# Old-fashioned Computer Go vs Monte-Carlo Go

*Bruno Bouzy (Paris 5 University)*

*CIG07 Tutorial*

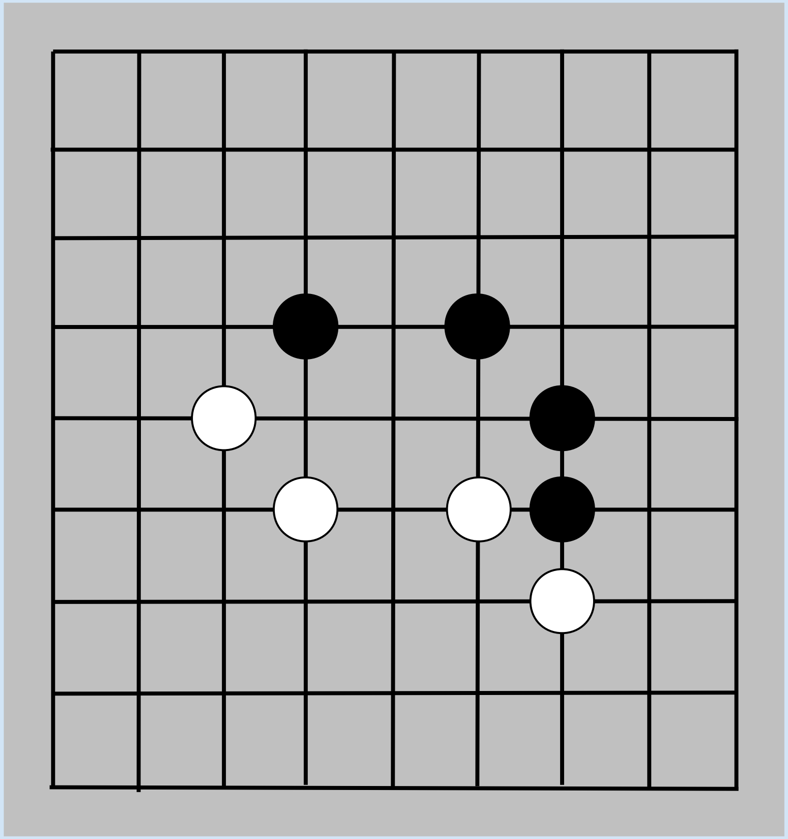
*April 1 2007*

# Outline

- Computer Go (CG) overview
  - Rules of the game
  - History and main obstacles
  - Best programs and competitions
- Classical approach: divide and conquer
  - Conceptual evaluation function
  - Global move generation
  - Combinatorial-game based
- New approach: Monte-Carlo Tree Search
  - Simple approach: depth-1 Monte-Carlo
  - MCTS, UCT
  - Results on 9x9 boards
- Enhancement assessment
  - 9x9 boards
  - Scaling up to 13x13 or 19x19 boards
  - Parallelisation
- Future of Computer Go

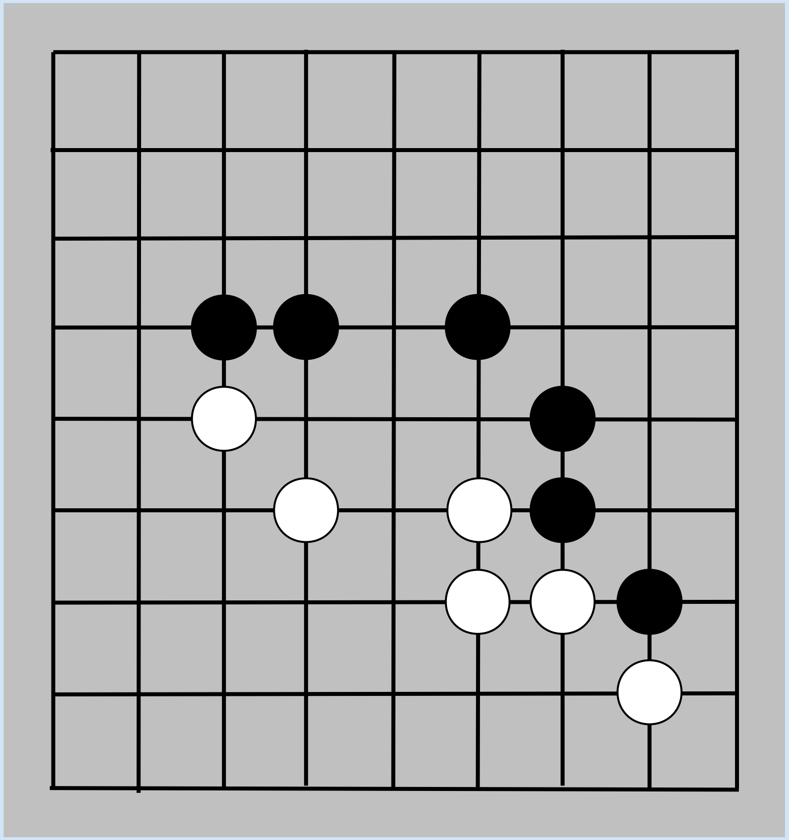
# Rules overview through a game (opening 1)

- Black and White move alternately by putting one stone on an intersection of the board.



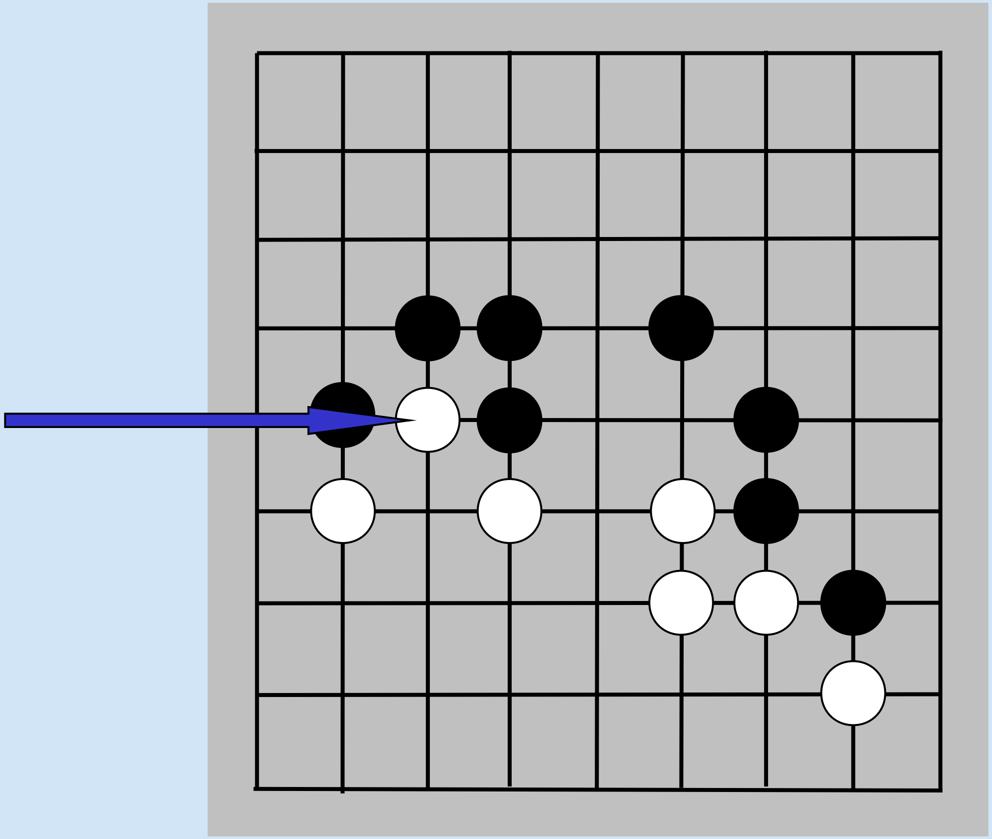
# Rules overview through a game (opening 2)

- Black and White aims at surrounding large « zones »



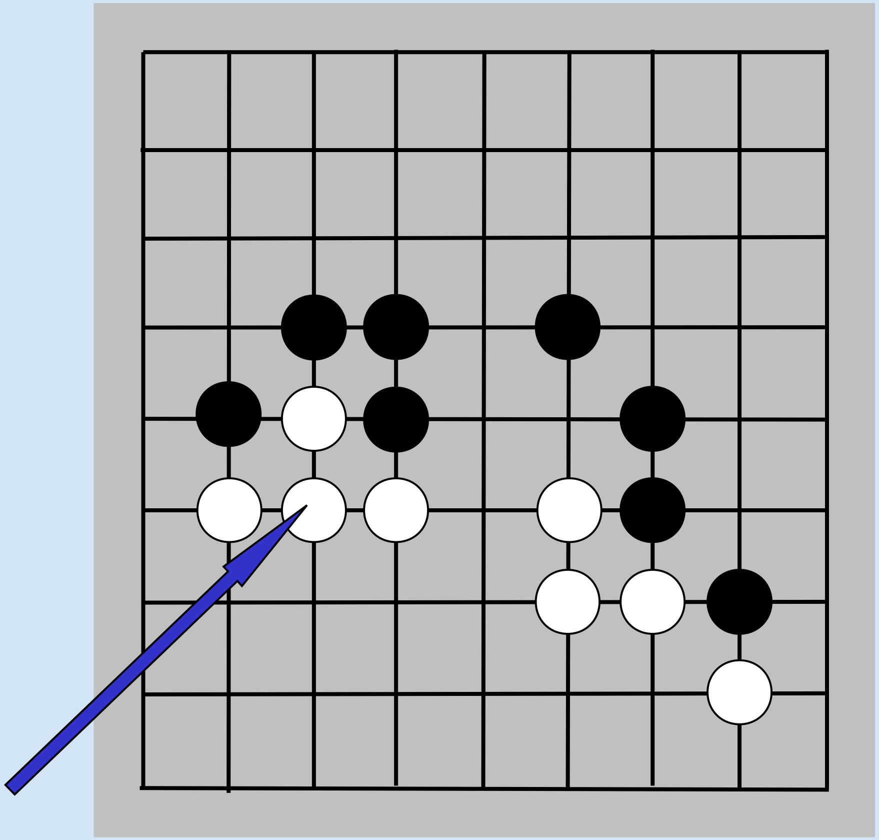
# Rules overview through a game (atari 1)

- A white stone is put into « atari » : it has only one liberty (empty intersection) left.



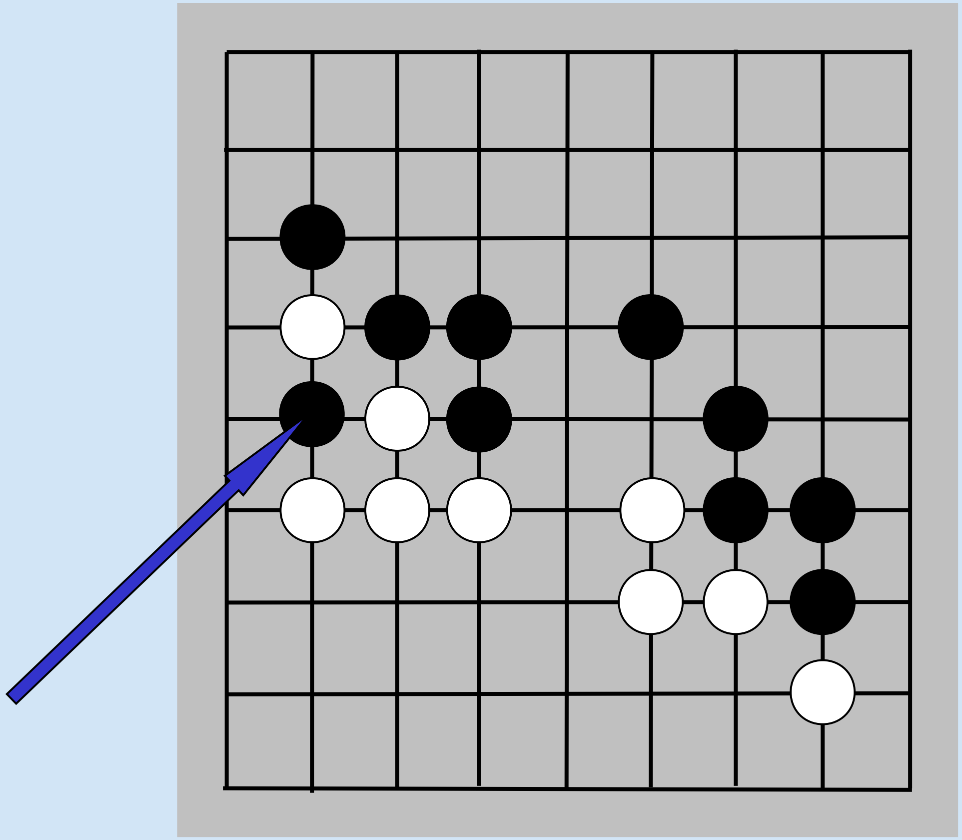
# Rules overview through a game (defense)

- White plays to connect the one-liberty stone yielding a four-stone white string with 5 liberties.



