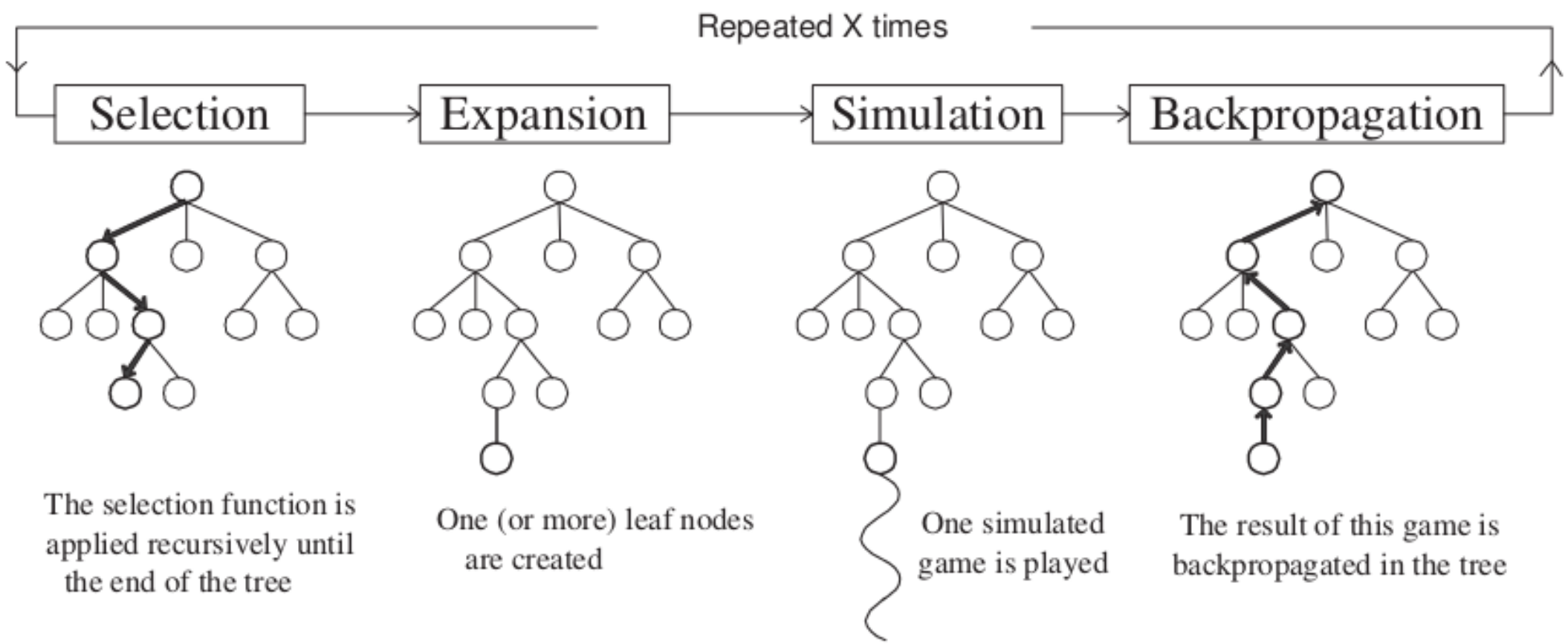
Game Algorithms
Go and MCTS

Petr Baudiš, 2013

Outline

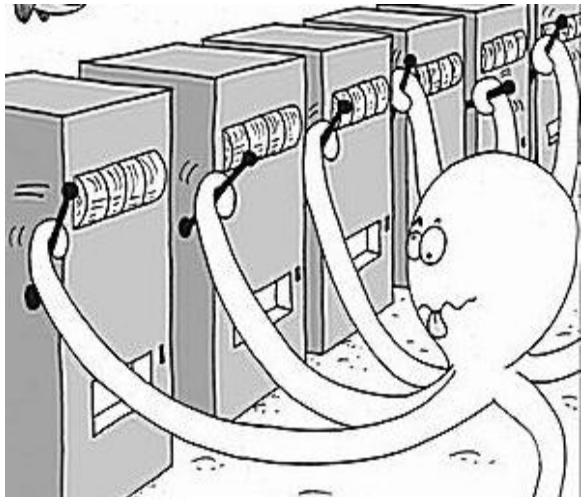
- Rehash: MCTS and bandits
- Enhancing the MC simulations
- Enhancing the tree search
- Automatic pattern extraction
- Information sharing
- Unsolved problems

Monte Carlo Tree Search



Multi-armed Bandit

- **=> Multi-armed bandit**
- Each node has *urgency* based on value and exploration desire
- **Urgency policy:** Minimize *regret* – expected total loss of selecting suboptimal nodes



$$\mu_i = \mathbb{E}\left[\frac{1}{T_i(n)}\sum_{t=1}^{T_i(n)} X_{it}\right]$$

- Several approaches: ε-greedy, upper confidence bounds

$$\mu^* = \max_i \mu_i$$

$$R_n = n\mu^* - \sum_{i=1}^{K} \mathbb{E}\left[T_i(n)\right]\mu_i$$

Upper Confidence Bound

- *urgency* = *value* + *bias*
- *value* = *expectation* = *wins / simulations*
- *bias* = UCB1 (Auer, 2002)
  upper bound on possible value

