Current Approaches in Computer Go

Petr Baudis, 2009

Outline

- What is Go and why is it interesting
- Possible approaches to solving Go
- Monte Carlo and UCT
- Enhancing the MC simulations
- Enhancing the tree search
- Automatic pattern extraction
- Unsolved problems

What is Go

- History
- Concepts
- Rules
- Basic Tactics

The Go Board Game

- Go / Igo / Goe / Baduk / Wei-Qi
- ~3000 years old - the oldest board game
- Very simple rules, very high complexity
- Wide-spread in China, Korea, Japan
- Rich culture surrounds the game
- http://senseis.xmp.net/

Go: Basic Concepts

- Square board with 19x19 intersections
  - Small board variation with 9x9
- Black and white players alternate in placing stones on the intersections
- Stones do not move; they can be removed if completely surrounded
- Players surround territory and capture enemy stones

appendix 1

Go: Capturing Stones

- Directly connected stones == *group*
- #of unoccupied intersections around group == *liberties*
- When group has no liberties, it is removed
- Removed group: *capture*; single lib.: *atari*
- Ko rule - later

Go: Tromp-Taylor Rules

- Players place stones alternately
- If the board is filled, players play “pass”
- The player controlling more intersections wins
- *Eye:* empty places completely surrounded by stones of one color
- Controlling intersection: Either occupied by a stone, or an eye of given color
- *Komi*: Point bonus for white

final position – appendix 4

Go: Other Rulesets

- Many Go rulesets: Tromp-Taylor, Chinese, Japanese, ...
- *Tromp-Taylor*: Formal, terse, easy for computers
- *Japanese*: Easier for humans, most common, hard for computers; slightly different counting
- All rulesets are equivalent or 1pt-equivalent in common situations

Go: Life and Death

- So much for the rules; now basic tactics!
- Group is *alive*: Can form *two eyes*
- Group is *dead*: Can be always captured locally
- Group is *in seki*: Cannot form two eyes, but opponent cannot capture it
- *Semeai*: Capturing race between two groups

Go: Tactical Concepts

- *Semeai*: Capturing race between two groups, the one which captures first also kills the other
- *Ladder*: Player keeps escaping, but opponent always plays atari and eventually captures
  - Extremely long move sequence, but easy even for beginners to read
- *Net*: Player plays a distant move preventing enemy group from escaping

appendix 2

Go: The Ko Rule

- *Ko*: The same board position cannot repeat in single game
- To re-take ko: Play a *ko threat* elsewhere on the board
  - Opponent replies and ko can be re-taken
  - Opponent connects ko and you can follow up on the threat
- Group is * *in ko*: Goal can be achieved if player wins a ko fight

Go: Strategic Concepts

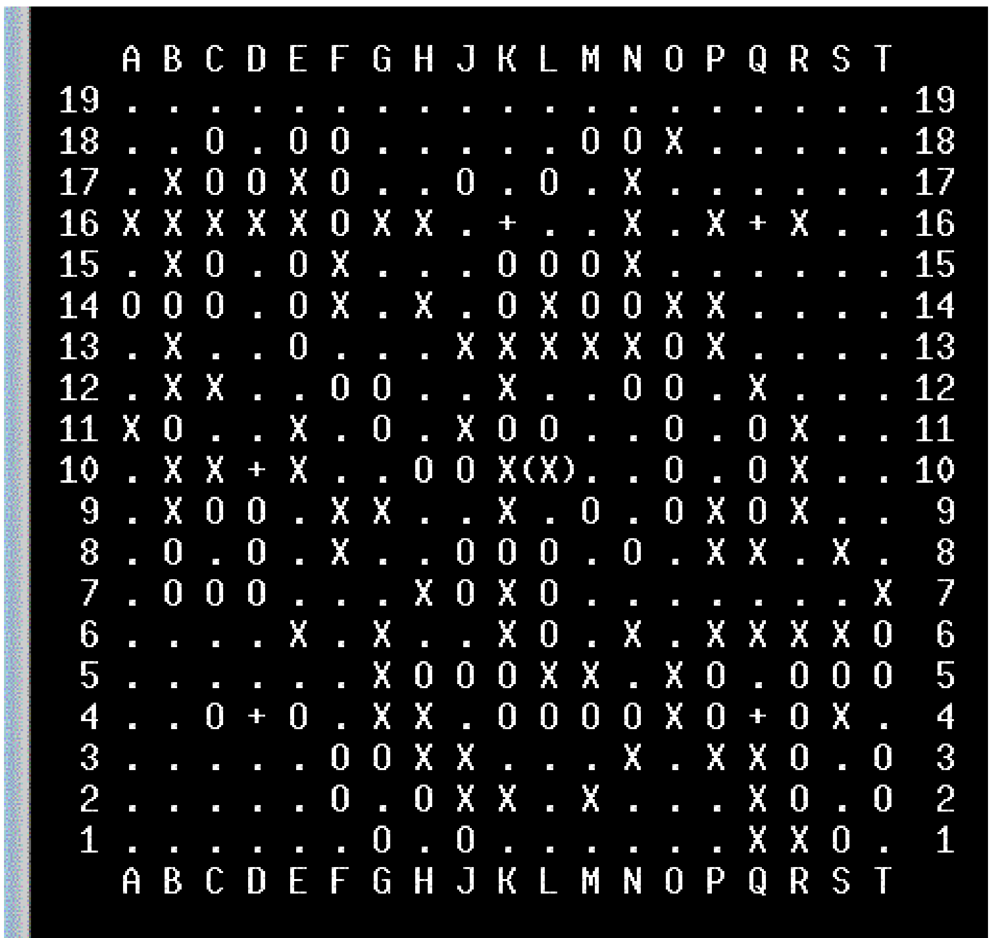
- *Territory*: Empty area where opponent cannot make live group anymore
- *Moyo*: Territorial framework part of which can be still reduced by the opponent (at the cost of turning the rest to territory)
- *Influence*: Using hard-to-kill group to attack weak group of the opponent