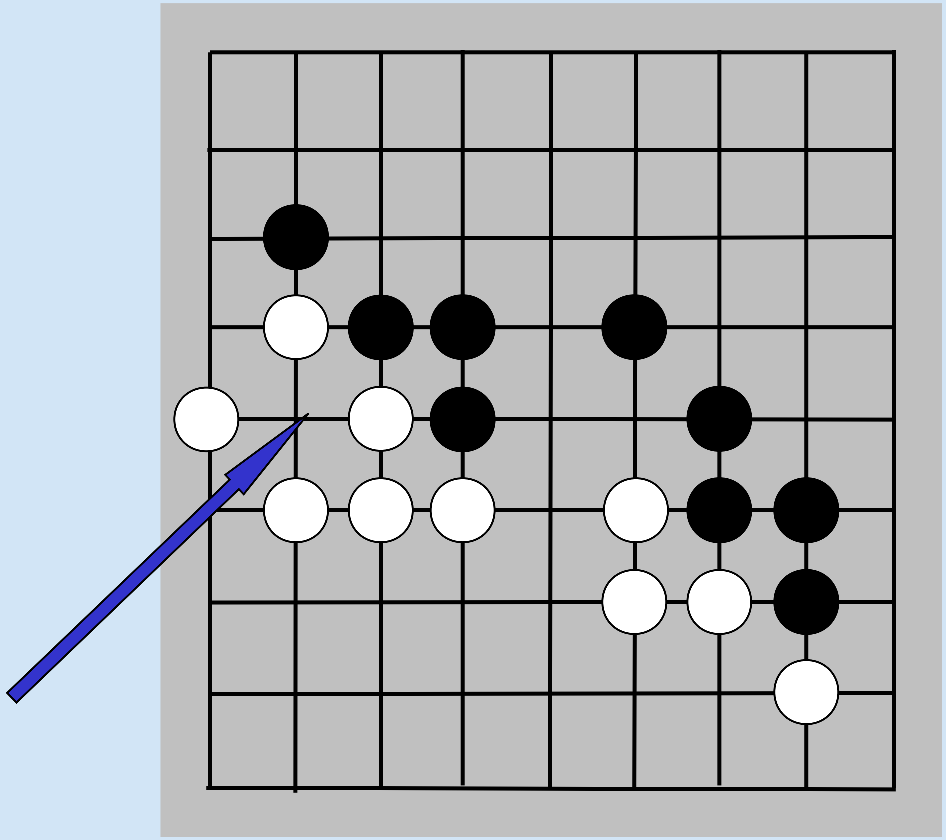
# Rules overview through a game (atari 2)

- It is White’s turn. One black stone is atari.



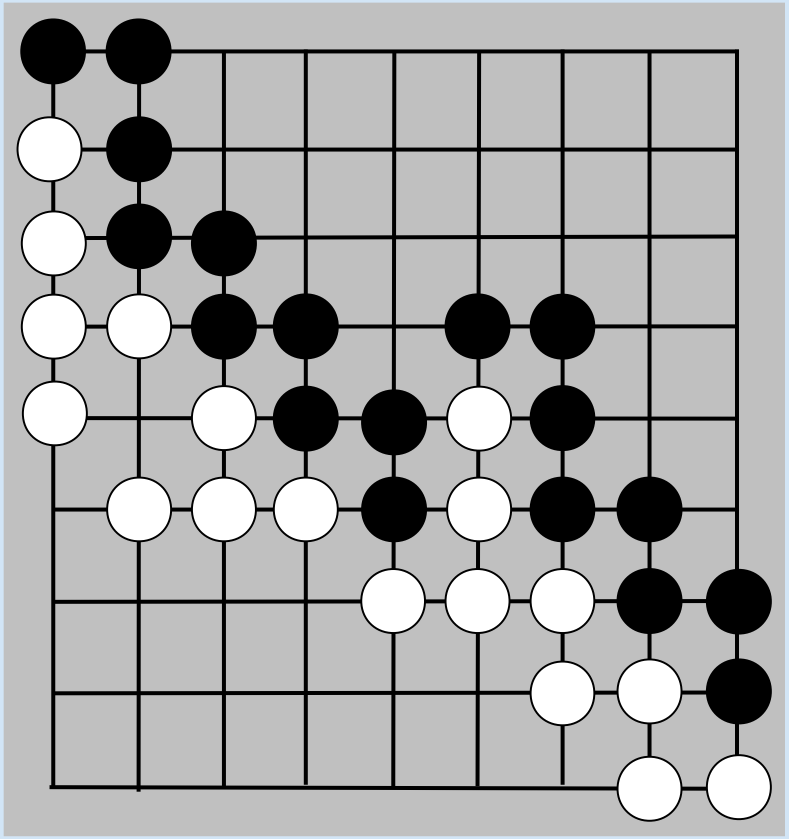
# Rules overview through a game (capture 1)

- White plays on the last liberty of the black stone which is removed



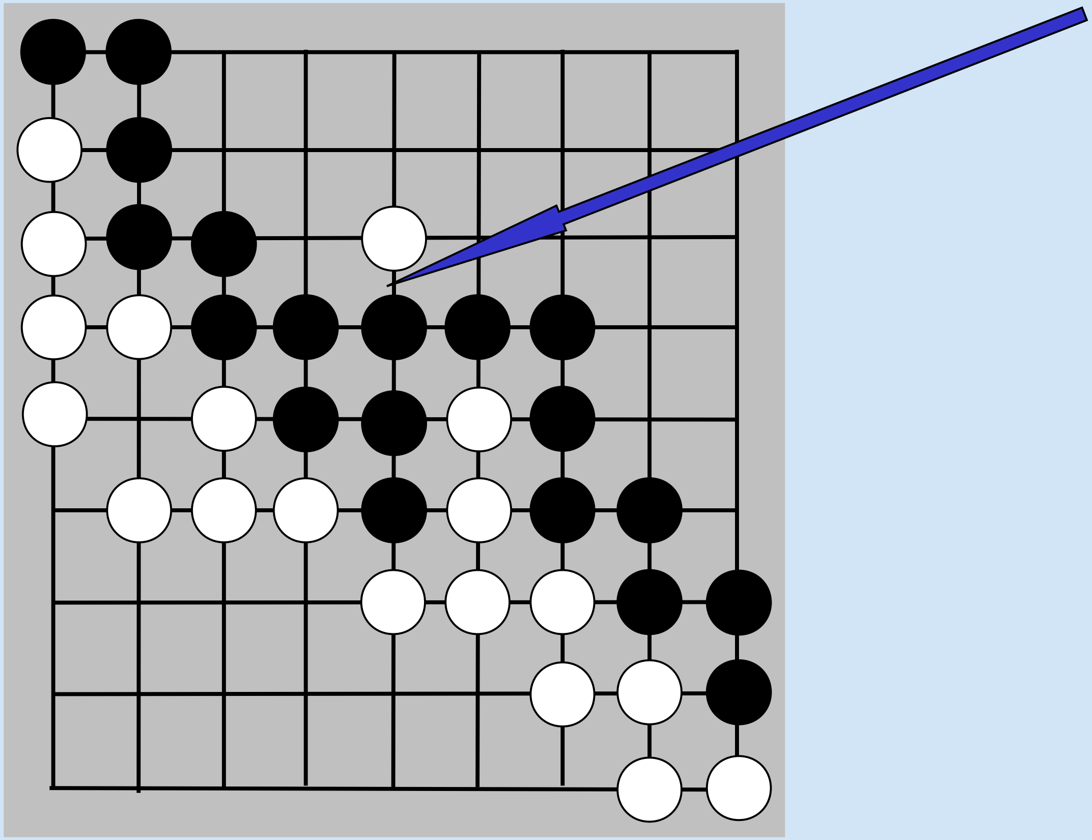
# Rules overview through a game (human end of game)

- The game ends when the two players pass.
- In such position, experienced players pass.



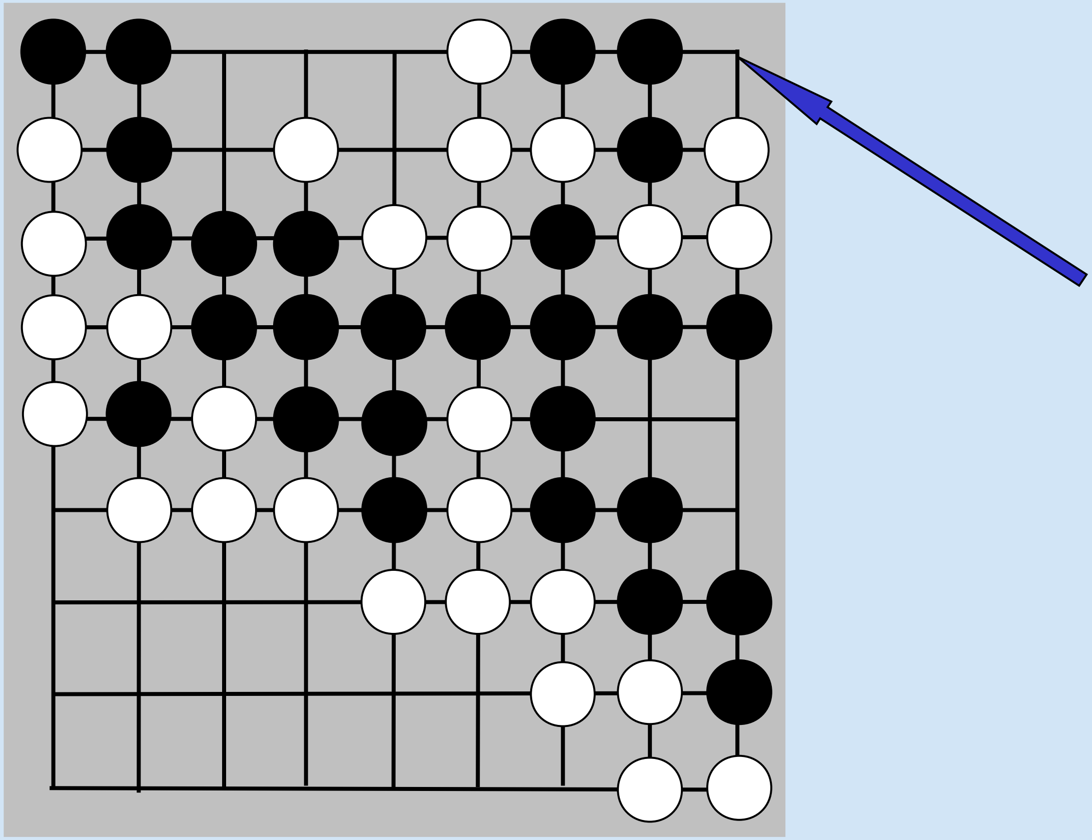
# Rules overview through a game (contestation 1)

- White contests the black « territory » by playing inside.
- Black answers, aiming at capturing the invading stone.



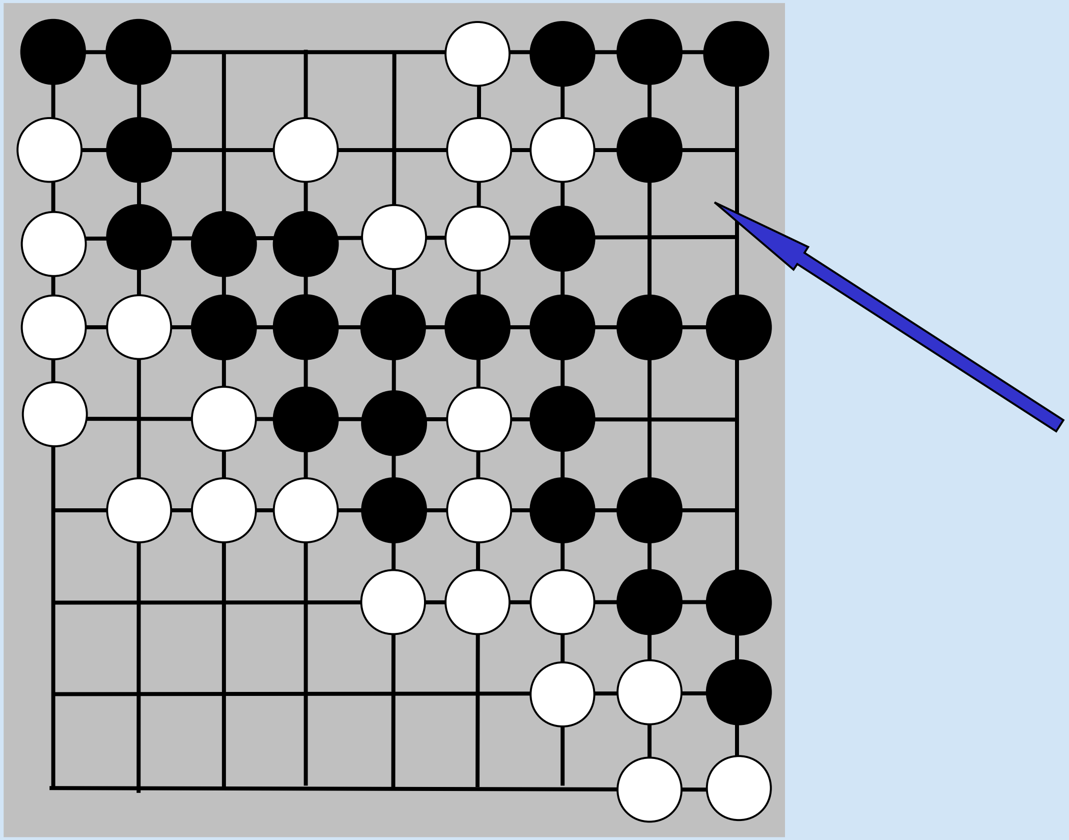
# Rules overview through a game (contestation 2)

- White contests black territory, but the 3-stone white string has one liberty left