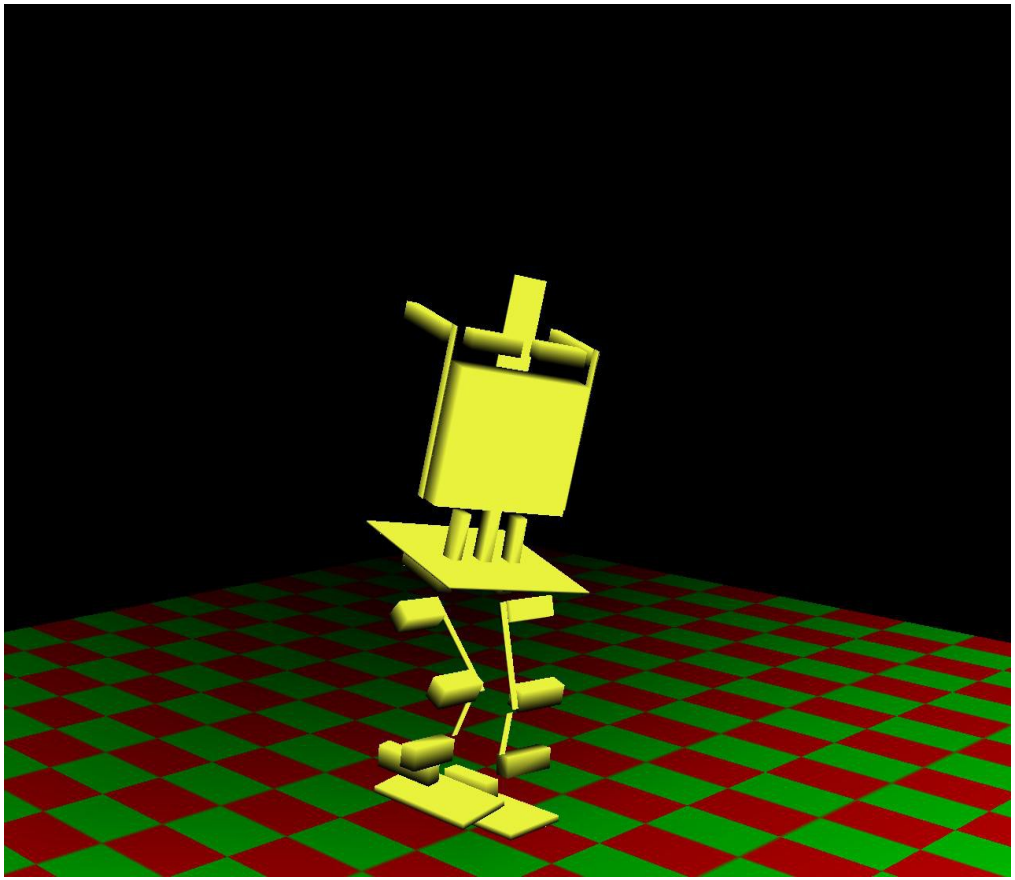
$$\sqrt{c\frac{\ln(n_0)}{n}}$$

$$\pi_{UCB1}(n) = \operatorname{argmax}_i \left(\mu_i + c\sqrt{\frac{2\ln n}{T_i(n)}} \right)$$

- *c* is parameter; best for random Go ~0.2
- **Optimistic** strategy – try most *promising* node

Inconsistent formalisms

Better Simulations



Basic Implementation

Trivial Heuristics

Local Patterns

Caveats!

Uniformly Random...

- In each move, pick a random element from the set of legal moves \ pass
- Never fill single-point eyes
- **Common termination rule:**
  - Pass only if no valid move remains
  - => Easy + fast counting
  - Mercy rule

appendix 4

Playout Requirements

- **Speed** – more simulations mean deeper tree and more accurate values
  - Small board, light playouts: Tens of thousands playouts per second
  - Large board, heavy playouts: ~2000 pPs
- **Plausibility** – situations should be resolved like in a real game

x

- **Balance** – all reasonable results should have the chance to appear in playouts (**x** policy drift)

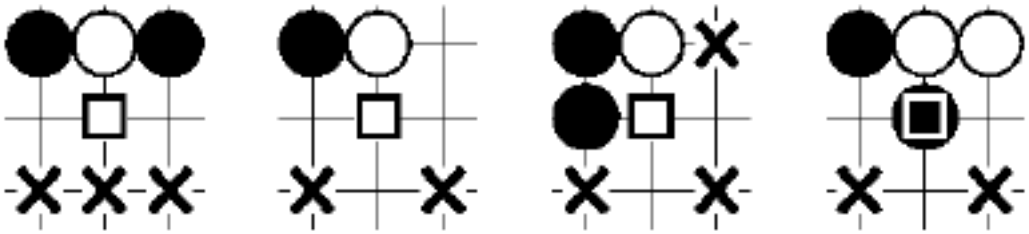
Simple Heuristics

- Hard to find heuristics that don't fail often
- Capture stones in atari vs. escape with stones in atari (possibly detect ladders)
  - Except when the stones cannot escape
- Do not self-atari – *but sometimes do!*
  - Putting large group in atari instead of connecting is bad
  - Self-atari of your stones in opponent's dead eyespace is necessary
- 2-liberty tactics similar to atari tactics

