Monte-Carlo Game Tree Search: Advanced Techniques

Tsan-sheng Hsu

徐讚昇

tshsu@iis.sinica.edu.tw

http://www.iis.sinica.edu.tw/~tshsu

Abstract

- **Adding new ideas to the pure Monte-Carlo approach for computer Go.**
  - **Domain independent knowledge during playing**
    - ▷ *Node expansion policy*
    - ▷ *Progressive pruning (PP)*
    - ▷ *All moves as first (AMAF) and RAVE heuristic*
    - ▷ *Better simulation policies*
    - ▷ *Depth-i tree search*
- **Conclusion:**
  - **Augmented with domain-independent knowledge extracted using statistical tools, Monte-Carlo approach reaches a new high for computer Go.**

Domain independent refinements

- **Main considerations**
  - **Avoid doing un-needed computations**
  - **Increase the speed of convergence**
  - **Avoid early mis-judgement**
  - **Avoid extreme cases**
- **Refinements obtained from on-line domain independent knowledge.**
  - **Node expansion policy.**
    - *Grow only nodes with potential.*
  - **Progressive pruning.**
    - *Cut hopeless nodes early in order not to corrupt the sampling statistics.*
  - **All moves at first and RAVE.**
    - *Increase the speed of convergence.*
  - **Temperature.**
    - *Introduce randomness.*
  - **Depth-i enhancement.**
    - *With regard the initial phase, the one on obtaining an initial game tree, exhaustively enumerate all possibilities instead of using only the root.*

Warning

- **Many of the domain independent refinements are invented earlier than the idea of UCT tree search.**
- **For a better flow of introduction, UCT is introduced earlier.**
- **These domain independent techniques can be used with or without UCT.**
- **These techniques speed up the convergence rate, but cannot really replace the importance of getting more simulations.**
  - **If the amount of simulations performed is well enough, then you can most likely find a good answer without using those techniques. In the worst case, you will be hurt by spending more time to do these additional techniques.**
  - **In the extreme case, if you can do well enough simulations, then no UCB formula is needed at all.**
- **Lesson: Do enough, but not over, simulations for the problem instance under the current resource constraint.**

Node expansion policy (1/2)

- **In the selection process when tracing down the PV, an expandable node h is found.**
  - **We expand the tree by growing on the part of h.**
- **Expansion policy all ends: Expand all possible children eventually of a found expandable node.**
  - **all ends one-at-a-time: expand one child per simulation, namely, a child k of h when a simulation is done from h.**
    - *Implementation I: do only one simulation from h per selection.*
    - *Implementation II: first do one simulation from h to find k and then do several more simulations from k per selection.*
  - **all ends at once: expand all children of h at once.**
    - *Implementation I: do some fixed number of, say 100, simulations for each child of h.*
    - *Implementation II:*
      - *Do a total of, say 500, simulations.*
      - *First do a small number of, say 10, simulations for each child.*
      - *For the rest, pick a child with the best UCB score to do some numbers of, say 10, simulations. Continue to do so until the required number of simulations are performed.*

Node expansion policy (2/2)

- **May decide to expand potentially good nodes judging from the current statistics [Yajima et al'11].**
  - **Visit count: delay the expansion of a child until it is visited a certain number of times.**
    - ▷ *For example: more than a certain times of the number of its siblings.*
    - ▷ *When v is to be expanded in the future, only expand the children having been simulated a certain number of times before.*
  - **Transition probability: delay the expansion of a child until the confidence of the current "score" is high enough comparing to that of its siblings.**
    - ▷ *Use the current mean, variance and parent's values to derive a good estimation using statistical methods.*

Node expansion: comments

- **Only the statistics of an expanded child are back propagated to its ancestors.**
- **In a zero sum two person game, the game tree is structured in a max-min fashion.**
  - **For a max node h, we pick the best child.**
    - ▷ *The statistics of bad children should not be considered as part of the evaluation for h.*
    - ▷ *However, it is difficult to tell which children are really bad using sampling when the number of simulations are mat large. Hence we use the average of all simulations done on children of h.*
    - ▷ *A child needs to be expended only when it has enough confidence of being the best or nearly the best child.*
  - **Similar arguments work for a min node.**
- **Expansion policy with some transition probability is much better than the "all ends" or pure "visit count" policy.**
- **When a node is expanded, its statistics are back propagated to the root. Otherwise, they are kept in the node only.**