appendix 3

Ranking in Go

- Several rating systems
- We will use KGS server ranking system:
  - 30kyu ... absolute beginner
  - 15kyu ... average beginner after 4 weeks
  - 5kyu – 1kyu ... intermediate player
  - 1dan – 9dan ... advanced to expert ama.
  - 1pro – 9pro ... professional player
- Handicaps based on rank difference

Solving Go



- The Problem
- Special Sub-Problems
- Possible Approaches
- Classic Solutions

What's So Hard?

- Extreme branching factor
  - Chess: 10^{126} ; Go: 10^{360}
  - Transposition tables are ineffective
- Evaluation function is difficult
  - Has to take into account changing status of stones
  - Influence, territory-moyo hard to assess
- Pruning branches is difficult
  - Universal pruning function hard to find

Specialized Sub-Problems

- Playing perfect late endgame (Berlekamp,1994)
  - Combinatorial Game Theory, performs better than professional players
  - Does not scale before last few moves
- Solving tsumego problems
  - Small board sub-section, short sequence
  - Best solvers can find the move in few seconds (Wolf, 2007)

How To Do It?

- alpha,beta search + hand-coded patterns
  - “GNUGO”, ~6kyu
- Neural networks, pure (auto-gen.) patterns
  - Unsuccessful in general (~15-20kyu?) (Ezenberger, 1996)
- Monte Carlo, Monte Carlo Tree Search
  - Most modern bots, on commodity HW up to ~1-2dan (on 9x9, up to ~4dan?)

Classic Approach

- GNUGO – complex classic knowledge, many hand-coded patterns, alpha,beta search
  - Very useful test opponent for MC bots
- Frequently misses moves – overpruning
  - Causes major tactical mistakes
- Drastic misjudgements of group status
- Points-greedy move choice (cannot adjust style for disparate situation)
- Strength does not scale with time

Monte Carlo and UCT

Monte Carlo Approach

Multi-armed Bandits

Upper Confidence Trees

Monte Carlo Go

- Basic idea: evaluate a position by playing many random games (simulations) and averaging the outcome
- Primitive: Run N simulations for each valid move, pick the one with best value (reward) (Bruegmann, 1993)

Monte Carlo Go

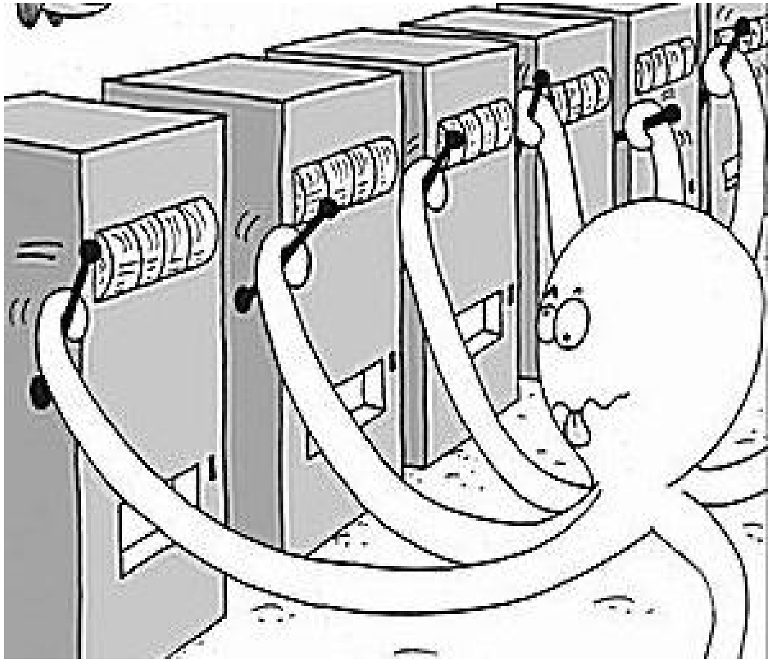
- Basic idea: evaluate a position by playing many random games (simulations) and averaging the outcome
- Primitive: Run N simulations for each valid move, pick the one with best value (reward) (Bruegmann, 1993)
- Outcome coding:
  - points_difference: too unstable
  - 0,1 (loss,win): usual approach
  - 0.01 for pts difference is slight bonus