3x3 Patterns

- ~10 wildcard 3x3 patterns centered at the candidate move (Gelly, 2006)
- Check only around last move; play on match
- => Produces "nice" local sequences
- 3x3 patterns = 16bit numbers => Very fast
- Optional: atari info in four directions (+4 bits)

appendix 5



Balanced Patterns

- Stronger playout is not better playout!
  - Imbalance => consistently biased assessment of position, UCT misbehaves
- Fresh approach – machine learning of patterns based on playout balance, not strength
  - (Silver, 2009) Don't minimize *error* but *expected error* – error over multiple moves in row (small mistakes cancel)
  - (Huang, 2010) Works on 19x19 too

Better Tree Search

Prior Node Values

All Moves As First

Rapid Action EValuation

Criticality

Dynamic komi

Multithreaded Search

Time Management

Fresh Nodes

- UCT: Play each node once first – too ineffective
- **First Play Urgency:** Initialize *urgency* with fixed value (~1.2), start UCB-selecting nodes
- "Progressive widening", initialize *value* heuristically
- "Progressive unpruning", rank nodes heuristically, consider only *f(n)* best nodes

Prior Values

- **Priors:**
  - Playout policy hinting – capture, atari, 3x3 patterns, eye filling
  - Distance from the board border
  - CFG distance from the last move
  - Smart static evaluation function

Common Fate Graph

(Graepel, 2001)

- Intersections: vertices, lines: edges
- Edges between same color: d=0, others: d=1
- CFG distance: the shortest path in CFG
  - Useful for the concept of "tactical locality"
  - Takes into account all moves affecting local groups

All Moves As First

- UCT converges very slowly, especially on large boards – no information sharing
- Idea: Find out and prefer moves that give good performance in all games (Bruegmann, 1993)
- *UCT value of M*: Winrate of games starting by *M*
- *AMAF value of M*: Winrate of games where we played *M* anytime in the rest of the game(!)
- Moves in-tree and in most of the playout are considered (late moves cut, or weighing)

Rapid Action Evaluation

- How to incorporate AMAF in the node value? (Gelly & Silver, 2007)
- *value* = $\beta \times$ *amafval* + *(1-*β*)* $\times$ *uctval*

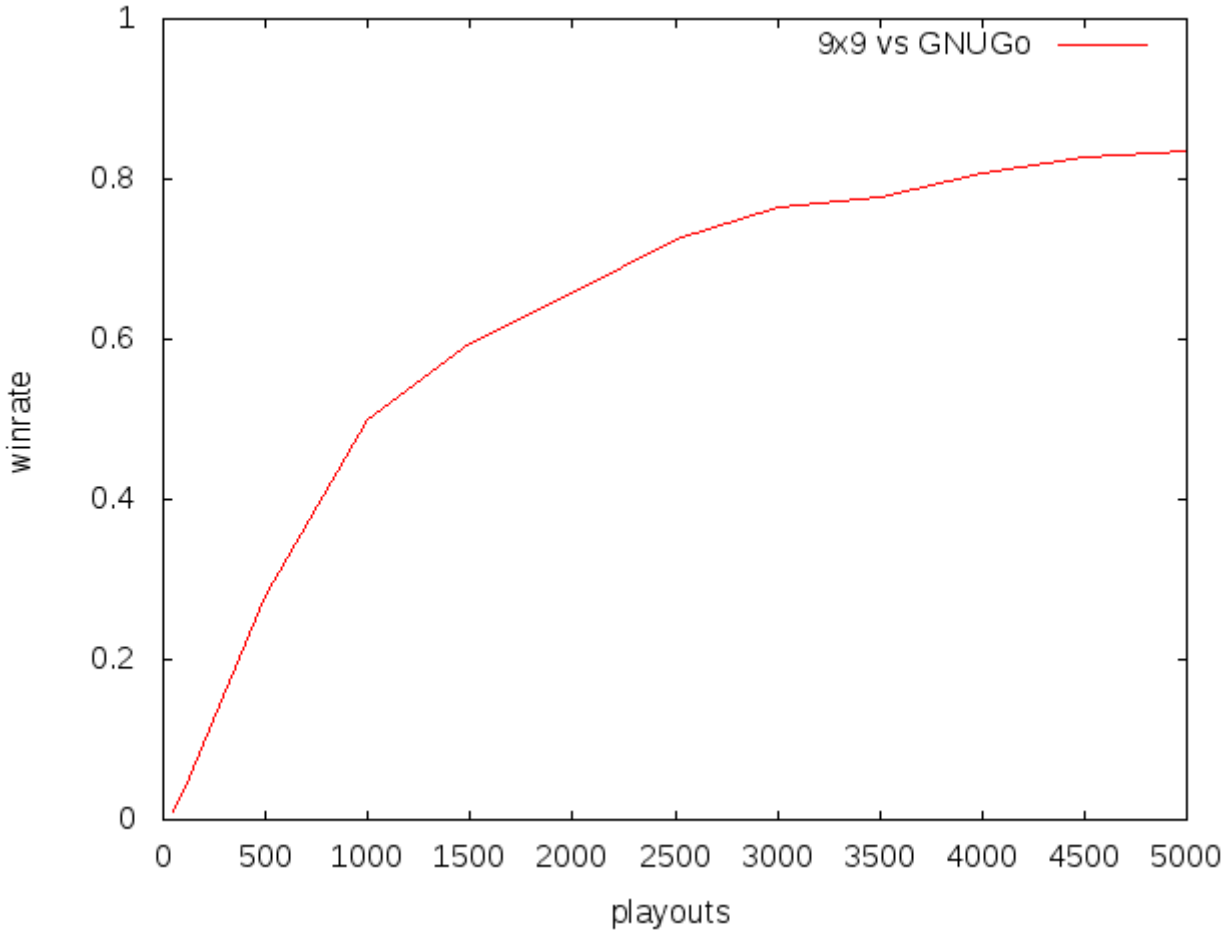
$$\beta = amafsims \times \left(amafsims + uctsims + \frac{amafsims \times uctsims}{r} \right)^{-1}$$