Progressive pruning (1/4)

- A hopeless child, though being simulated less using UCB, still contributes some statistics of its ancestors.
  - If it is cut, the sampling is more accurate.
- A child may be good (respectively, bad) at the beginning, but becomes bad (respectively, good) later when more simulations are done.

Progressive pruning (2/4)

- Each position has a mean value μ and a standard deviation σ after performing some simulations.
  - Left expected outcome $\mu_l = \mu - r_d * \sigma$.
  - Right expected outcome $\mu_r = \mu + r_d * \sigma$.
  - The value r_d is a constant fixed up empirically.
- Let P_1 and P_2 be two child positions of a position P.
- P_1 is *statistically inferior* to P_2 if $P_1.\mu_r < P_2.\mu_l$, $P_1.\sigma < \sigma_e$ and $P_2.\sigma < \sigma_e$.
  - The value σ_e is called *standard deviation for equality*.
  - Its value is determined empirically.
- P_1 and P_2 are *statistically equal* if $P_1.\sigma < \sigma_e$, $P_2.\sigma < \sigma_e$, which means no one is statistically inferior to the other.

Progressive pruning (3/4)

- **After a minimal number of random games, say 100 per move, a position is pruned as soon as it is statistically inferior to another.**
  - **For a pruned position:**
    - *Not considered as a legal move.*
    - *No need to maintain its UCB information.*
  - **This process is stopped when**
    - *this is the only one move left for its parent, or*
    - *the moves left are statistically equal, or*
    - *a maximal threshold, say 10,000 multiplied by the number of legal moves, of iterations is reached.*
- **Two different pruning rules.**
  - **Hard: a pruned move cannot be a candidate later on.**
  - **Soft: a move pruned at a given time can be a candidate later on if its value is no longer statistically inferior to a currently active move.**
    - *The score of an active move may be decreased when more simulations are performed.*
    - *Periodically check whether to reactive it.*

Progressive pruning (4/4)

- **Remarks:**
  - **Assume each trial is an independent Bernoulli trial and hence the distribution is normal.**
    - ▷ *This needs to be checked in your application.*
  - **We only compare nodes that are of the same parent.**
  - **We usually compare their raw scores not their UCB values.**
    - ▷ *UCB and PP are similar in ideas, but using different pre-assumptions.*
  - **If you compare UCB scores, then the mean and standard deviation of a move are those calculated only from its un-pruned children.**
- **Important remark**
  - **When a node is pruned, its statistics are removed from the MCTS tree using back propagation.**
  - **When the punning is undo due to using a soft policy, then the statistics of the node need to be added back and then be propagated.**

PP: experiments (1/3)

- **Experimental setup:**
  - **9 by 9 Go.**
  - **Difference of stones plus eyes after Komi is applied.**
  - **The experiment is terminated if any one of the followings is true.**
    - ▷ *There is only move left for the root.*
    - ▷ *All moves left for the root are statistically equal.*
    - ▷ *A given number of simulations are performed.*
  - **The baselines of the experiments are those with scores 0.**

PP: experiments (2/3)

- **Selection of r_d.**
  - **The greater r_d is,**
    - *the less pruned the moves are;*
    - *the better the algorithm performs;*
    - *the slower the play is.*
  - **Results [Bouzy et al'04]:**

r_d	1	2	4	8
score	0	+ 5.6	+ 7.3	+9.0
time	10'	35'	90'	150'

- **Selection of σ_e.**
  - **The smaller σ_e is,**
    - *the fewer equalities there are;*
    - *the better the algorithm performs;*
    - *the slower the play is.*
  - **Results [Bouzy et al'04]:**

σ_e	0.2	0.5	1
score	0	-0.7	-6.7
time	10'	9'	7'