Monte Carlo Tree Search

- Primitive MC cannot converge to best result
  - Does not discover forced sequences
- Tree Search: Explore best replies of best replies of best replies of best replies of best moves... (minimax tree)
- Exploration vs exploitation:
  - Focus simulations on the best candidates
  - Make sure we know which are the best

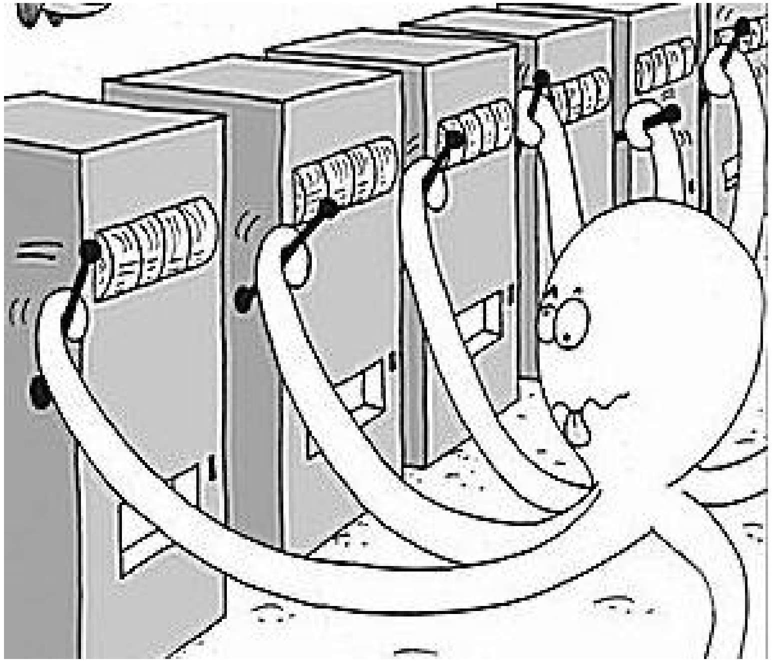
Multi-armed Bandit

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



Multi-armed Bandit

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



- Several approaches: ε-greedy, upper confidence bounds

Upper Confidence Bound

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

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

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

UCB1 Hardcore

(supplementary slide)

- (Lai & Robbins, 1985) Maximum regret:

$$E[T_j(n)] \leq \left(\frac{1}{D(p_j \| p)} + o(1) \right) \ln(n)$$

- D(P|Q) – Kullback-Leibler divergence

$$D(P \| Q) = \int P \ln\left(\frac{P}{Q}\right)$$

- In these policies, optimal node is selected exponentially more often than second best