- With small *uctsims*, $\beta \sim 1$, but goes $\rightarrow 0$
- *r*: RAVE weight ("equivalence") parameter, e.g. ~3000

RAVE Aftermath

- **Key result** in MCTS Go, making it stronger than the classical engines:
  - ~ 30% UCT → 70% UCT-RAVE
- Good playout policy is crucial for good AMAF!
- Priors: *amafval* vs *uctval* – small difference
  - Important new prior: "Even game" *p=0.5* protects against inaccurate first results
- No exploration: Best results with *c=0* on 19x19 (*c=~0.005* on 9x9) – AMAF is sufficiently noisy
- Alternatives exist, though

RAVE Performance



Parallel MCTS

- **Root-level**
- **Leaf-level**
- **In-tree**

Parallel MCTS

(Chaslot, 2008)

- **Root-level** – independent search in each thread, merge at the end
  - Threads "vote" on best move
  - Slight-to-medium improvement, does not seem to scale much
- **Leaf-level** – single thread searches, all threads play in parallel
  - More accurate node value
  - Small improvement, large overhead

Parallel MCTS in-tree

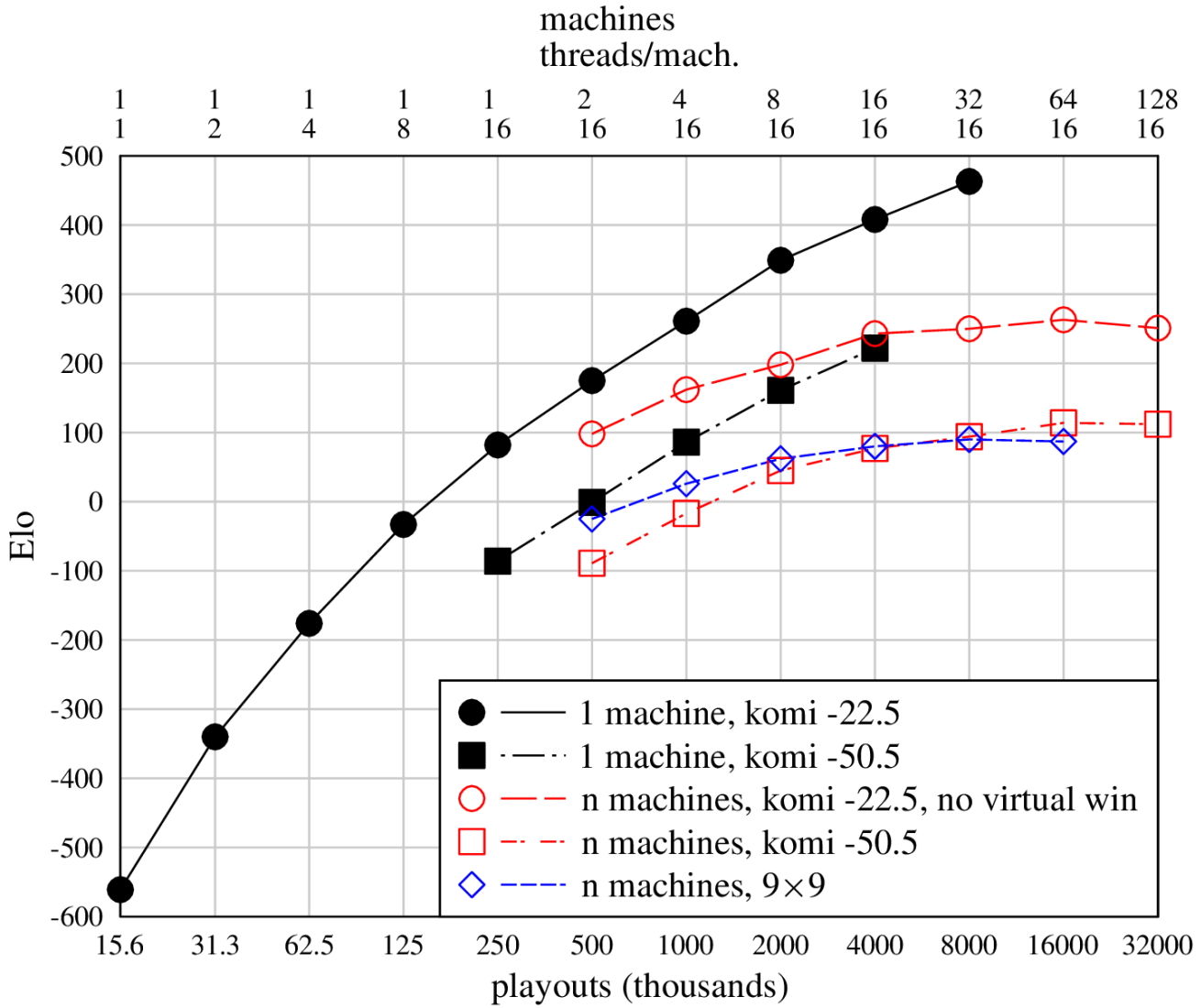
- **In-tree** – all threads search in the same tree
  - No locking necessary if we are careful (Enzenberger, 2009)
  - Never delete nodes during search
  - Update values atomically
  - *Virtual loss* spreads exploration (add loss in descend, remove during update)