- **Conclusions:**
  - r_d **plays an important role in the move pruning process.**
  - σ_e **is less sensitive.**

PP: experiments (3/3)

- **Comments:**
  - **It makes little sense to compare nodes of different depths or belonging to different players.**
  - **Another trick that may need consideration is progressive widening or progressive un-pruning.**
    - ▷ *A node is effective if enough simulations are done on it and its values are good.*
  - **Note that we can set a threshold for the number of simulations done on whether to expand or grow the end of the selected PV$_{UCB}$ path.**
    - ▷ *This threshold can be enough simulations are done and/or the score is good enough.*
    - ▷ *Use this threshold to control the way the underline tree is expanded.*
    - ▷ *If this threshold is high, then it will not expand any node and looks like the original version.*
    - ▷ *If this threshold is low, then we may make not enough simulations for each node in the underline tree.*
  - **If you want to do the above, you need to use a hash table to store the number of simulations done a node and its win rate.**

Comments on using PP

- **Important remarks:**
  - Ideas for using the confidence interval on PP and the ideas for using upper and lower confidence bounds (LCB and UCB) are similar.
  - Statistical issues.
    - ▷ *If the result of a simulation can only be 0 or 1, then the mean of a sampling uniquely determines its standard deviation.*
    - ▷ *If the result of a simulation can have only very few variations, e.g., -1, 0, 1, then there are only a few possible standard deviations once the mean of a sampling is given.*
  - The range of possible scores is important in using PP.
    - ▷ *A very narrow range makes the cutting not very flexible.*
    - ▷ *A very wide range makes the cutting too random.*

Types of nodes during MCTS searching

- **A node in an MCTS tree can have the following flags marked.**
  - **Officially grown or not.**
    - *Being considered expanded by an expansion policy.*
  - **Expandable or not.**
    - *Expandable: some child is not officially grown.*
  - **Being PP cut or not.**
    - *Hard or soft cut.*
    - *Only an officially grown node can be cut.*

All-moves-as-first heuristic (AMAF)

- **How to perform statistics for a completed playout?**
  - **Basic idea: its score is used for the first move of the game only.**
  - **All-moves-as-first AMAF: its score is used for all moves played in the game as if they were the first to be played [Bruegmann'93].**
- **AMAF updating rules:**
  - **If a playout S, starting from the position following PV$_{UCB}$ towards the best leaf and then appending a simulation run, passes through a position V from W with a sibling position U, then**
    - ▷ *the counters at the position V leads to is updated;*
    - ▷ *the counters at the node U leads to is also updated if S later contains a ply from W to U.*
  - **Note, we apply this update rule for all nodes in S regardless nodes made by the player that is different from the root player.**

Illustration: AMAF (1/2)

- Assume a playout P is simulated from the child Y of the leaf L in the PV with the sequence of plys starting from the position Y being y, u, w, $\cdots$.
  - ▷ ***From node L to Y is the ply v.***
- The winning rate of Y is first updated.
  - ▷ ***Due to the playout P.***
- The winning rate of X, a child of L, is also updated.
  - ▷ ***Due to the added playout P' at X, exchange u and v in P.***
- The winning rate of Y is updated again.
  - ▷ ***Due to the added playout P'' at Y, exchange w and y in P.***

