me know.