Distributed MCTS

- **Distributed** – cluster of machines (nodes) with separate trees
- Independent searches + information exchange
- Information exchange = higher overhead
- Best: *Little* exchange, e.g. only single level
- Virtual wins (Baudiš and Gailly, 2011)

Parallel Performance

(19x19 vs Fuego)



Time Management

- How to allocate time during the game?
- Main time, overtime *n* periods of *m* moves
- Pachi: *Default* and *maximal* time,
  unclear results imply overspending
- Allocate most time in the “middle game”

Learning Patterns

Pattern Features

ELO Pattern Ranking

Storing Patterns

Pattern Usage

Pattern Usage

- Wildcard 3x3 centered patterns: see before
- Circular *n*-radius patterns – hash matching
- Arbitrarily shaped patterns: incremental decision trees

- Shape matching only
- Tactical goal matching
- Point owner matching

- Used both in playouts (simplified) and in priors (full features set)

Zobrist Hashing

- Hashing board positions (Zobrist, 1990)

Zobrist Hashing

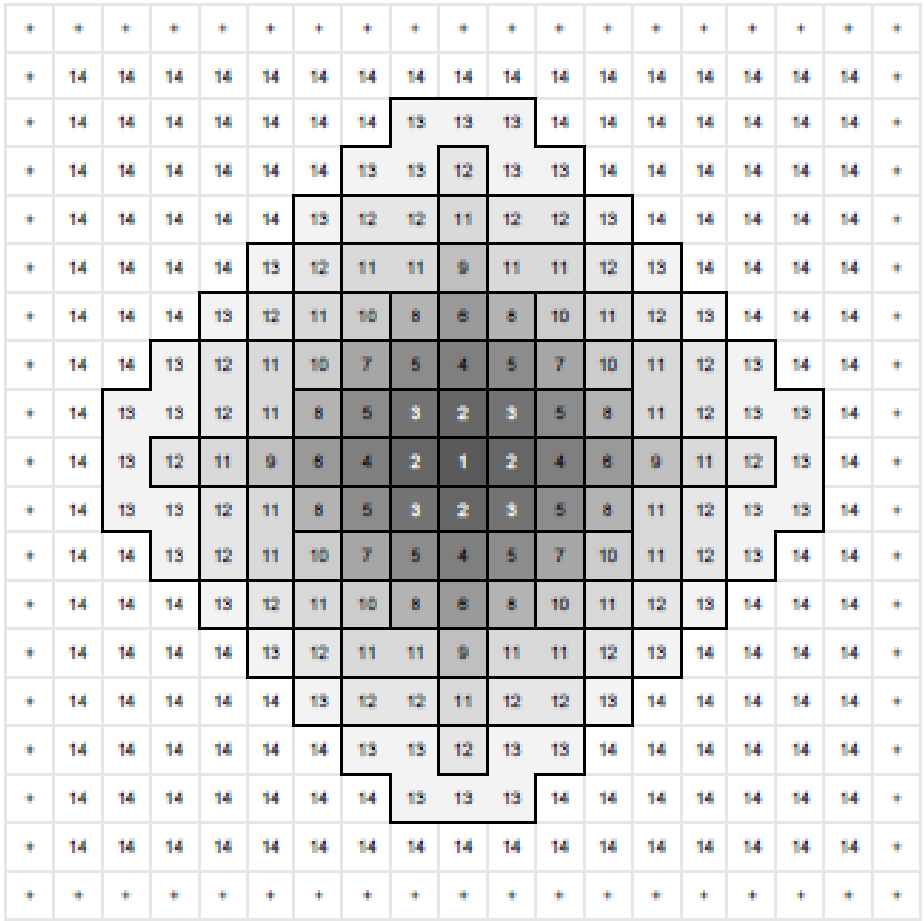
- Hashing board positions (Zobrist, 1990)
- Initialization: Each point gets assigned random numbers *b, w*
- Position: XOR of *b* values for all black stones and *w* values for all white stones
- Good uniform distribution, reasonable hash size
- Incremental updates on move plays possible!

Shape Patterns

- Represented as Zobrist hashes of the area
  - All rotations and color reversals
  - Matching can be incremental for multiple shape sizes
  - Lookup is very fast
- Extended board with special “edge color” - already common in fast board implementations

Circular Shapes

- ...on square grid? (Stern, 2006)
- Metric?