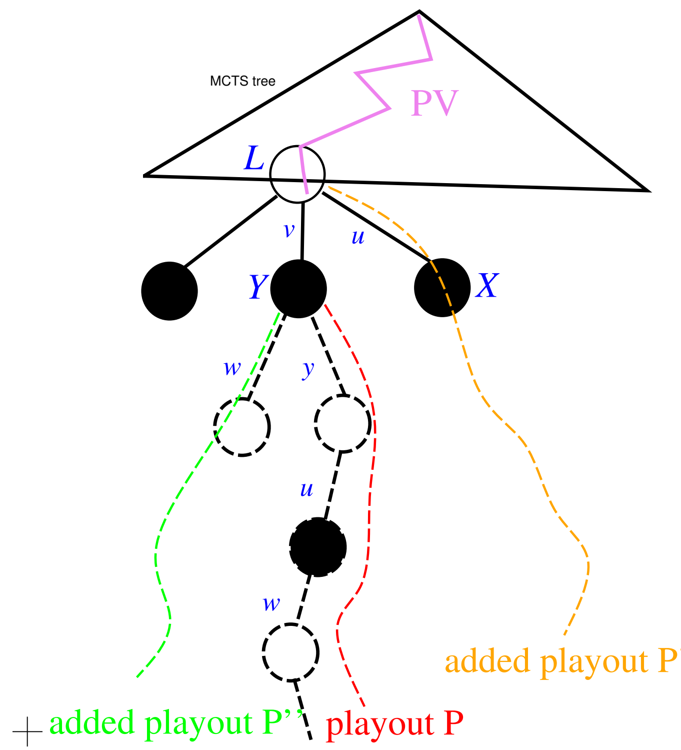
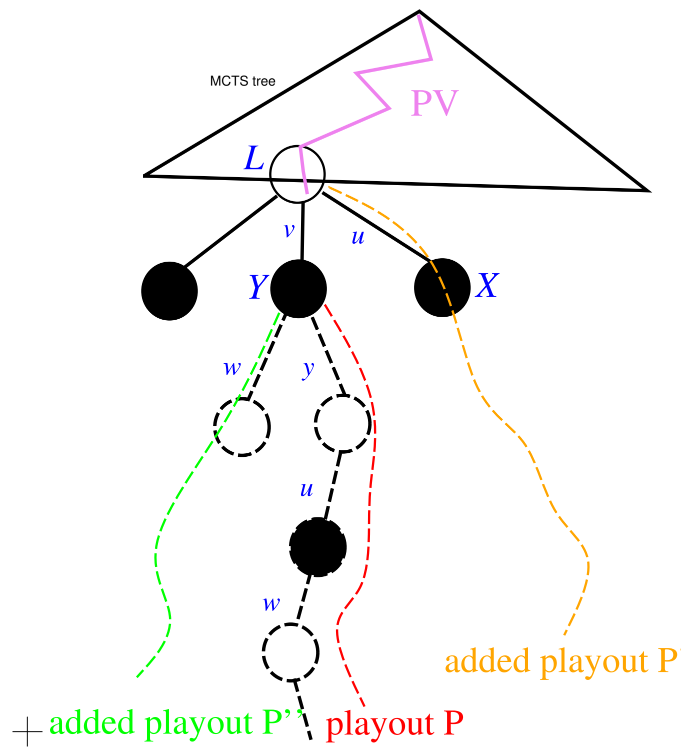


Illustration: AMAF (2/2)

- In this example, 3 playouts are recorded for the position L though only one is performed.
- Note: Need to also update the scores of affected nodes.



AMAF: Implementation (1/2)

- **When a playout, say $a_1, a_2, \ldots, a_h, \ldots, a_g$ is simulated where a_1 is the root position of the selected PV$_{UCB}$ and a_h is the expandable node and $S = a_{h+1}, \ldots, a_g$ is the newly added simulation, then we perform the following updating operations bottom up:**
  - **for** $i := h + 1$ **downto** 1 **do**
    - ▷ ***for each child position*** c ***of*** a_i ***that is not equal to*** a_{i+1} ***do***
    - ▷ ***if the ply*** $(a_i \to c)$ ***is played in*** $a_{h+1}, a_{h+2}, \ldots, a_g$ ***then***
    - ▷ ***{***
    - ▷ ***update the score and counters of*** c ***as if the playout*** S ***is also occurred in*** c***;***
    - ▷ ***}***

AMAF: Implementation (2/2)

- **Some forms of hashing is needed to check the if condition efficiently.**
  - **Use a bit array** $PT[pieces][fromposition][toposition]$
  - **initialize** $PT[][][]$ **to be all 0's.**
  - **When a ply, namely move a piece** p **from a location** f **to the destination** d**, in the simulated playout is picked,** $PT[p][f][d]$ **is set to 1.**
  - **Hence it is** $O(1)$ **time to check whether a particular ply is made in the simulated playout later.**
  -
- **It is better to use a good data structure to record the children of a position when it is first generated to avoid regenerating.**