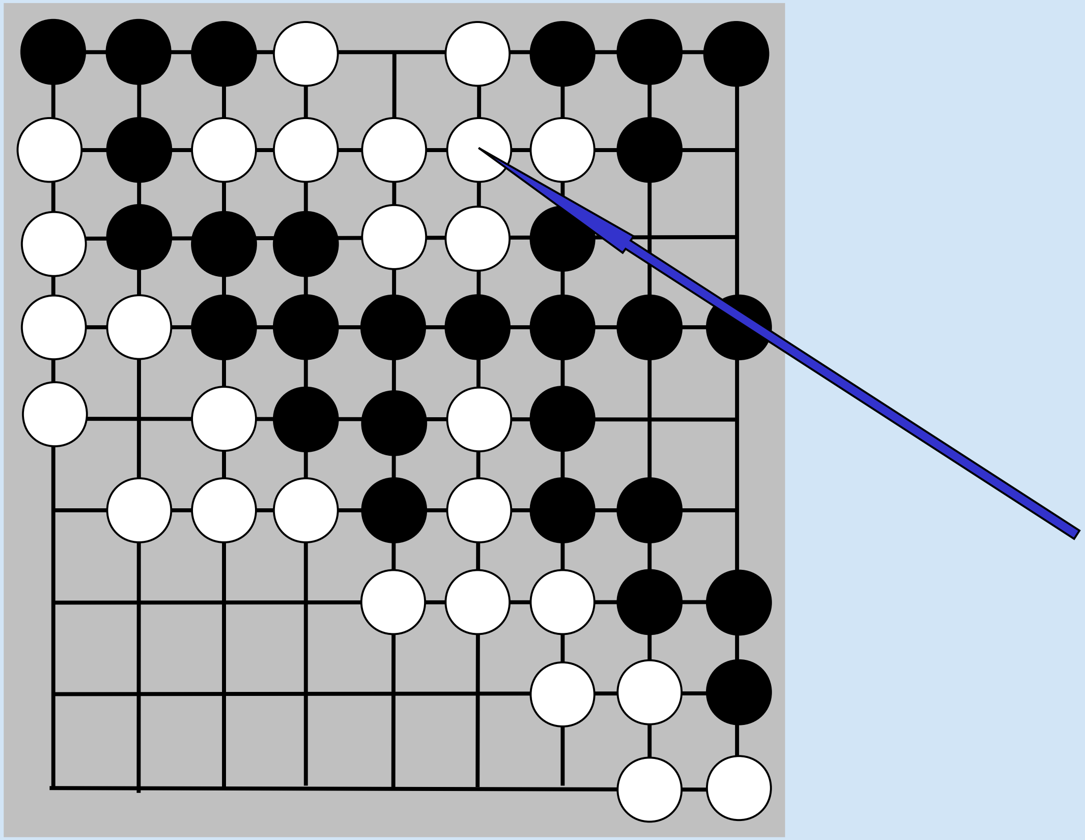
# Rules overview through a game (follow up 1)

- Black has captured the 3-stone white string



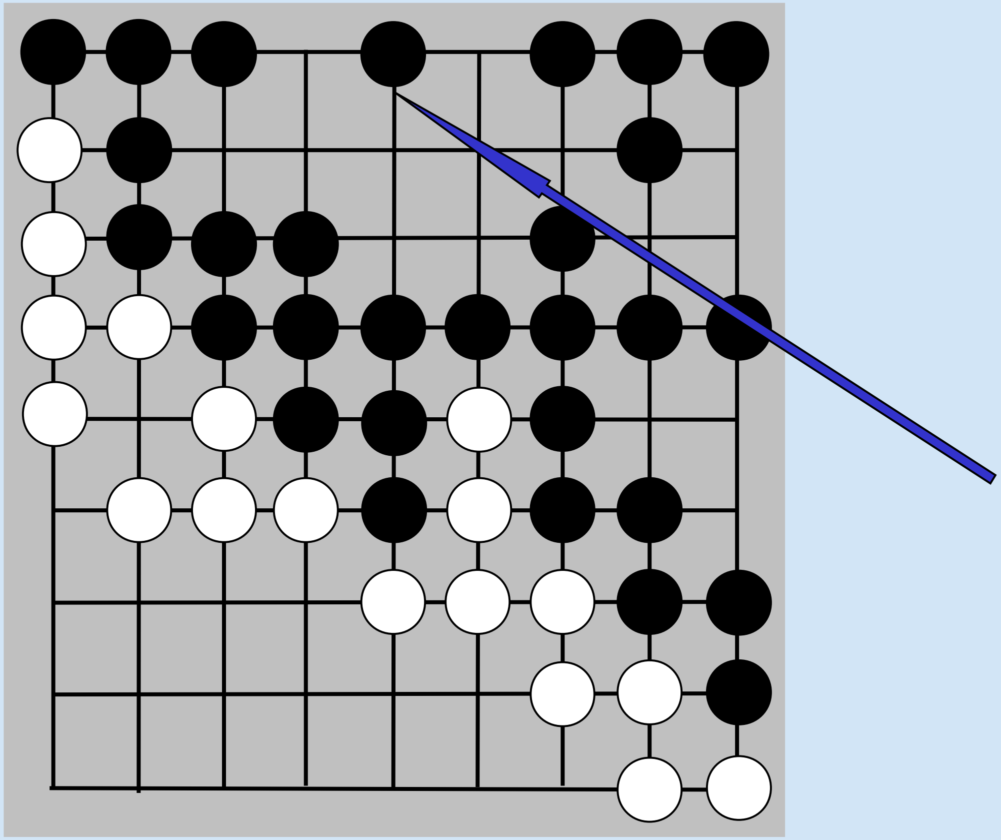
# Rules overview through a game (follow up 2)

- White lacks liberties…



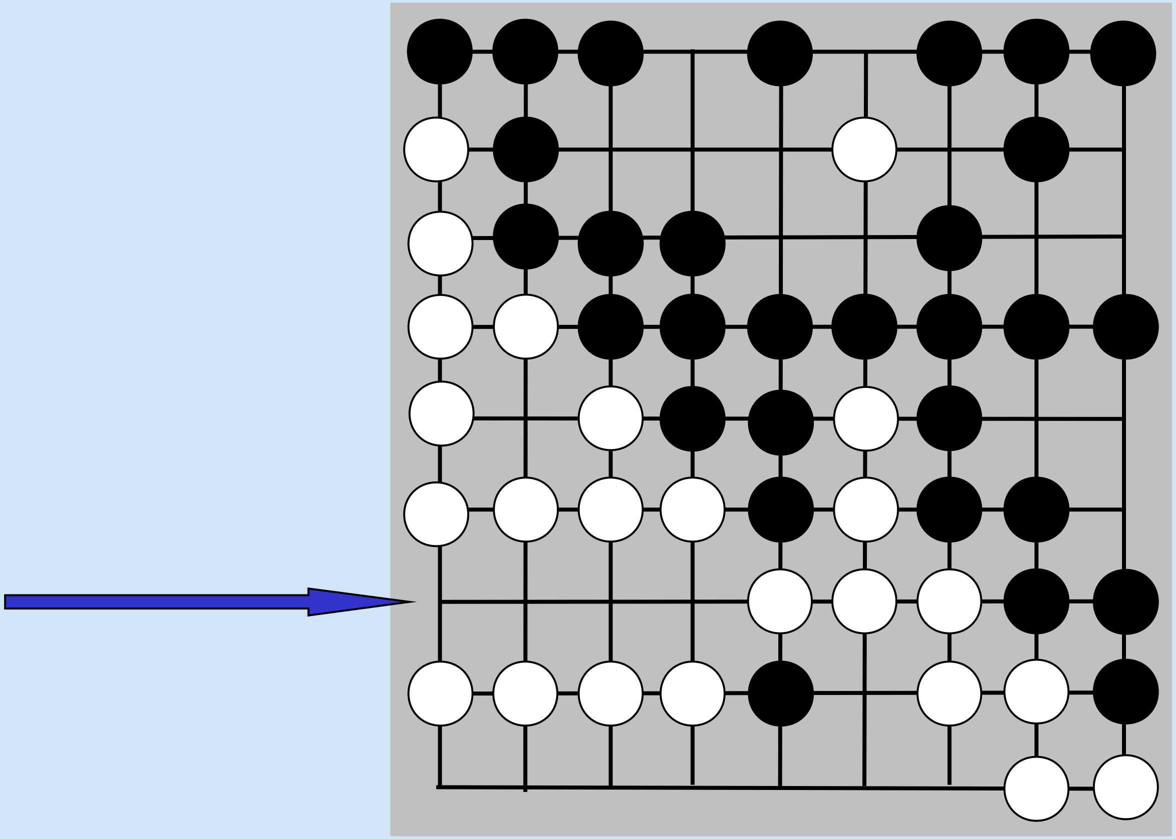
# Rules overview through a game (follow up 3)

- Black suppresses the last liberty of the 9-stone string
- Consequently, the white string is removed



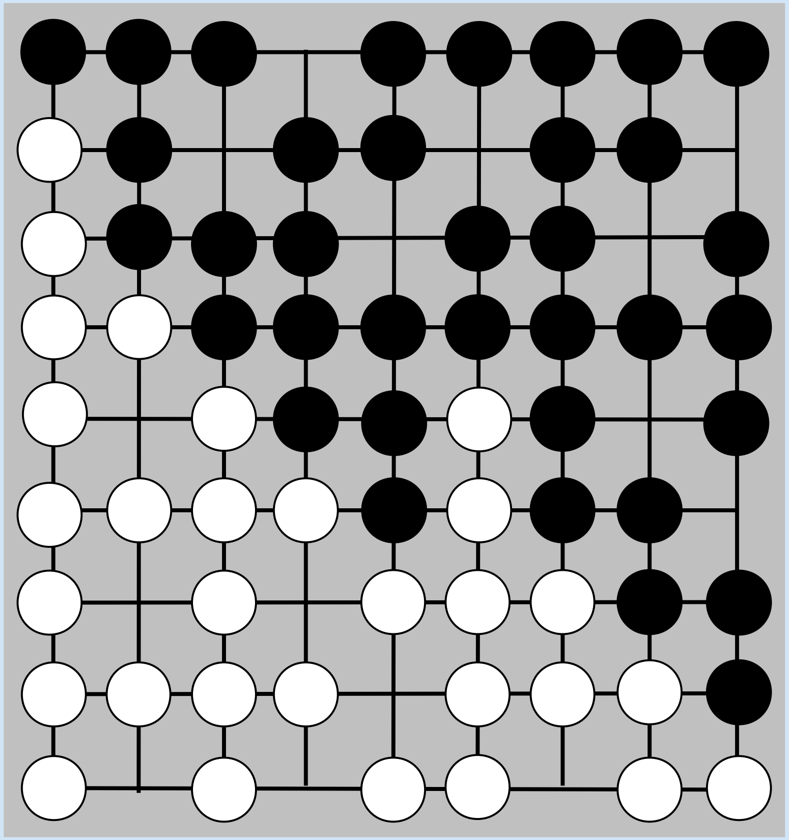
# Rules overview through a game (follow up 4)

- Contestation is going on on both sides. White has captured four black stones



# Rules overview through a game (concrete end of game)

- The board is covered with either stones or « eyes »
- The two players pass



# History (1/2)

- First go program (Lefkovitz 1960)
- First machine learning work (Remus 1963)
- Zobrist hashing (Zobrist 1969)
- First two computer go PhD thesis
  - Potential function (Zobrist 1970)
  - Heuristic analysis of Go trees (Ryder 1970)
- First-program architectures: influence-function based
- Small boards (Thorpe & Walden 1964)
- Interim2 program (Wilcox 1979)
- G2 program (Fotland 1986)
- Life and death (Benson 1988)
- Pattern-based program: Goliath (Boon 1990)

# History (2/2)

- Combinatorial Game Theory (CGT)
  - ONAG (Conway 1976),
  - Winning ways (Conway & al 1982)
  - Mathematical Go (Berlekamp 1991)
  - Go as a sum of local games (Muller 1995)
- Machine learning
  - Automatic acquisition of tactical rules (Cazenave 1996)
  - Neural network-based evaluation function (Enzenberger 1996)
- Cognitive modelling
  - (Bouzy 1995)
  - (Yoshikawa & al 1997)

# Main obstacles (1/2)

- CG witnesses AI improvements
  - 1994: Chinook beat Marion Tinsley (Checkers)
  - 1997: Deep Blue beat Kasparov (Echecs)
  - 1998: Logistello >> best human (Othello)
  - (Schaeffer, van den Herik 2002)

- Combinatorial complexity
  - B: branching factor,
  - L: game length,
  - $B^L$ estimation :
  - Go ($10^{400}$) > Echecs($10^{123}$) > Othello($10^{58}$) > Checkers($10^{32}$)

# Main obstacles (2/2)

- 2 main obstacles :
  - Global tree search *impossible*
  - Non terminal position evaluation *hard* 
- Medium level (10th kyu) 😐
- Huge effort since 1990 :
  - Evaluation function,
  - Break down the position into sub-positions (Conway, Berlekamp),
  - Local tree searches,
  - pattern-matching, knowledge bases.

# Kinds of programs

- Commercial programs
  - Haruka, Many Faces, Goemate, Go4++, KCC Igo,
  - Hidden descriptions. 😐
- Free Programs
  - GNU Go, available sources. ☺
- Academic programs
  - Go Intellect, GoLois, Explorer, Indigo, Magog,
  - CrazyStone, MoGo, NeuroGo,
  - Scientific descriptions ☺.
- Other programs...