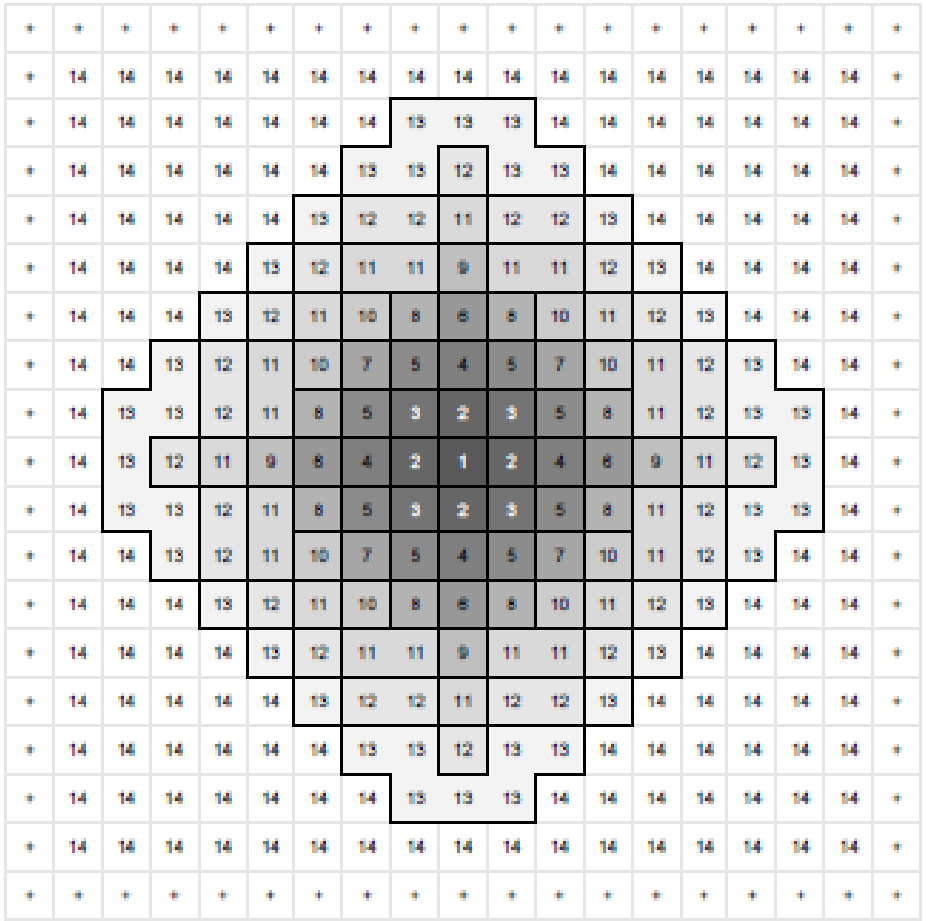


Circular Shapes

- ...on square grid? (Stern, 2006)
- Metric:

  d(x,y) = |dx| + |dy|
  + max(|dx|,|dy|)

- Incrementally matched nested circles
- Commonly used



Arbitrary Shapes

- Hard to recognize and harvest automatically, useful mostly for expert patterns
- Use probably uncommon

Arbitrary Shapes

- Hard to recognize and harvest automatically, useful mostly for expert patterns
- Use probably uncommon
- Proposed method: Incremental Patricia trees (Boon, 2009)
  - Build a decision tree (node-per-intersection) from the patterns
  - For each intersection, store nodes from decision trees
  - When the point changes, re-walk branch

Pattern Features

- For each candidate move, a pattern is matched:
- Shape – as just described
- Capture, atari, selfatari, liberty counts, ko... (van der Werf, 2002)
- Distance to the last, next-to-last move
  - CFG distance or circular distance
- Monte Carlo owner – portion of simulations where I am point owner at the game end
- Each feature can have its zobrist hash

Naive Pattern Probability

- Simple but powerful idea – just ignore all pattern context (MoyoGo - de Groot, 2005)
- Pattern for each spatial context occuring at least twice; use largest context available
- *Pattern play probability* is probability that a pattern is played when it is available to play
- Use *pat. play prob.* for probability distribution
- More complex models (except Elo) not so successful; speed and model suitability?