AMAF: Pro's and Con's

- **Advantage:**
  - **All-moves-as-first helps speeding up the convergence of the simulations.**
- **Drawbacks:**
  - **The evaluation of a move from a random game in which it was played at a late stage is less reliable than when it is played at an early stage.**
  - **Recapturing.**
    - ▷ *Order of moves is important for certain games.*
    - ▷ *Modification: if several moves are played at the same place because of capturing, modify only the statistics for the player who played first.*
  - **Some move is good only for one player.**
    - ▷ *It does not evaluate the value of an intersection for the player to move, but rather the difference between the values of the intersections when it is played by one player or the other.*

AMAF: results

- **Results [Bouzy et al'04]:**
  - **Relative scores between different heuristics.**

AMAF	basic idea	PP
0	+13.7	+ 4.0

    - ▷ *Basic idea is very slow: 2 hours vs 5 minutes.*
  - **Number of random games N: relative scores with different values of N using AMAF.**

N	1000	10000	100000
score	-12.7	0	+3.2

    - ▷ *Using the value of 10000 is better.*
- **Comments:**
  - **The statistical natural is something very similar to the history heuristic as used in alpha-beta based searching.**
  - **Be aware of counter overflow.**

AMAF refinements

- **Definitions:**
  - **Let $v_1(P)$ be the score of a position P without using AMAF.**
  - **Let $v_2(P)$ be the score of a position P with AMAF.**
    - ▷ ***In calculating $v_2(P)$ we need to take into consideration all playouts, actual and added ones.***
    - ▷ ***It is odd to use only added playouts to compute.***
- **Remark: $v_2(P)$ uses both information of actual playouts and the added playouts from AMAF, while $v_1(P)$ uses only information from actual playouts only.**
- **Observations:**
  - **$v_1(P)$ is a good indicator for the goodness of P when sufficient number of trials are performed starting with P.**
  - **$v_2(P)$ is a good guess for the goodness of P for the true score of the position P when**
    - ▷ ***it is approaching the end of a game;***
    - ▷ ***too few trials are performed starting with P such as when the node for P is first expanded.***
- **Q: How to make the best use of $v_1(P)$ and $v_2(P)$ together?**

RAVE

- **Definitions:**
  - **Let $v_1(P)$ be the score of a position P without using AMAF.**
  - **Let $v_2(P)$ be the score of a position P with AMAF.**
- **Rapid Action Value Estimate (RAVE) [Silver'09]**
  - **Let the revised score $v_3(P) = \alpha \cdot v_1(P) + (1 - \alpha) \cdot v_2(P)$ with a properly chosen value of α.**
    - ▷ ***Other formulas for mixing the two scores exist.***
  - **Can dynamically change α as the game goes.**
    - ▷ ***For example: $\alpha = \min\{1, \frac{N_P}{10000}\}$, where N_P is the number of playouts done on P.***
    - ▷ ***This means when N_P reaches 10000, no AMAF is used.***
- $v_3(P) = \alpha \cdot v_1(P) + (1 - \alpha) \cdot v_2(P)$
  - **When $\alpha = 0$, it is pure AMAF.**
  - **When $\alpha = 1$, it uses no AMAF.**

Other formulations of RAVE (1/2)

- **Note:** $v_3(P) = \alpha \cdot v_1(P) + (1 - \alpha) \cdot v_2(P)$
- **Example: Silver in his 2009 Ph.D. thesis [Silver'09] originally set the parameters as follows:**
  - **Let** $\tilde{N}_P = N_P + N'_P$ **where** N_P **is the number of actual simulations done at the position** P **and** N'_P **is the number of extra added simulations generated from AMAF at** P**.**
    - ▷ *$\tilde{N}_P$ is the total number of simulations (actual and added) used to generate the AMAF score $v_2(P)$.*
    - ▷ *N_P is the total number of actual simulations used to generate $v_1(P)$.*
  - $1 - \alpha = \beta = \frac{\tilde{N}_P}{N_P + \tilde{N}_P + 4b^2 N_P \tilde{N}_P}$ **where** b **is a constant to be decided empirically.**
  - **Namely,** $v_3(P) = (1 - \beta) \cdot v_1(P) + \beta \cdot v_2(P)$