# Indigo

- Indigo
  - www.math-info.univ-paris5.fr/~bouzy/INDIGO.html
- International competitions since 2003:
  - Computer Olympiads:
    - 2003: 9x9: 4/10, 19x19: 5/11
    - 2004: 9x9: 4/9, 19x19: 3/5 (bronze) ☺
    - 2005: 9x9: 3/9 (bronze) ☺, 19x19: 4/7
    - 2006: 19x19: 3/6 (bronze) ☺
  - Kiseido Go Server (KGS):
    - « open » and « formal » tournaments.
  - Gifu Challenge:
    - 2006: 19x19: 3/17 ☺
  - CGOS 9x9

# Competitions

- Ing Cup (1987-2001)
- FOST Cup(1995-1999)
- Gifu Challenge (2001-)
- Computer Olympiads (1990;2000-)
- Monthly KGS tournaments (2005-)
- Computer Go ladder (Pettersen 1994-)
- Yearly continental tournaments
  - American
  - European
- CGOS (Computer Go Operating System 9x9)

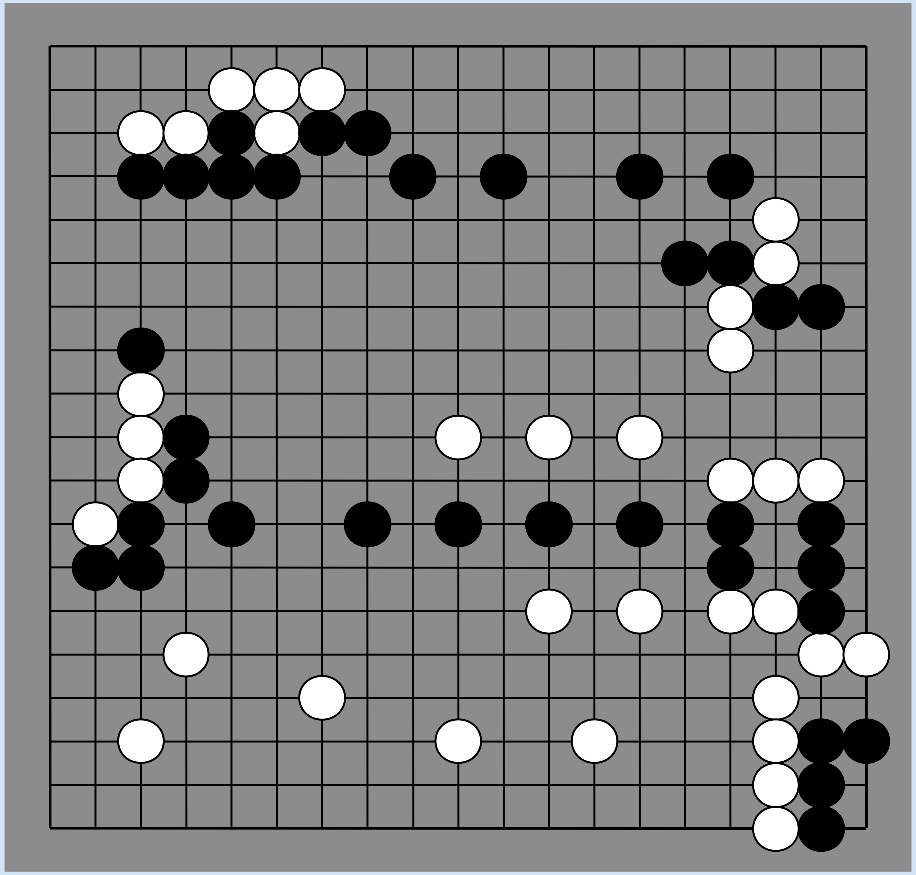
# Best 19x19 programs

- Go++
  - Ing, Gifu, FOST, Gifu, Olympiads
- Handtalk (=Goemate)
  - Ing, FOST, Olympiads
- KCC Igo
  - FOST, Gifu
- Haruka
  - ?
- Many Faces of Go
  - Ing
- Go Intellect
  - Ing, Olympiads
- GNU Go
  - Olympiads

# Divide-and-conquer approach (start)

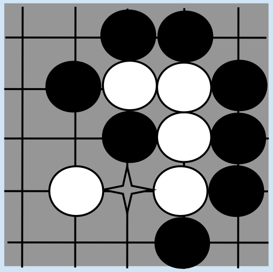
- Break-down
  - Whole game (win/loss; score)
  - Goal-oriented sub-games  String capture (shicho)
    - Connections, Dividers, Eyes, Life and Death
- Local searches
  - Alfa-beta and enhancements
  - PN-search, Abstract Proof Search, lambda-search
- Local results
  - Combinatorial-Game-Theory-based
    - Main feature:
      - If Black plays first, if White plays first
    - (>, <, *, 0, {a|b}, …)
- Global Move choice
  - Depth-0 global search:
    - Temperature-based: *, {a|b}
  - Shallow global search

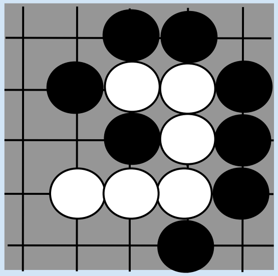
# A Go position

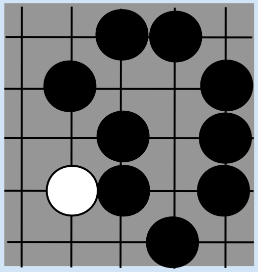


# Basic concepts, local searches, and combinatorial games (1/2)

- Block capture
  - || 0
  - First player wins

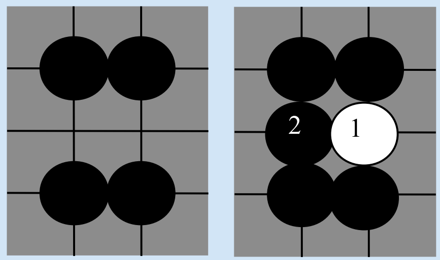




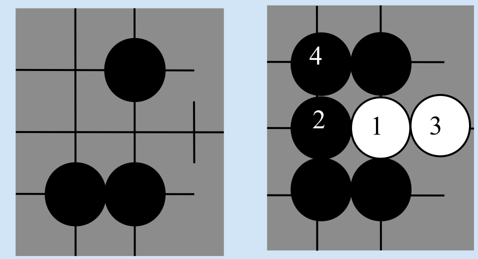


# Basic concepts, local searches, and combinatorial games (2/2)

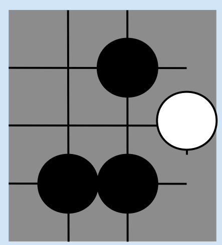
- Connections:
  - >0

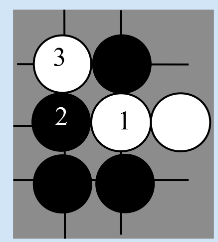


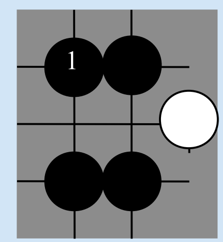

  - >0



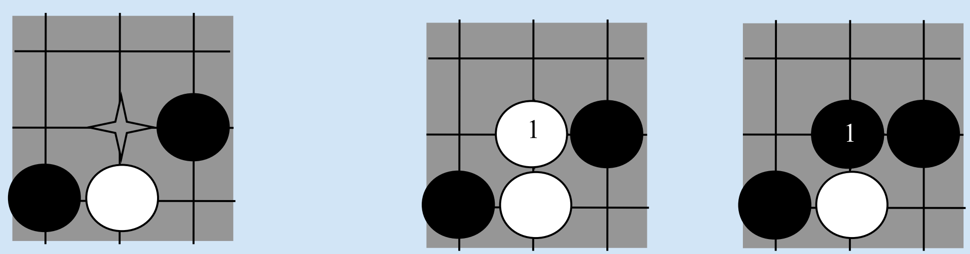

  - || 0





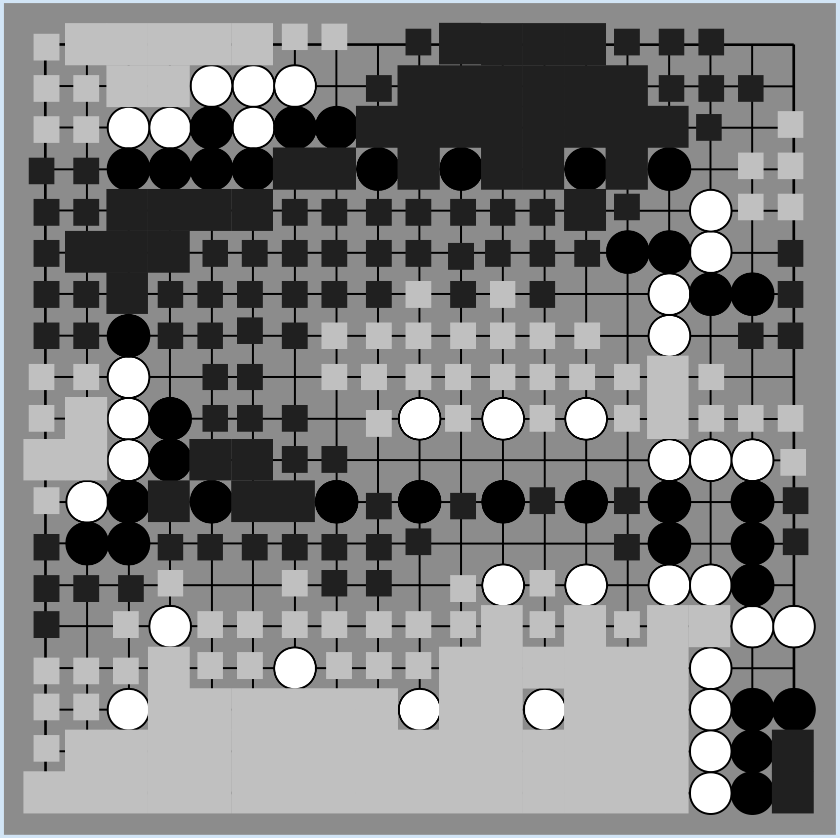




- Dividers:
  - || 0



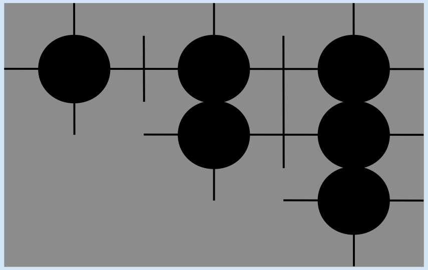
# Influence function

- Based on dilation (and erosion)



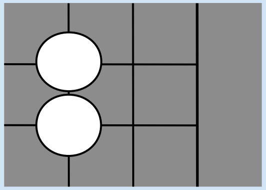
# Group building

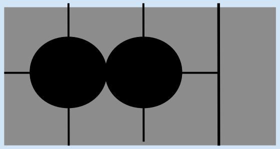
- Initialisation:
  - Group = block
- Influence function:
  - Group = connected compound
- Process:
  - Groups are merged with connector >
- Result:



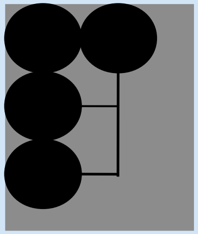
# Group status

- Instable groups:





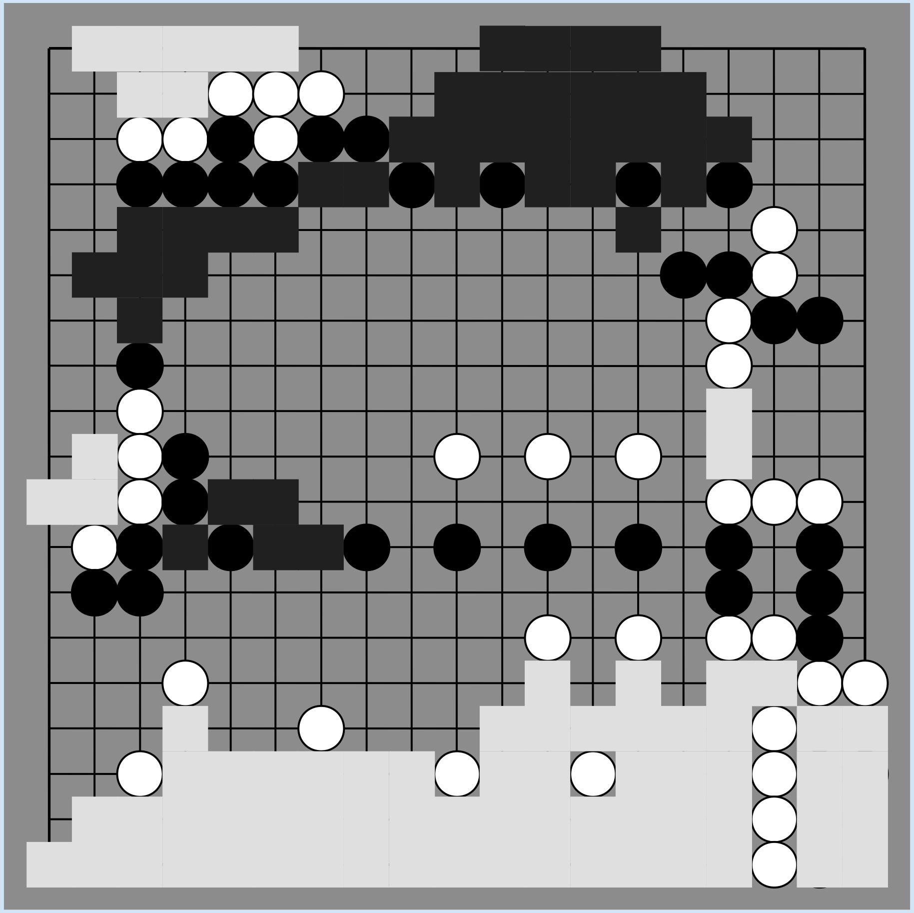
- Dead group:



# Conceptual Evaluation Function pseudo-code

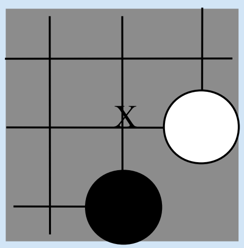
- While dead groups are being detected,
  - perform the inversion and aggregation processes
- Return the sum of
  - the “value” of each intersection of the board
  - (+1 for Black, and –1 for White)