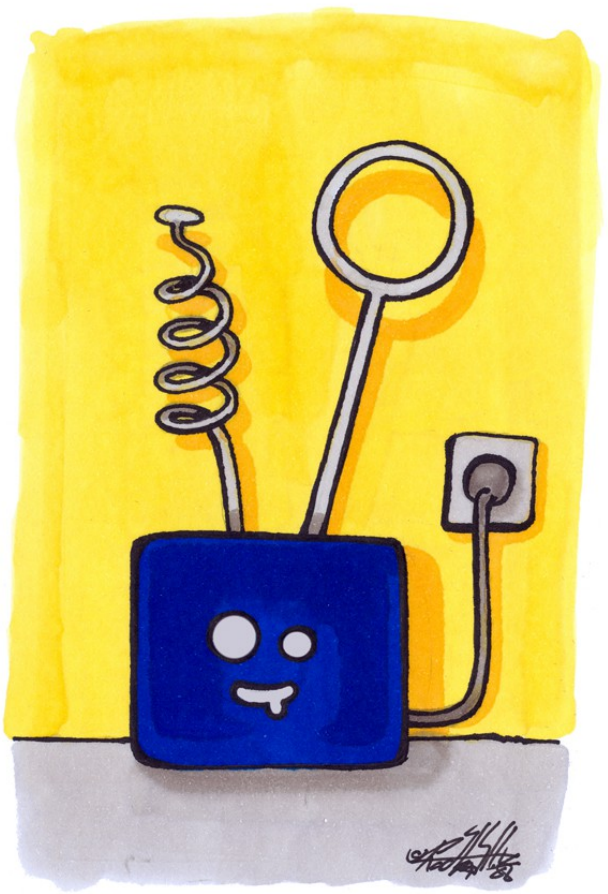
Elo Ratings

- Elo: Putting competitive strength of many individuals on a single scale (Elo, 1978)
- Used in Chess and Go to rate players strength
- Based on **Bradley-Terry model**:
  - Each individual has *strength* γ
  - $P(i \text{ beats } j) = \gamma_i / (\gamma_i + \gamma_j)$
- Works for competition of >2 players too
- Works for teams: $\gamma_1\gamma_3 / (\gamma_1\gamma_2\gamma_3 + \gamma_1\gamma_2 + \gamma_1\gamma_3)$
- Makes rather strong assumptions

Elo Patterns

- **Key result:** 38.2% → 90% (Coulom, 2007)
- Consider *teams of pattern features*, assign each feature its "strength"
  - capture=30, atari=1.7 self-atari=0.06
- Total strength of each intersection is product of the features strength
- Produces probability distribution over moves
- Use to choose the next move in playout; only easy features (e.g. shapes up to 3x3) are used
- Use to progressively unprune nodes

Current Programs



- Mogo – UCT pioneer
- CrazyStones – Elo
- ManyFaces – UCT+classic
- Zen – Elo reimplemented?
- Erica – Elo + Balancing

Opensource UCT:

- Fuego – complex, general
- **Pachi** – simple, Go focus

State of Computer Go

- 1998: Jean-loup Gailly 5k won 17-handi
- March 2012: Takemiya Masaki 9p lost giving 5 and 4 stones to Zen
- Bonn 2012: Motoki Noguchi 6d lost 0-2 on 9x9, 1-1 on 13x13 against Zen
- Bonn 2012: Catalin Taranu 5p 1-1 on 19x19 giving 4 stones to CrazyStone
- KGS (blitz): Zen19D 6d; CrazyStone 5d; GinseiIgo 5d; Pachi2 4d

http://computer-go.info/

Pachi

- Densely-commented C code, about 17k LOC
- Modular architecture for play engines (random, playout, MonteCarlo, UCT)
- Modular architecture for UCT policies (UCB1, UCB1AMAF/RAVE)
- Modular architecture for playout policies (random, “Moggy”, probability distribution)
- Modular dynamic komi policy, priors, etc.
- **Autotest** – generic UNIX framework for testing of stochastic engines performance

Information Sharing

- RAVE
- Dynamic komi
- Narrow sequence problem:
  - Criticality
  - Liberty maps
  - Local trees?

Playing in Extreme Situations

- **Extreme situation:** The computer has either a huge advantage or a huge disadvantage
- Common in handicap games
- Black: **big advantage** – suboptimal moves, no account for difference in strength
- White: **big disadvantage** – the problem is not so visible and harder to solve
- Interpretation: Too low signal-noise ratio when the outlook is extreme

Black in Handicap

- Linear dynamic komi, situational dynamic komi, artificial passes
- **Dynamic komi:** Before counting the final position in the simulation, subtract a certain amount of points from black score (Baudiš, 2011)
- **Situational komi:** Adjust the komi to keep probabilities between ~[0.4,0.5]; universal (not only handicap games), ~57% self-play
  - Fixed step or avgscore-based step

Linear Dynamic Komi

- **Linear DK:** Calculate komi value *K* based on the handicap amount
- K ~= *-cH* where *c* is point value of handi stone
  - c=8 (based on default komi value) seems optimal; non-linear scaling experiments discouraging
- Apply for first *M* moves: *k = K(1-m/M)*
- *M=200* works well on 19x19
- **Adaptive:** Keep winrate between 0.85 and 0.8

Handicap Performance

(19x19 vs GNUGo level 10)

Criticality

- Focus on places that are "key" for both players – the point is important for winning the game
- Similar to AMAF, but statistical covariance of winrates for *both* players

$$\frac{v(x)}{N} - \left(\frac{w(x)}{N} \frac{W}{N} + \frac{b(x)}{N} \frac{B}{N} \right)$$

- How to use it? Patterns (Coulom, 2009), another UCB term (Pellegrino, 2009) or proportional virtual AMAF wins (Baudiš, 2012)
- Small improvement (49% → 54%)

Liberty Maps

- Playout heuristics suggest a set of moves to choose from in given situation
- Collect statistics of *expected local value* of such moves, we want to prefer good ones
- Sharing statistics between different situations: *liberty map* of the local group (Baudiš, 2012)
- How to use the statistics?
  - Random move with eff. over threshold
  - Bandits! Use UCB to choose move in sim.
    Followup moves: Tree-like structure

Local Trees

- A pair of trees (black, white) of all non-tenuki sequences
- Parallel descent of main and local trees; way to share information about solutions of local situations?
- Using information: Local sequence forcing?
- Still in research...