Other formulations of RAVE (2/2)

- **Note:** $v_3(P) = (1-\beta)\cdot v_1(P) + \beta \cdot v_2(P)$
- **Discussion:**
  - $\beta = \frac{1}{\frac{N_P}{\tilde{N}_P}+1+4b^2 N_P}$
  - **We know $\tilde{N}_P \geq N_P$, hence $\frac{1}{2+4b^2N_P} \leq \beta \leq \frac{1}{1+4b^2N_P}$.**
  - **When $N_P >> 1/(4b^2)$ is large, then $\beta \to 0$ which means uses mostly information in $v_1(P)$.**
    - ▷ ***When N_P is small, β is larger.***
  - **For the same $\tilde{N}_P$, if N_P is smaller, then β is larger, which means using more information in $v_2(P)$.**
- **Comments:**
  - **Silver is the first one to propose RAVE, but we choose to introduce a simpler formulation earlier for ease of description.**

Better simulation policies

- **Methods to make the sampling done by a simulation is a better approximation of the goodness of the current position.**
  - **Using the idea of "temperature" to control the degree of randomness in sampling.**
  - **Stop the simulation earlier if the result of the simulation can be inferred: early playout termination [Hsueh et al '16]**
  - **Assign proper scores to simulations whose results are more confident than the others: quality based rewards [Hsueh et al '16]**

Temperature

- **Considering all scores in MCTS are estimations with errors, try to make decisions allowing errs, not preciously.**
- **Do not always pick one with the best score since the second best may be just off a bit. Give each one a chance to be picked according to its score.**
  - **The probability of playing the ith move is $P_i = \frac{score_i}{\sum_{\forall q} score_i}$.**
- **Add a degree of randomness, called temperature, in picking the plys in the random playout.**
  - **Constant temperature**
  - **Temperature from high to low.**

Constant temperature (1/2)

- **Constant temperature: consider all the legal moves and play the ith move with a probability proportional to $e^{(K \cdot v_i)}$, where**
  - v_i **is the current value of the position obtained by taking move** i**;**
    - ▷ ***It is usually the case*** $v_i \geq 0$.
    - ▷ $e^{(K \cdot v_i)} \geq 1$.
  - $K \geq 0$ **is the inverse of the temperature** T**, a given constant, used in a simulated annealing setting.**
    - ▷ ***Add extra randomness by setting a constant*** $K = 1/T$.
    - ▷ ***The probability of playing the*** i***th move is*** $P_i(K) = \frac{e^{K \cdot v_i}}{\sum_{\forall q} e^{K \cdot v_q}}$.
    - ▷ ***When*** $K \to 0$, ***which means temperature*** $T \to \infty$, ***and the selection is uniformly random.***
    - ▷ ***If*** $v_i > v_j$ ***and*** $K_1 > K_2$, ***then*** $P_i(K_1) - P_j(K_1) > P_i(K_2) - P_j(K_2)$.
      $\rightarrow$ ***When*** K ***becomes larger,***
      the value of v_i ***contributes more in the calculation of*** $P_i(K)$.
    - ▷ ***When*** K ***is very large, which means temperature is very low, it looks like some form of the "greedy", or best first, approach.***

Constant temperature (2/2)

- **Results for using a constant temperature [Bouzy et al'04]:**

K	0	2	5	10	20
score	-8.1	0	+2.6	-4.9	-11.3

  - **When temperature is very high ($K = 0$) when means pure random, then it looks bad.**
  - **When there is no added randomness ($K > 5$), it also looks bad.**
  - **Tradeoff between the current score and randomness.**
    - ▷ ***Currently, a greedy approach is worse than a random approach!!!***