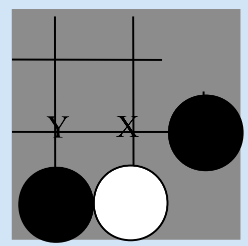
# A Go position conceptual evaluation

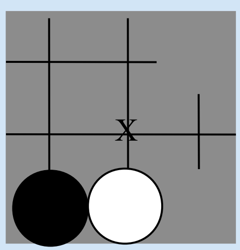


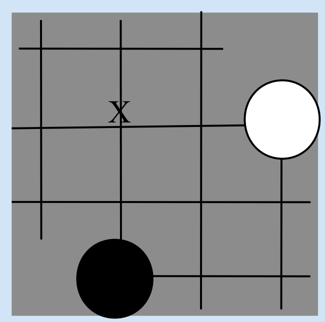
# Local move generation

- Depend on the abstraction level
- Pattern-based

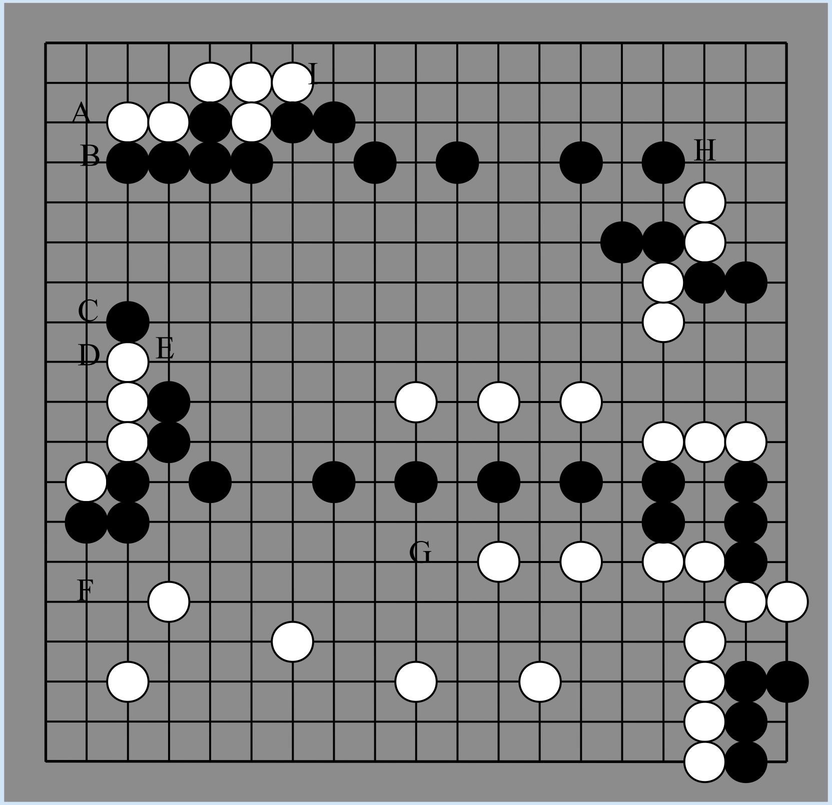


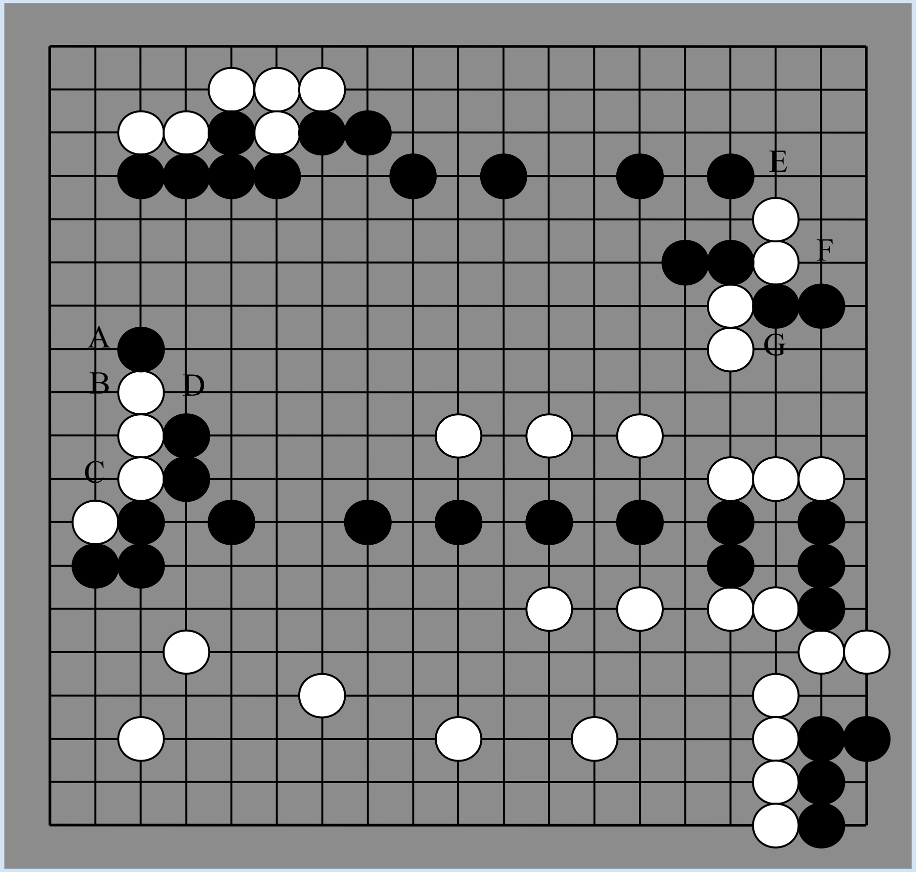



# « Quiet » global move generation



# « Fight-oriented » global move generation



# Divide and conquer approach (end)

- Upsides
  - Feasible on current computers
  - Local search « precision »
  - Local result accuracy based on anticipation
  - Fast execution
- Downsides
  - The breakdown-stage is not proved to be correct
  - Based on domain-dependent knowledge
  - The sub-games are not independent
  - Heuristic-based move choice
  - Two-goal-oriented moves are hardly considered
  - Data structure updating complexity

# Move choice

- Two strategies using the divide and conquer approach
  - Depth-0 strategy, global move evaluation
    - Local tree searches result based
    - Domain-dependent knowledge
    - No conceptual evaluation
    - GNU Go, Explorer
  - Shallow global tree search using a conceptual evaluation function
    - Many Faces of Go, Go Intellect,
    - Indigo2002.

# Monte Carlo and Computer games (start)

- Games containing chance:
  - Backgammon (Tesauro 1989-),
- Games with hidden information:
  - Bridge (Ginsberg 2001),
  - Poker (Billings & al. 2002),
  - Scrabble (Sheppard 2002).

# Monte Carlo and complete information games

- (Abramson 1990) general model of terminal node evaluation based on simulations
  - Applied to 6x6 Othello
- (Brügmann 1993) simulated annealing
  - Two move sequences (one used by Black, one used by White)
  - « all-moves-as-first » heuristic
  - Gobble

# Monte-Carlo and Go

- Past history
  - (Brugmann 1993),
  - (Bouzy & Helmstetter 2003) ,
  - Min-max and MC Go (Bouzy 2004),
  - Knowledge and MC Go (Bouzy 2005),
  - UCT (Kocsis & Szepesvari 2006),
  - UCT-like (Coulom 2006),

- Quantitative assessment:
  - $\sigma$ (9x9) ~= 35
  - 1 point precision: N ~= 1,000 (68%), 4,000 (95%)
  - 5,000 up to 10,000 9x9 games / second (2 GHz)
  - few MC evaluations / second

# Monte Carlo and Computer Games (basic)

- Evaluation:
  - Launch N random games
  - Evaluation = mean of terminal position evaluations
- Depth-one greedy algorithm:
  - For each move,
    - Launch N random games starting with this move
    - Evaluation = mean of terminal position evaluations
  - Play the move with the best mean
- Complexity:
  - Monte Carlo: O(NBL)
  - Tree search: $O(B^L)$

# Monte-Carlo and Computer Games (strategies)

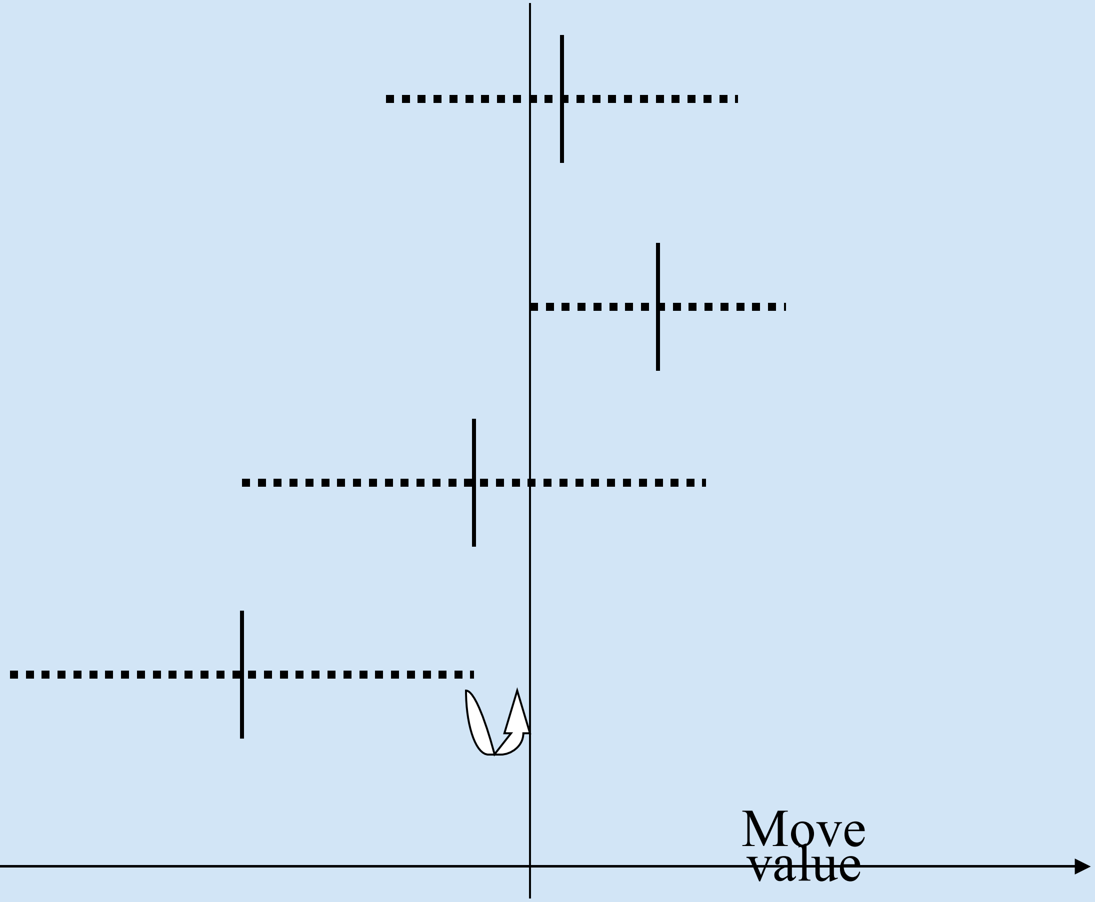
- Greedy algorithm improvement: confidence interval update
  - $[m - R\sigma/N^{1/2}, m + R\sigma/N^{1/2}]$
  - R: parameter.

- Progressive pruning strategy :
  - First move choice: randomly,
  - Prune move inferior to the best move,
  - (Billings al 2002, Sheppard 2002, Bouzy & Helmstetter ACG10 2003)

- Upper bound strategy:
  - First move choice : argmax $(m + R\sigma/N^{1/2})$,
  - No pruning
  - IntEstim (Kaelbling 1993), UCB (Auer & al 2002)

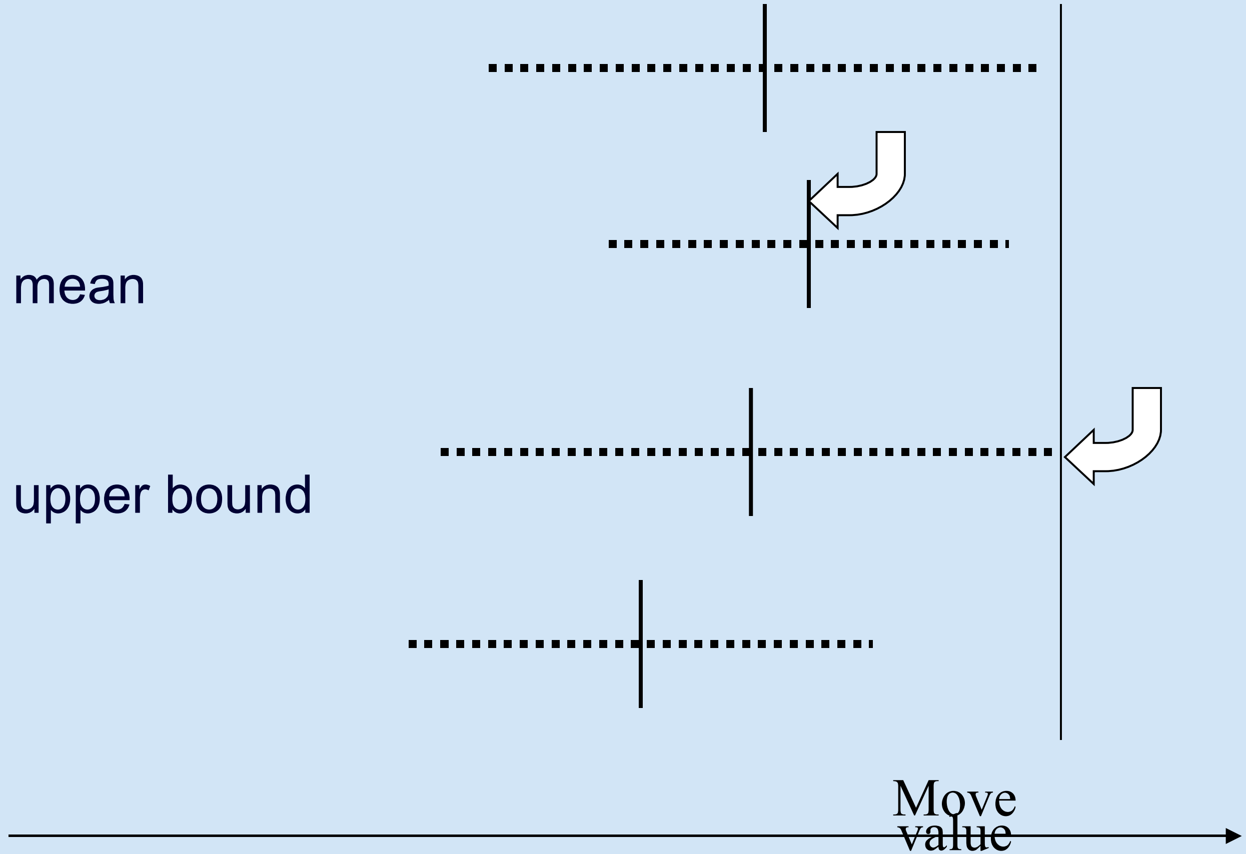
# Progressive Pruning strategy

- Are there unpromising moves ?
- Move 1
- Move 2
  - Current best
- Move 3
- Move 4
  - Can be pruned