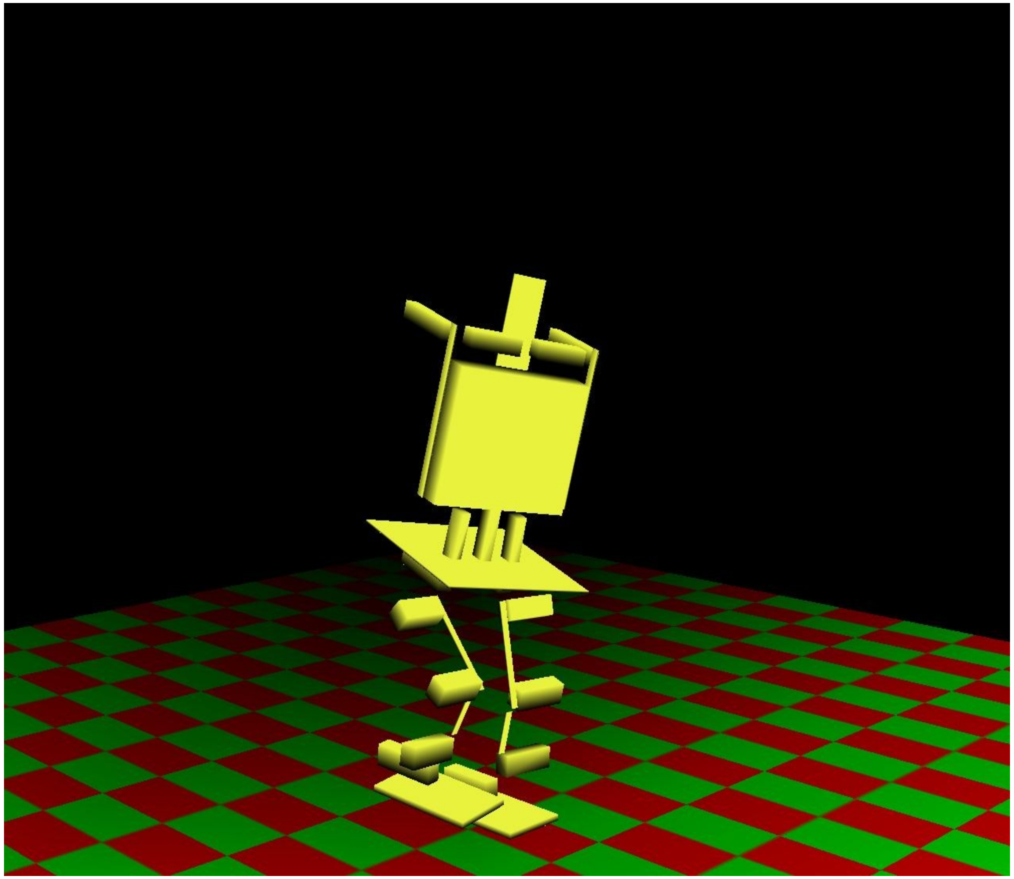
Upper Confidence Tree

- Minimax tree with UCB-based urgencies (Kocsis & Szepesvari, 2006)
- Leaf node: MC simulation, expand after *k* visits
- Online algorithm – can be stopped anytime and give meaningful result
  - Final move selection: node with highest #simulations
- Converges – given unlimited time, will find optimal solution

MCTS: Other Applications

- General planning tasks with large search space and stochastic evaluation function
- Other games (Poker, Amazons, Arima, ...)
- Robot online task planning
- Sailing "auto-navigator"
- Etc. etc.

Better Simulations



Basic Implementation

Trivial Heuristics

Local Patterns

Caveats!

Uniformly Random…

- In each move, pick a random element from 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 means deeper tree and more accurate values
- **Plausibility** – situations should be resolved like in real game

X

- **Balance** – all reasonable results should have chance to appear in playouts

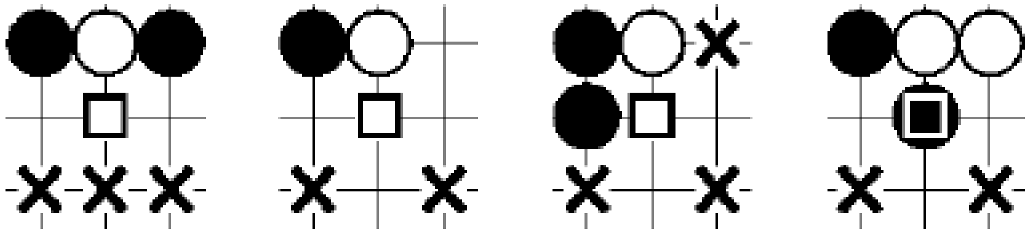
Simple Heuristics

- 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 candidate move (Gelly, 2006)
- Considered only around last move
- => Produces "nice" local sequences
- 3x3 patterns = 16bit numbers => Very fast

appendix 5



Balanced Patterns

- Stronger playout is not better playout!
  - Imbalance => consistently biased assessment of situation, UCT misbehave
- 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)
  - Significant improvement on 5x5 board, not researched yet on larger boards

Better Tree Search

Prior Node Values

All Moves As First

Rapid Action EValuation

Progressive Widening

Multithreaded Search

Fresh Nodes

- UCT: Play each node once first – too ineffective
- **First Play Urgency:** Initialize *urgency* with fixed value (~1.2), start UCB-selecting nodes
- **Priors:** Initialize *value* heuristically
  - => “Progressive unpruning/widening”
  - Playout policy hinting – capture, atari, 3x3 patterns, eye filling
  - Distance from board border
  - CFG distance from 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: shortest path in CFG
  - Useful for 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* in the rest of the game(!)
- Moves in-tree and in most of playout are considered (nakade or last 1/3 of playout cut)

Rapid Action Evaluation

- How to incorporate AMAF in node value? (Gelly & Silver, 2007)
- *value = β × amafval + (1-β) × uctval*