Temperature from high to low

- **Simulated annealing (temperature decreasing, or K increasing):** $P_i(K_t) = \frac{e^{K_t \cdot v_i}}{\sum_{\forall q} e^{K_t \cdot v_q}}$ **where K_t is the value of K at the tth moment.**
  - **Change the temperature, namely $1/K$, over the time.**
    - ▷ ***In the beginning, allow more randomness, and decrease the amount of randomness over the time.***
  - **Increasing K from 0 to 5 gradually over the time does not enhance the performance [Bouzy et al'04].**

Early playout termination

- **When the result of an on-going simulation can be inferred, terminate it earlier to save time.**
  - **A sure win/loss.**
    - ▷ ***Example: In CDC, the opponent has no cannon and you have a piece that can capture all of your opponent's pieces.***
  - **A draw.**
    - ▷ ***Example: In Chinese chess, both sides as no attacking pieces.***

Quality based rewards

- **When a simulation is done, assign score to better reflect the confidence of such a simulation.**
  - **A big win/loss usually means the position where the simulation starts is food/bad.**
    - ▷ ***Example: Assign the score according the difference of remaining material values in Chinese chess.***

Depth-i enhancement

- **Try to do a limited exhaustive enumeration at the beginning to beat extremely bad luck in using randomness.**
- **Algorithm:**
  - **Enumerate all possible positions from the root after i moves are made.**
  - **For each position, use Monte-Carlo simulation to get an average score.**
  - **Use a minimax formula to compute the best move from the average scores on the leaves.**
- **Result [Bouzy et al'04]: depth-2 is worse than depth-1 due to oscillating behaviors normally observed in iterative deepening.**
  - **Depth-1 overestimates the root's value.**
  - **Depth-2 underestimates the root's value.**
  - **It is computational difficult for computer Go to get depth-i results when $i > 2$.**

Putting everything together

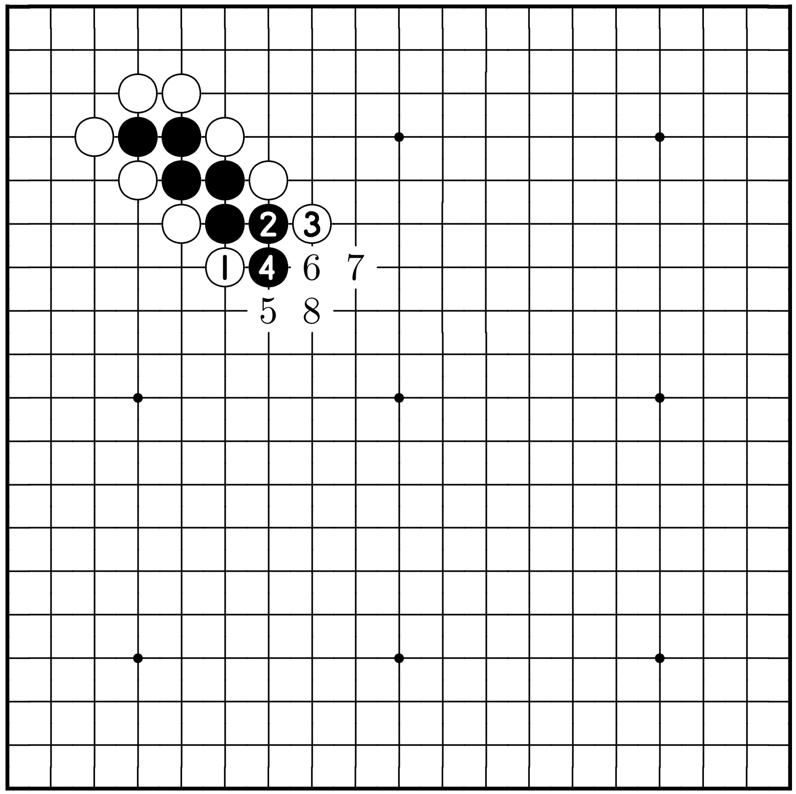
- **Two versions [Bouzy et al'04]:**
  - **Depth = 1, $r_d = 1$, $\sigma_e = 0.2$ with PP, and basic idea.**
  - **$K = 2$, no PP, and all-moves-as-first.**
- **Still worse than GnuGo in 2004, a Go program with lots of domain knowledge, by more than 30 points.**
- **Note: as we said before, most of the techniques are invented before UCT.**
  - **The idea of UCT is not part of "everything" used in his experiments.**
  - **This somehow shows that the idea of UCT may be critical among all techniques.**