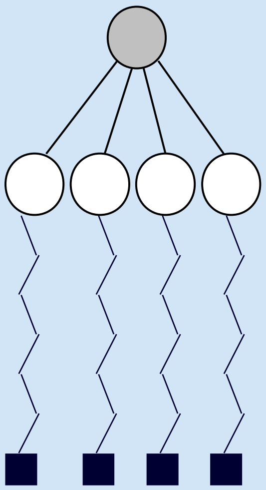
# Upper bound strategy

- Which move to select ?
- Move 1
- Move 2
  - Current best mean
- Move 3
  - Current best upper bound
- Move 4



# Monte-Carlo and Computer Games (pruning strategy)

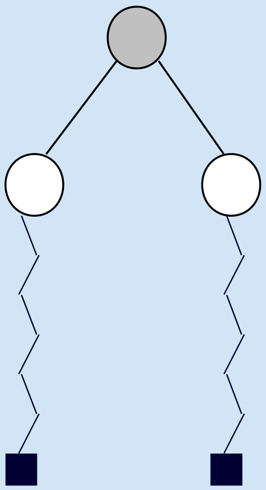
- Example



- The root is expanded
- Random games are launched on child nodes

# Monte-Carlo and Computer Games (pruning strategy)

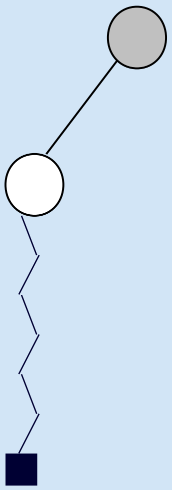
- Example



- After several games, some child nodes are pruned

# Monte-Carlo and Computer Games (pruning strategy)

- Example



- After other random games, one move is left…
- And the algorithm stops.

# Monte-Carlo and “complex” games (4)

- “Complex” games:
  - Go, Amazones, Clobber
- Results:
  - Move quality increases with computer power ☺
  - Robust evaluation ☺
  - Global (statistical) search ☺
- Way of playing:
  - Good global  sense ☺,
  - local tactical weakness --
- Easy to program ☺
  - Rules of the games only,
  - No break down of the position into sub-positions,
  - No conceptual evaluation function.

# Multi-Armed Bandit Problem (1/2)

- (Berry & Fristedt 1985, Sutton & Barto 1998, Auer & al 2002)
- A player plays the Multi-armed bandit problem
  - He selects a arm to push
  - Stochastic reward depending on the selected arm
  - For each arm, the reward distribution is unknown
  - Goal: maximize the cumulated reward over time
  - Exploitation vs exploration dilemma
- Main algorithms
  - $\varepsilon$-greedy, Softmax,
  - IntEstim (Kaelbling 1993)
  - UCB (Auer & al 2002)
  - POKER (Vermorel 2005)

# Multi-Armed Bandit Problem (2/2)

- Monte-Carlo games & MAB similarities
  - Action choice
  - Stochastic reward (0 1 or numerical)
  - Goal: choose the best action

- Monte-Carlo games & MAB: two main differences

  - **Online or offline** reward ?
    - MAB: cumulated online reward
    - MCG: offline
      - Online rewards counts nothing
      - Reward provided later by the game outcome

  - MCG: **Superposition** of MAB problems
    - 1 MAB problem = 1 tree node

# Monte-Carlo Tree Search (MCTS) (start)

- Goal: appropriate integration of MC and TS
  - TS: alfa-beta like algorithm, best-first algorithm
  - MC: uncertainty management
- UCT: UCB for Trees (Kocsis & Szepesvari 2006)
  - Spirit: superpositions of UCB (Auer & al 2002)
  - Downside: Tree growing left unspecified
- MCTS framework
  - Move selection (Chaslot & al) (Coulom 2006)
  - Backpropagation (Chaslot & al) (Coulom 2006)
  - Expansion (Chaslot & al) (Coulom 2006)
  - Simulation (Bouzy 2005) (Wang & Gelly 2007)

# Move Selection

- UCB (Auer & al 2002)
  - Move eval = mean + C * sqrt(log(t)/s)
    **= Upper Confidence interval Bound**
- OMC (Chaslot & al 2006)
  - Move eval = probability to be better than best move
- PPBM (Coulom 2006)
  - Move eval = probability to be the best move

# Backpropagation

- Node evaluation:
  - "Average" back-up = average over simulations going through this node
  - "Min-Max" back-up = Max (resp Min) evaluations over child nodes
  - "Robust max" = Max number of simulations going through this node

- Good properties of MCTS:
  - With "average" back-up, the root evaluation converges to the "min-max" evaluation when the number of simulations goes to infinity
  - "Average" back-up is used at every node
  - "Robust max" can be used at the end of the process to complete properly

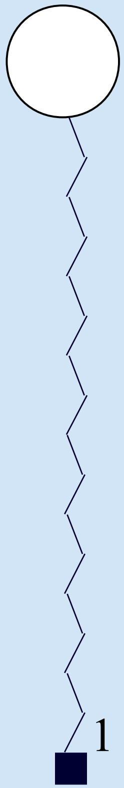
# Node expansion and management

- Strategy
  - Everytimes
  - One node per simulation
  - Few nodes per simulation according to domain dependent probabilities
- Use of a Transposition Table (TT)
  - When hash collision: link the nodes in a list
    - (different from TT in usual fixed depth alpha-beta tree search)

# Monte-Carlo Tree Search (end)

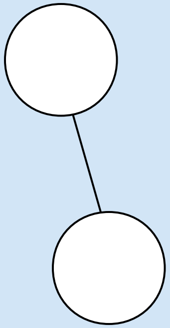
- MCTS():
  - While time,
    - PlayOutTreeBasedGame (list)
    - outcome = PlayOutRandomGame()
    - Update nodes (list, outcome)
  - Play the move with the best mean
- PlayOutTreeBasedGame (list)
  - node = getNode(position)
  - While node do
    - Add node to list.
    - M = Select move (node)
    - Play move (M)
    - node = getNode(position)
  - node = new Node()
  - Add node to list.

# Upper Confidence for Trees (UCT) (1)



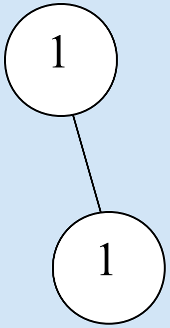

- A first random game is launched, and its value is backed-up

# Upper Confidence for Trees (UCT) (2)



- A first child node is created.

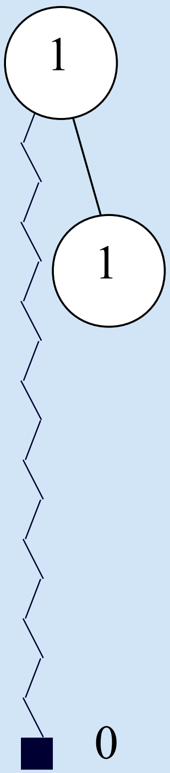
# Upper Confidence for Trees (UCT) (3)



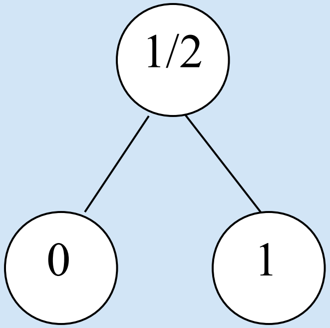

- The outcome of the random game is backed up.

# Upper Confidence for Trees (UCT) (4)

- At the root, unexplored moves still exist.
- A second game is launched, starting with an unexplored move.

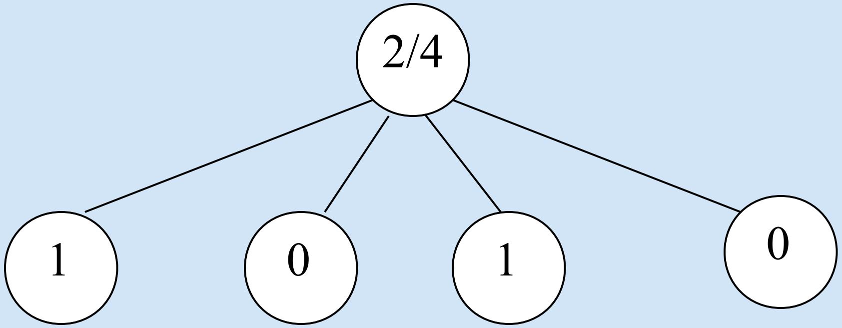


# Upper Confidence for Trees (UCT) (5)



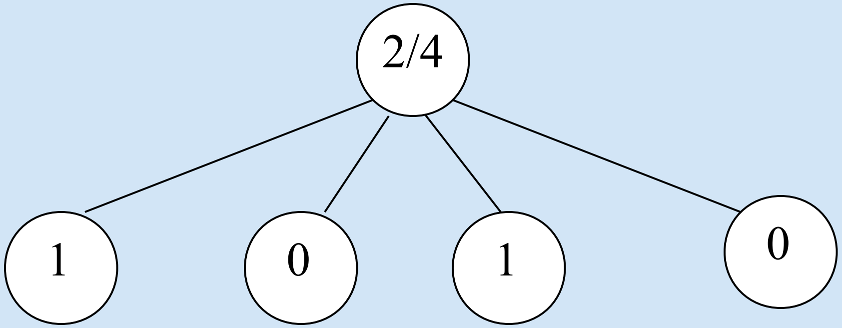

- A second node is created and the outcome is backed-up to compute means.

# Upper Confidence for Trees (UCT) (6)



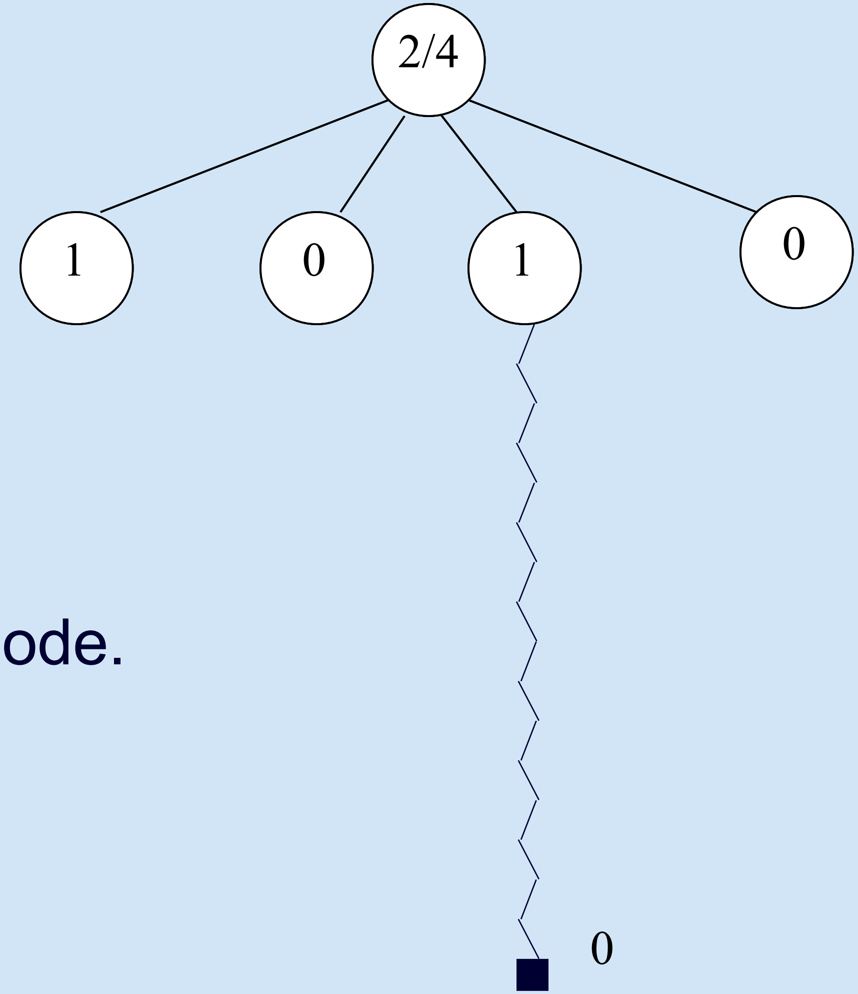

- All legal moves are explored, the corresponding nodes are created, and their means computed.