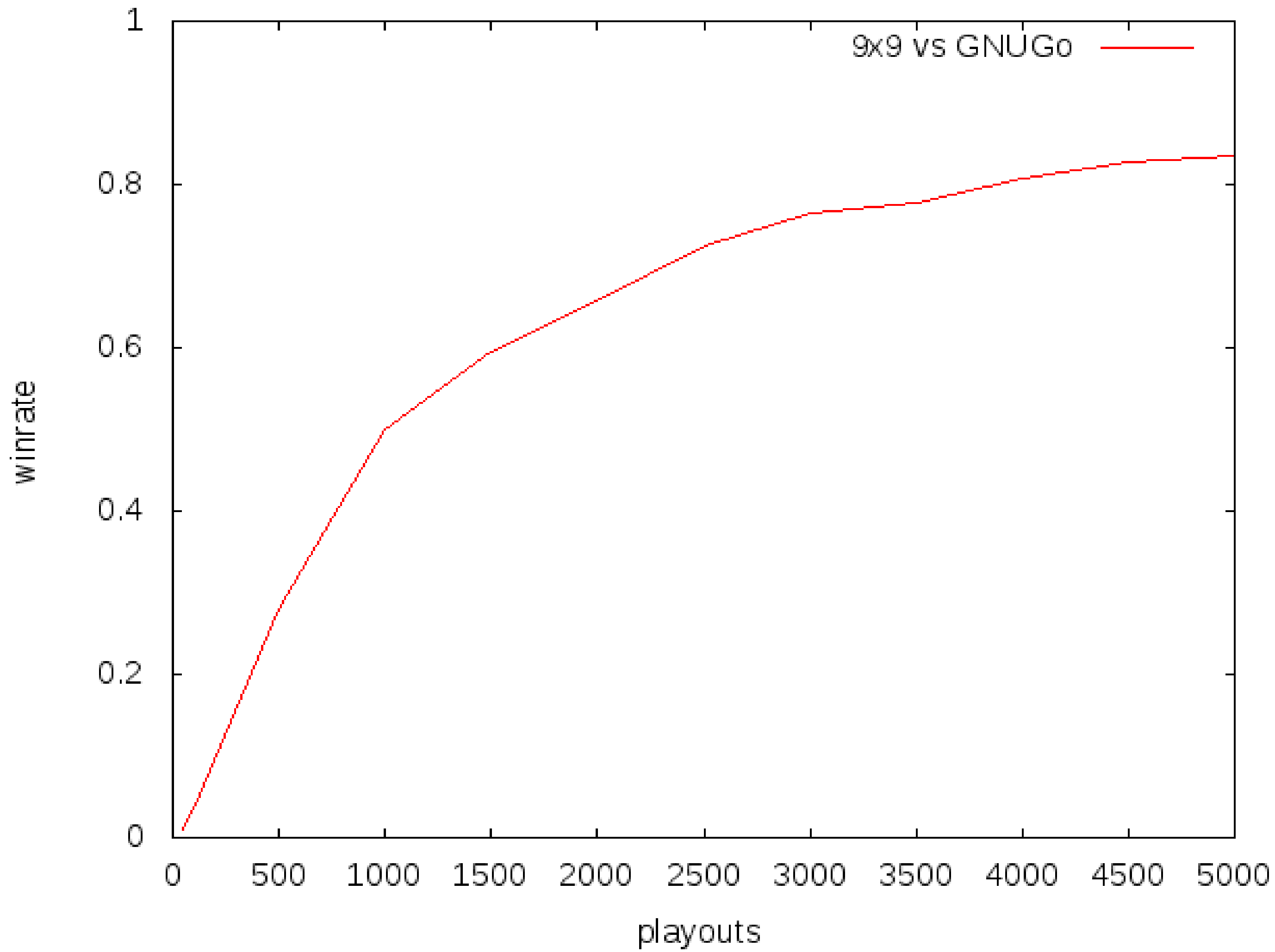
$$\beta = amafsims \times \left(amafsims + uctsims + \frac{amafsims \times uctsims}{r} \right)^{-1}$$

- With small *uctsims*, *β ~ 1*, but goes → *0*
- *r*: RAVE weight (“equivalence”) parameter, usually ~3000

RAVE Aftermath

- **Key result** in MCTS Go, making it stronger than 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 ($c=\sim 0.005$ on 9x9) – AMAF is sufficiently noisy

RAVE Performance



Criticality

- (Coulom, 2009) Focus on places that are "key" for both players – owning the point is important for winning the game
- Similar to AMAF, but:
  - Covariance of winrates for both players
  - Ownership of point, not play of stone

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

- Small improvement (49% $\rightarrow$ 54%)

Parallel MCTS

(Chaslot, 2008)