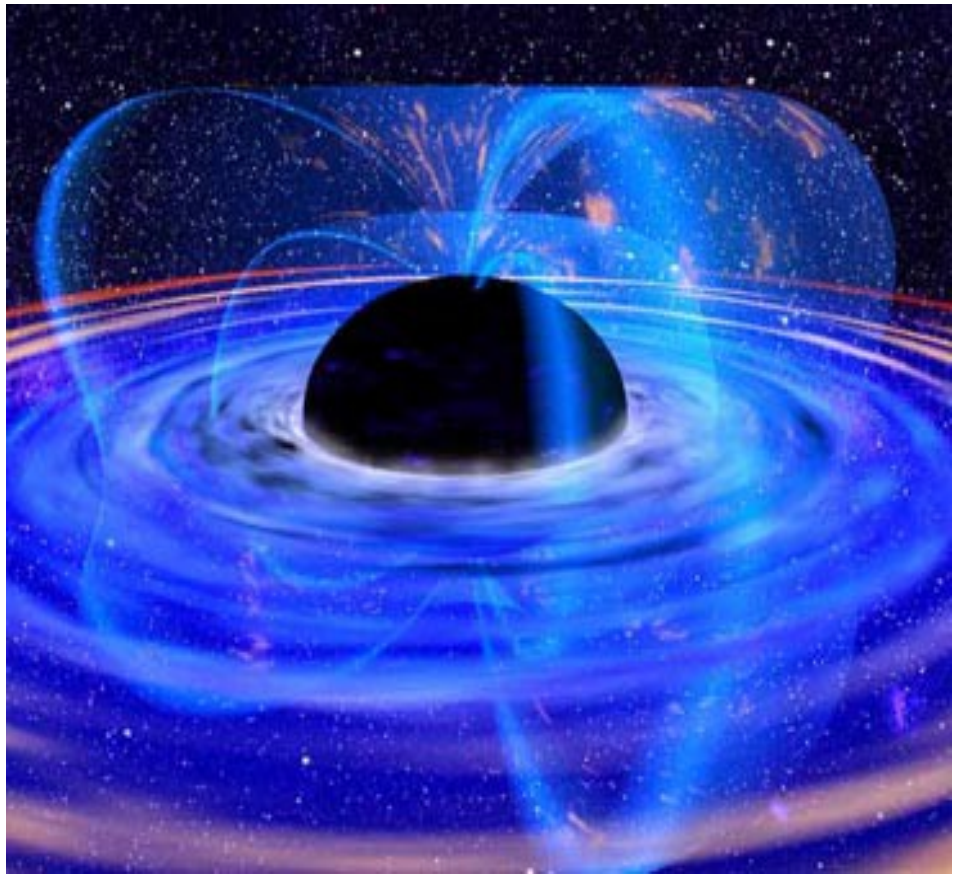
Unsolved Problems

Narrow sequences

HPC implementation

Aesthetically pleasing play

Abstract understanding of the board



Narrow Sequences

- The most visible and probably most important current issue
- UCT/RAVE bots miserably fail in most semeai situations, some classes of unsettled tsumego and sometimes even misread simple ladders
- RAVE gives single-level information, same problem as Monte Carlo vs UCT

Narrow Sequences: The Problem

- General situation description: After one player's move X, the other player has one right reply Y* (winrate converges) and many wrong replies {Y-} (winrate diverges)
- All replies have equal simulation probability, giving player's move X too high winrate
- Thus, RAVE gives the move massive bias everywhere in the tree; tree quickly discovers Y*, but this only pushes X down in tree

Narrow Sequences: Solutions?

- Common: Enhance simulations to natively choose Y* after X with high probability
  - Simulations must be fast, only static evaluation reasonably possible, case-by-case, tedious
- Prefer best local moves found by tree search in simulations?
- Pre-bias node values based on local sequences found in other tree branches?
- Preliminary results promising, still researching

High Performance Computing

- Big clusters tried – Mogo on 900 cores etc.
- Mix of root and tree parallelization
- Scaling limits: overhead, limited information sharing
- GPGPU needs a lot of research, preliminary experiments not too encouraging
  - Game parallelization – playout / thread
  - Point parallelization – intersection / thread

Aesthetically Pleasing Play

- Computers like to play “strange-looking” moves
- Unclear if solving these problems would improve win rate
- Playing opening moves very far from the edge
- Playing suboptimal moves at the game end when win is secured

Abstract Understanding

- Useful since simulations cannot be deep enough to assess true values of some aspects
- E.g. solidness of territory and groups, thickness value, ko fights status, latent aji
- Maybe ManyFaces does it to a degree, no published results; can be obsoleted by narrow sequences solution
- Describe point/chain dynamics as polynomial system (nice prediction results, in research – Wolf, 2009 preprint)

Thank you!

pasky@ucw.cz
http://pasky.or.cz/go/

http://senseis.xmp.net/

http://gokgs.com/
http://computer-go.org/
http://www.citeulike.org/group/5884/library