# Upper Confidence for Trees (UCT) (7)



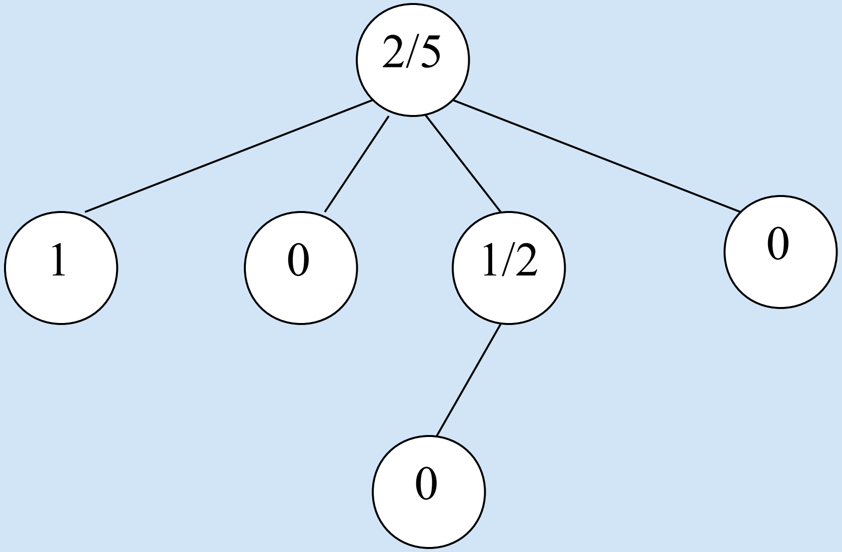

- For the next iteration, a node is greedily selected with the UCT move selection rule:
  - Move eval = mean + C * sqrt(log(t)/s)
- (In the continuation of this example, for a simplicity reason, let us consider C=0).

# Upper Confidence for Trees (UCT) (8)



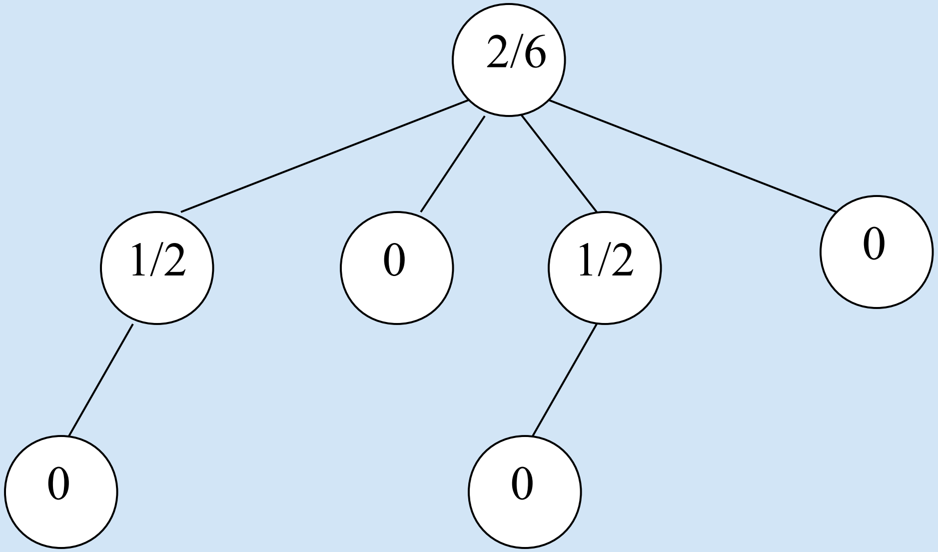

- A random game starts from this node.

# Upper Confidence for Trees (UCT) (9)



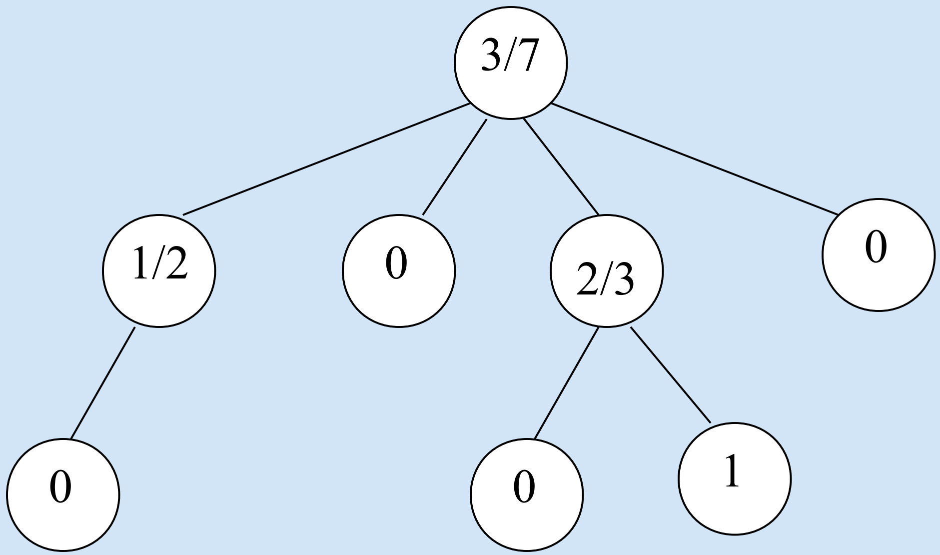

- A node is created.

# Upper Confidence for Trees (UCT) (9)



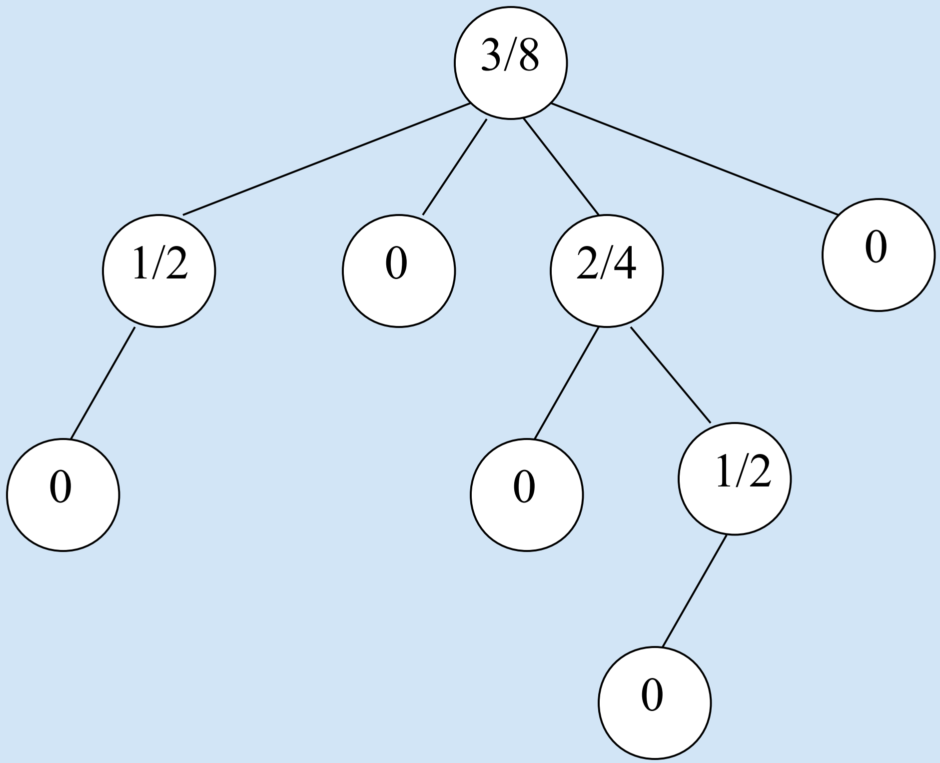

- The process repeats…

# Upper Confidence for Trees (UCT) (10)



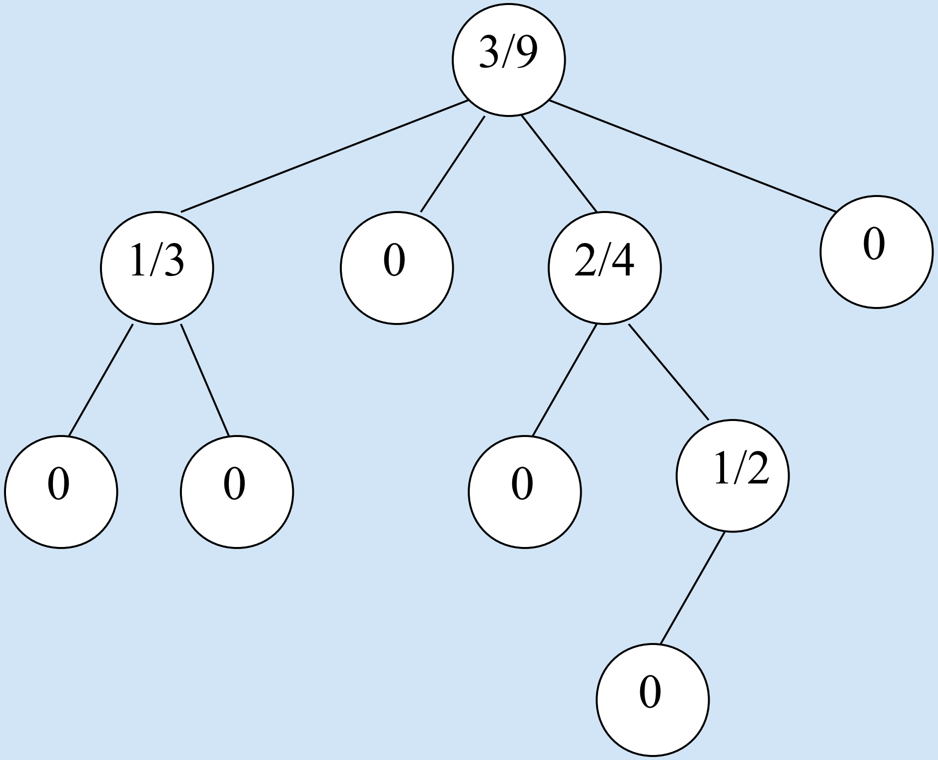

- … several times …

# Upper Confidence for Trees (UCT) (11)



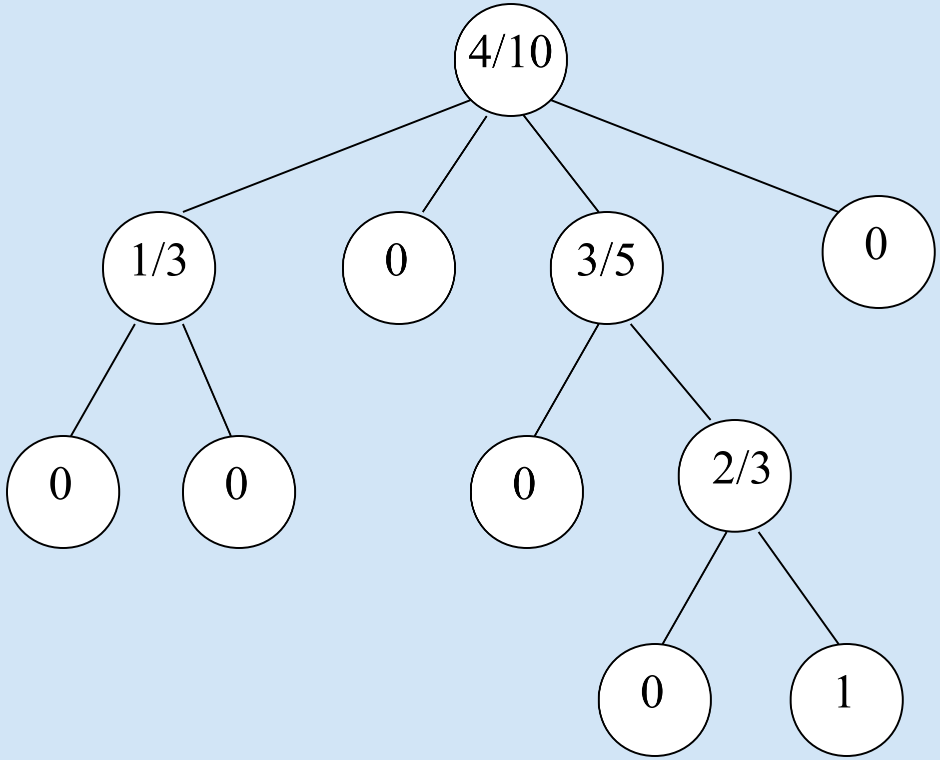

- … several times …

# Upper Confidence for Trees (UCT) (12)




- … in a best first manner …

# Upper Confidence for Trees (UCT) (13)



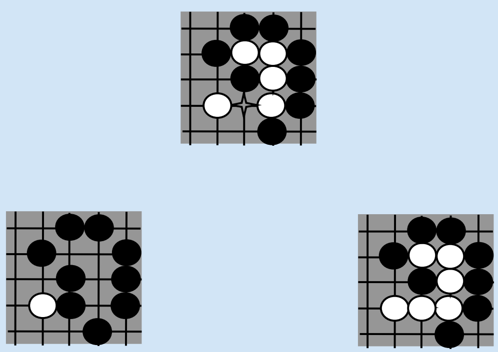

- … until timeout.

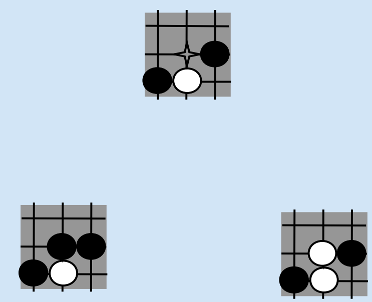
# Remark

- Moves cannot stay unvisited
  - Move eval = mean + C * sqrt(log(t)/s)
  - t is the number of simulations of the parent node
  - s is the number of simulations of the node
  - Move eval increases while move stays unvisited.