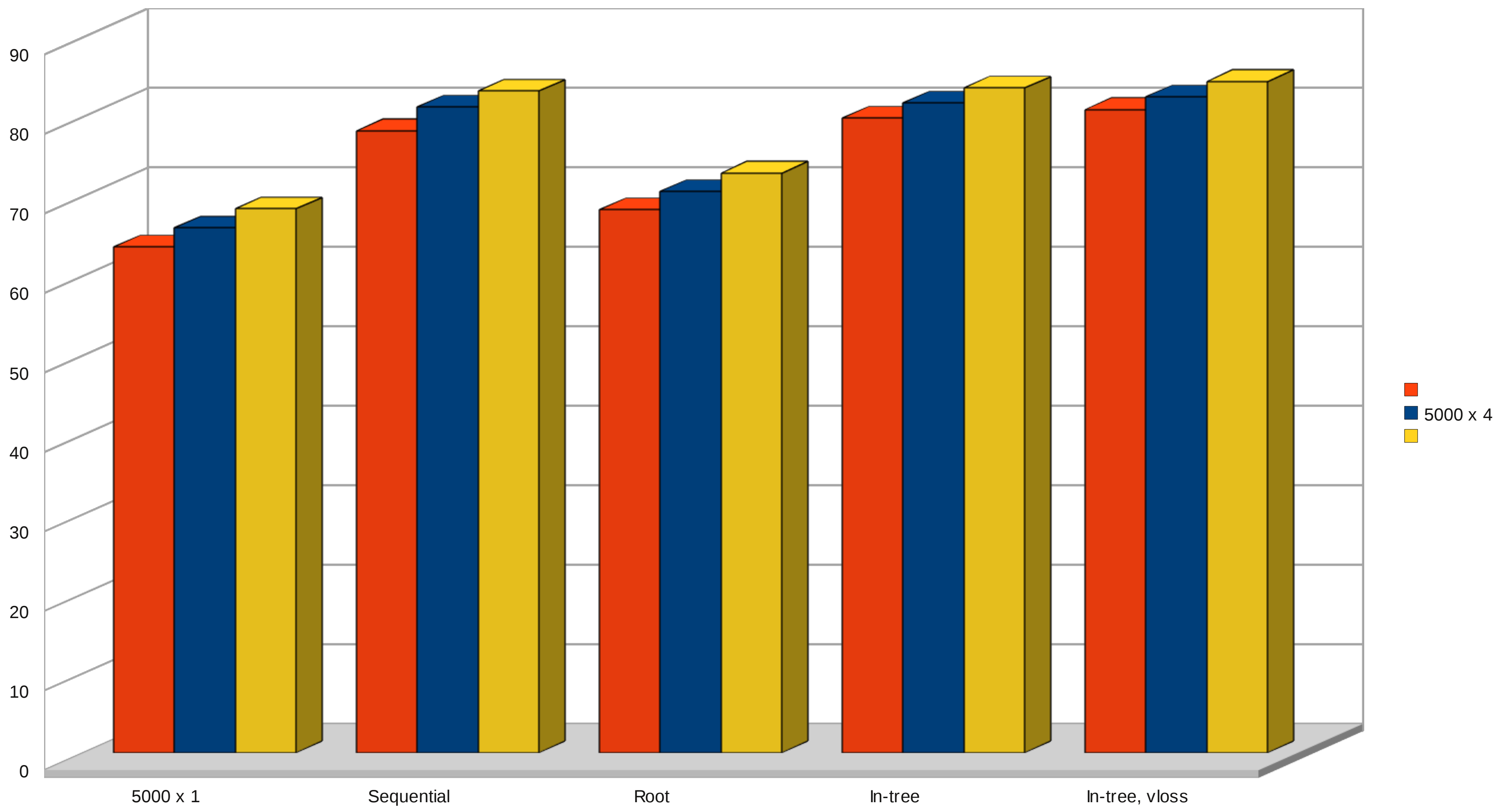
- **Root-level** – independent search in each thread, merge at the end
  - Threads “vote” on best move
  - Slight-to-medium improvement, does not seem to scale much
- **Leaf-level** – single thread searches, all threads play in parallel
  - More accurate node value
  - Small improvement, large overhead

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)

Parallel Performance

(9x9 vs GNUGo)



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!

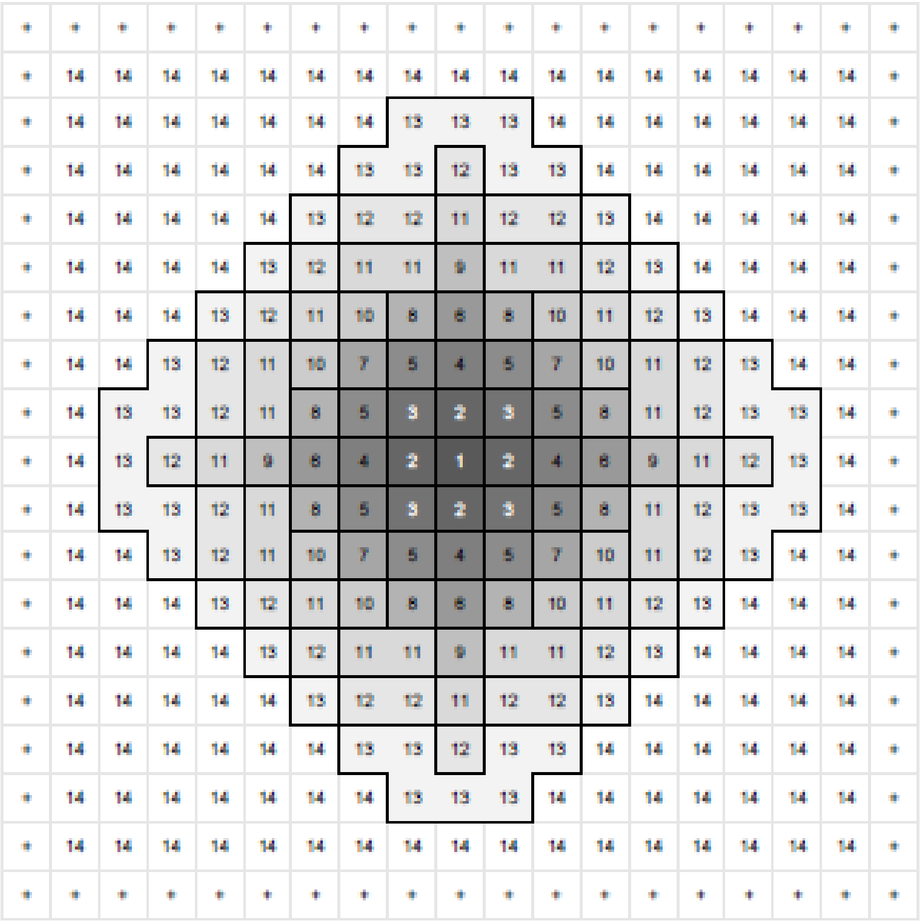
Shape Patterns

- 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:

  $d(x,y) = |dx| + |dy| + \max(|dx|,|dy|)$
- 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, pattern is matched
- Shape – as just described
- Capture, atari, selfatari, liberty counts, ko... (van der Werf, 2002)
- Distance to last, next-to-last move
    - CFG distance or circular distance
- MonteCarlo owner – portion of simulations where I am point owner at the game end
- Each feature can have its zobrist hash

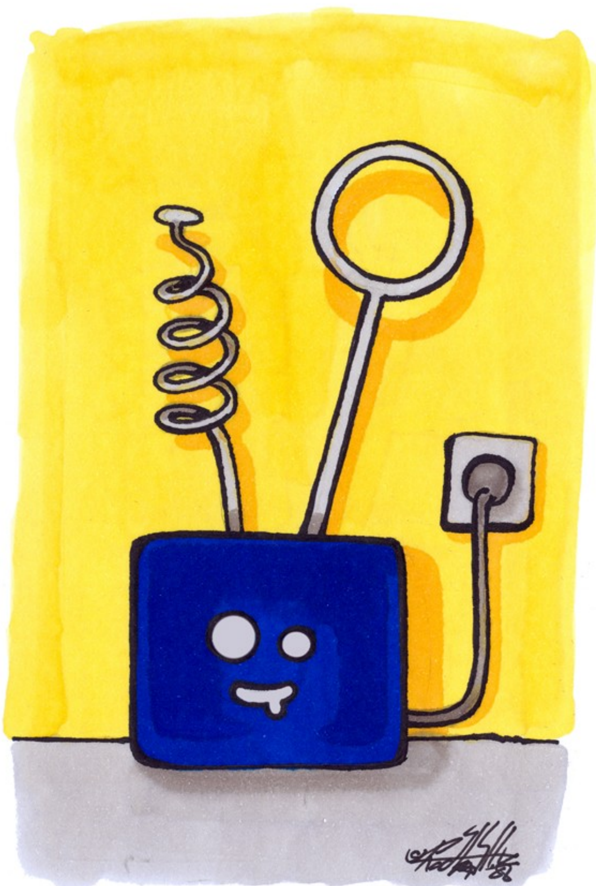
ELO Ratings

- ELO: Putting competitive strength of many individuals on a single scale (Hunter, 2004)
- 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 features strength
- Produces probability distribution over moves
- Use to choose 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?

Opensource UCT:

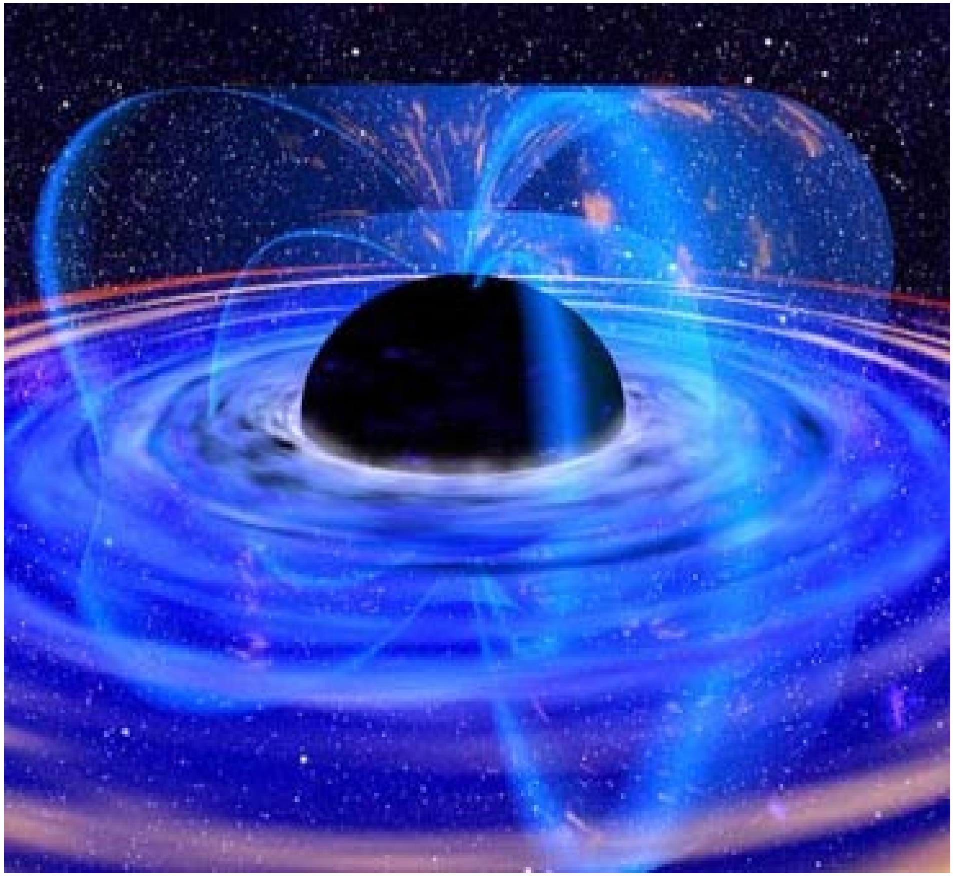
- Fuego – complex, general
- **Pachi** – simple, Go focus