# MCGo and knowledge (1)

- Pseudo-random games:
  - Instead of being generated with a uniform probability,
  - Moves are generated with a probability *depending on specific domain-dependent knowledge*
    - Liberties of string in « atari »: Patterns 3x3:





  - Pseudo-random games look like go,
  - Computed means are more significant than before ☺

# MCGo and knowledge (2)

- Indigo(pseudo alea + preselect) vs Indigo(preselect)
- (Nselect = 10)

| | | | |
|---|---|---|---|
| size | 9x9 | 13x13 | 19x19 |
| mean | +8 | +40 | +100 |
| % wins | 68% | 93% | 97% |
| time | 1’30 | 10’ | 1h30’ |

# MCGo and knowledge (3)

- Features of a Pseudo-Random (PR) player
  - 3x3 pattern urgency table
  - $3^8$ patterns (empty intersection at the center)
  - 25 dispositions with the edge
  - #patterns = 250,000
  - Urgency « atari »
- « Manual » player
  - The PR player used in Indigo2004
  - Urgency table produced with a translation of an existing pattern database built « manually »
  - With a few dozens of 3x3 patterns
- « Automatic » player

# Enhancing raw UCT up to a more sophisticated UCT

- **The enhancements are various...**
  - UCT formula tuning (C tuning, “UCB-tuned”)
  - Exploration-exploitation balance
  - Outcome = Territory score or win-loss information ?
  - Doubling the random game number
  - Transposition Table
    - Have or not have, Keep or not keep
    - Update nodes of transposed sequences
  - Use grand-parent information
  - Simulated games
    - Capture, 3x3 patterns, Last-move heuristic,
    - Move number, «Mercy» rule
  - Speeding up
    - Optimizing the random games
    - Pondering
    - Multi-processor computers
    - Distribution over a (local) network

# Assessing an enhancement

- Self-play
  - Ups and downs
  - First and easy test
  - Few hundred games per night
  - % of wins
- Against one differently designed program
  - GNU Go 3.6
  - Open source with GTP (Go Text Protocol)
  - Few hundred games per night
  - % of wins
- Against several differently designed programs
  - CGOS (Computer Go Operating System)
  - Real test
  - ELO rating improvement
  - 9x9
  - Slow process

# CGOS rankings on 9x9

- ELO ratings on 6 march 2007

| | |
|---|---|
| MoGo 3.2 | 2320 |
| MoGo 3.4 10k | 2150 |
| Lazarus | 2090 |
| Zen | 2050 |
| AntiGo | 2030 |
| Valkyria | 2020 |
| MoGo 3.4 3k | 2000 |
| Irene (=Indigo) | 1970 |
| MonteGnu | 1950 |
| firstGo | 1920 |
| NeuroGo | 1860 |
| GnuGo | 1850 |
| Aya | 1820 |
| … | |
| Raw UCT | 1600? |
| … | |
| AnchorMan | 1500 |
| … | |
| Raw MC | 1200? |
| … | |
| ReadyToGo | 1000? |
| … | |

# Move selection formula tuning

- Using UCB
  - Move eval = mean + **C** * sqrt(log(t)/s)
  - What is the best value of C ?
  - Result: 60-40%
- Using “UCB-tuned” (Auer & al 2002)
  - The formula uses the variance V:
    - Move eval = mean + sqrt(log(t)***min(1/4,V)**/s)
  - Result: “substantially better” (Wang & Gelly 2006)
  - No need to tune C

# Exploration vs exploitation

- General idea
  - Explore at the beginning of the process
  - Exploit near the end
- Argmax over the child nodes with their...
  - Mean value
  - Number of random games performed (i.e. « robust-max »)
  - Result: Mean value vs robust-max = +5%
- Diminishing C linearly in the remaining time
  - Inspired by (Vermorel & al 2005)
  - +5%

# Which kind of outcome ?

- 2 kinds of outcomes
  - Win-Loss Information (WLI): 0 or 1
  - Territory Score: integer between -81 and +81
  - Combination of Both TS + Bonus*WLI
- Resulting statistical information
  - WLI: probability of winning ++
  - TS: territory expectation
- Results
  - Against GNU-Go
    - TS: 0%
    - WLI: +15%
    - TS+WLI: +17% (with bonus = 45)

# The diminishing return experiment

- Doubling the number of simulations
  - N = 100,000
- Results:
  - 2N vs N: 60-40%
  - 4N vs 2N: 58-42%

# Transposition table (1)

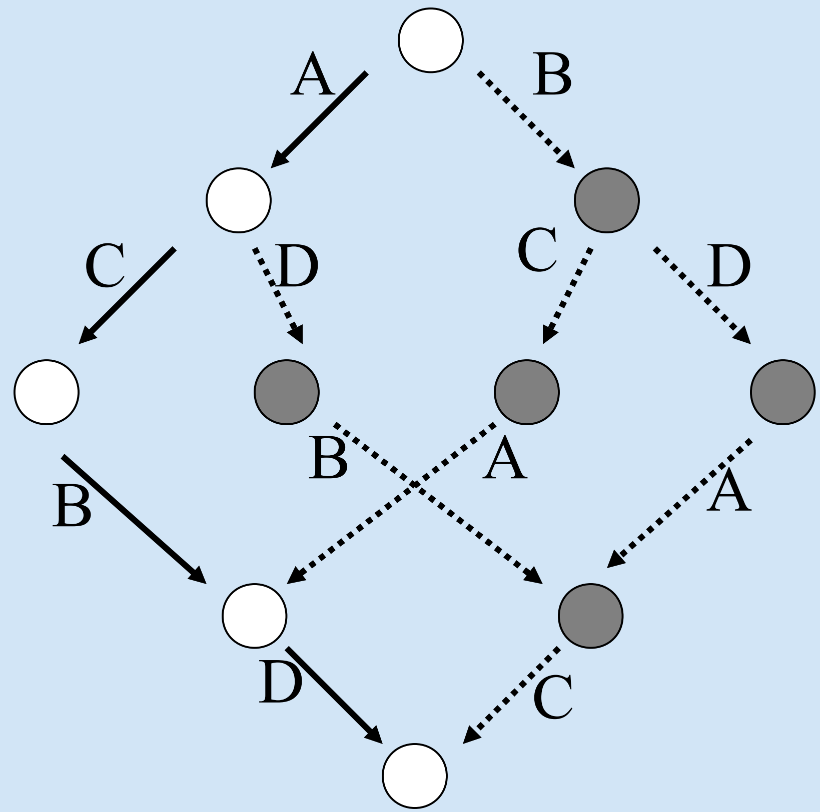
- Have or not have ?
  - Zobrist number
  - TT access time << random simulation time
    - HashTable collision solved with a linked list or records
  - Interest: merging two node information for the same position
    - Union of samples
    - Mean value refined
  - Result: 60-40%
- Keep or not keep TT info from one move to the next ?
  - Result: 70-30%

# Transposition table (2a)

- Update nodes of transposed sequences
  - If no capture occurs in a sequence of moves, then
    - Black moves could have been played in a twist order
    - White moves as well
  - There are « many » sequences that are transposed from the sequence actually played out
  - Up: one simulation updates much more nodes that the nodes the actual sequence gets through
  - Down: most of these « transposed » nodes do not exist
    - If you create them: memory explosion occurs
    - If you don't: the effect is lowered.
  - Result: 65-35%

# Transposition table (2b)

- Which nodes to update ?
- Actual
  - Sequence:
    - ACBD
  - Nodes:
- Virtual
  - Sequences:
    - BCAD, ADBC, BDAC
  - Nodes:

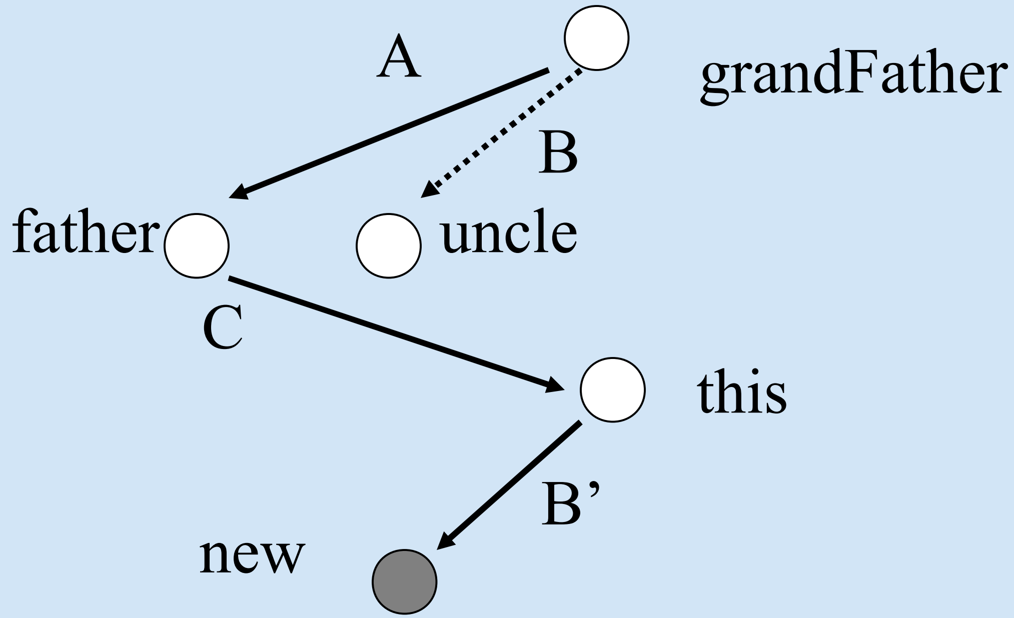


# Grand-parent information (1/2)

- Mentioned by (Wang & Gelly 2006)
- A move is associated to an intersection
  - Use statistical information available in nodes associated to the same intersection
  - For...
    - Initializing mean values
    - Ordering the node expansion
  - Result: 52-48%

# Grandparent information (2/2)

- Given its ancestors, estimate the value of a new node ?




- Idea:
  - move B’ is similar to move B because of their identical location
  - new.value = this.value +uncle.value – grandFather.value

# Simulated games improvement

- High urgency for...
  - Capturing-escaping Result: 55-45%
  - Moves advised by 3x3 patterns Result: 60-40%
  - Moves located near the last move (in the 3x3 neighbourhood)
    - (Wang & Gelly 2006)
    - Result: 60-40%
- The « mercy » rule (Hillis 2006)
  - Interrupt the game when the difference of captured stones is greater than a threshold
    - Up: random games are shortened with some confidence
    - Result: 51-49%

# Speeding up the random games (1)

- Full random on current desktop computer
  - 50,000 rgps (Lukas Lew 2006) an exception !
  - 20,000 rgps (commonly eared)
  - 10,000 rgps (my program!)
- Pseudo-random (with patterns and few knowledge)
  - 5,000 rgps (my program)
- Optimizing performance with profiling
  - Rough optimization is worthwhile

# Speeding up the random games (2)

- Pondering
  - Think on the opponent time
  - Result: 55-45%
- Parallelization on a multi-processor computer
  - Shared memory: UCT tree = TT
  - TT locked with a semaphore
  - Result: 2 proc vs 1 proc : 58-42%
- Parallelization over a network of computers
  - Like the Chessbrain project (Frayn & Justiniano)
  - One server manages the UCT tree
  - N clients perform random games
  - Communication with messages
  - Result: not yet available!

# Parallelizing MCTS

Light processes using TT

- While time do,
  - PlayOutTreeBasedGame (list)
  - outcome = PlayOutRandomGame()
  - Update nodes (list, outcome)
- Play the move with the best mean

Heavy and stand-alone process using board information and not the TT

# Scaling up to 19x19 boards

- Knowledge-based move generation
  - At every nodes in the tree
- Local MC-searches
  - Restrict the random game to a « zone »
  - How to define zones ?
    - Statically with domain-dependent knowledge
      - Result: 30-70%
    - Statistically: proper appoach, but how ?
  - Warning: avoid the difficulties of the breaking-down approach
- Parallelization
  - The promising approach

# Summing up the enhancements

- Details
  - UCT formula tuning 60-40
  - Exploration-exploitation balance 55-45
  - Proba of winning vs territory expect. 65-45
  - Transposition Table
    - Have or not have 60-40
    - Keep or not keep 70-30
    - Update nodes of transposed sequences 65-35
  - Use grand-parent information 52-48
  - Simulated games
    - Capture, 3x3 patterns 60-40
    - Last-move 60-40
    - « Mercy » rule 51-49
  - Speeding up
    - Optimizing the random games 60-40
    - Pondering 51-49
    - Multi-processor computers 58-42
    - Distribution over a network ?
- Total 99-1 ?

# Current results

- 9x9 Go: the best programs are MCTS based
  - MoGo (Wang & Gelly), CrazyStone (Coulom),
  - Valkyria (Persson), AntGo (Hillis), Indigo (Bouzy)
  - NeuroGo (Enzenberger) is the exception
  - CGOS, KGS
- 13x13 Go: medium interest
  - MoGo, GNU Go
  - Old-fashioned programs does not play
- 19x19 Go: the best programs are still old-fashioned
  - Old-fashioned go programs, GNU Go
  - MoGo is catching up (regular successes on KGS)

# Perspectives on 19x19

- To what extent MCTS programs may surpass old-fashioned program ?
  - Are old-fashioned go programs all old-fashioned ?
    - Go++ is one of the best program
    - Is Go++ Old-fashioned or MCTS based ?
  - Can old-fashioned programs improve in the near future ?
  - Is MoGo strength mainly due to MCTS approach or to the skill of their authors ?
    - 9x9 CGOS: MoGo is far ahead the other MCTS programs
  - Is the break-down approach mandatory for scaling up MCTS up to 19x19 ?
  - The parallelization question: may we easily distribute MCTS over a network ?

# Thank you for your attention...