Pachi

- Densely-commented C code, about 5k 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)
- Root-level or in-tree parallelism (modular)
- **Autotest** – generic UNIX framework for testing of stochastic engines performance

Unsolved Problems

- Handling extreme situations
- Narrow sequences
- HPC implementation
- Aesthetically pleasing play
- Abstract understanding of the board



Playing in Extreme Situations

- **Extreme situation:** The computer has either huge advantage or 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 outlook is extreme

Black in Handicap

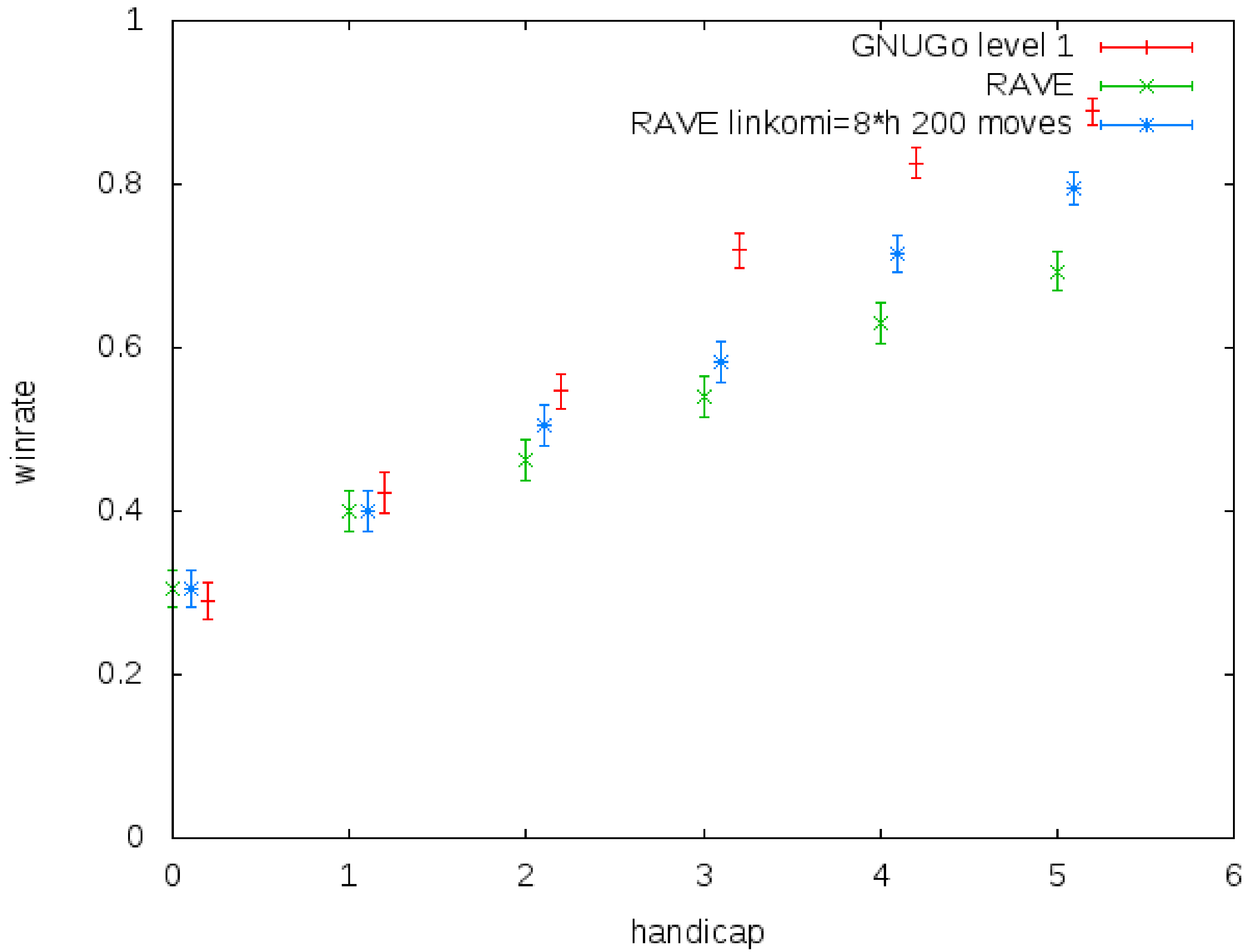
- Linear dynamic komi, online dynamic komi, artificial passes
- **Dynamic komi:** Before counting final position in the simulation, subtract certain amount of points from black score
- Online komi: Adjust komi to keep probabilities between ~[0.5,0.6]; universal (not only handicap games), not well researched

Linear Dynamic Komi

- **Linear DK:** Calculate komi value K based on 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

Handicap Performance

(19x19 vs GNUGo level 10)



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, but mysterious behavior in some cases
- 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
- Thomas Wolf is trying to apply results from study of dynamic systems

Thank you!

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

http://senseis.xmp.net/

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

Tue 18:00 Mustek
Wed 20:00 Koleje Troja
Thu 19:00 Dejvice