Conclusions

- **Add tactical search: for example, ladders.**
  - **A ladder is a kind of string whose live-or-death is certain many plys ahead.**
- **Add more domain knowledge besides no filling of eyes: for example, in Go, simulate extending plys first.**
  - **An extending ply is one which increases the liberty of some strings that are in danger.**
- **As the computer goes faster, more domain knowledge can be added.**
- **Exploring the locality of Go using statistical methods.**

Ladder

- **White to move next at 1, then black at 2, then white at 3, and then black at 4, ...**



Ladder: comments

- **Ladder in Go is a perfect example to illustrate the idea of getting the "right" sampling is important.**
  - **In the previous shown Ladder example, it is very bad for BLACK.**
  - **However, the WHITE only has one correct response out of a few hundreds of bad ones.**
  - **If you do uniform sampling, then the odds of finding the right one is remote.**
- **The "true" meaning of doing a "fair" random sampling is thus**
  - **when the position is good, do sampling so that the final outcome of a playout is more likely to be good;**
  - **when the position is bad, do sampling so that the final outcome of a playout is more likely to be bad.**
- **"Fair" sampling will be a very hard, though may not be impossible, task for a program that has no domain knowledge.**
  - **"Fairness" has something to do with your opponent.**
    - ▷ *If your opponent is weak, then thinking too much may not be optimal.*

Comments

- **We only describe some specific implementations of some general Monte-Carlo techniques.**
  - **Other implementations exist for say AMAF and others.**
- **Depending on the amount of resources you have, you can**
  - **decide the frequency to update the node information;**
  - **decide the frequency to re-pick PV$_{UCB}$;**
  - **decide the frequency to prune/un-prune nodes.**
- **Most of the methods introduced have a statistical flavor.**
  - **First the heuristic is "discovered" based on some clever intuitions or observations.**
  - **Then people try to fine tune the parameters used in the heuristic manually.**
  - **Finally statistical tools are found or established to formally settle it.**
- **Over-use too many heuristics may cause bad side effects.**
  - **A warning for using the cock tail styled method.**
    - ▷ ***Do not know where the real contribution comes from.***
    - ▷ ***Using too much resource.***

References and further readings (1/2)

* Sylvain Gelly and David Silver. Combining online and offline knowledge in UCT. In *Proceedings of the 24th international conference on Machine learning*, ICML '07, pages 273–280, New York, NY, USA, 2007. ACM.
* David Silver. Reinforcement Learning and Simulation-Based Search in Computer Go. PhD thesis, University of Alberta, 2009.
* B. Bouzy and B. Helmstetter. Monte-Carlo Go developments. In H. Jaap van den Herik, Hiroyuki Iida, and Ernst A. Heinz, editors, *Advances in Computer Games, Many Games, Many Challenges, 10th International Conference, ACG 2003, Graz, Austria, November 24-27, 2003, Revised Papers*, volume 263 of *IFIP*, pages 159–174. Kluwer, 2004.

References and further readings (2/2)

- **Coulom, R. (2007). Computing Elo ratings of move patterns in the game of go. In Computer games workshop.**
- **Takayuki Yajima, Tsuyoshi Hashimoto, Toshiki Matsui, Junichi Hashimoto, and Kristian Spoerer. Node-expansion operators for the UCT algorithm. In H. Jaap van den Herik, H. Iida, and A. Plaat, editors,** *Lecture Notes in Computer Science 6515: Proceedings of the 7th International Conference on Computers and Games*, **pages 116–123. Springer-Verlag, New York, NY, 2011.**
- **Chu-Hsuan Hsueh, I-Chen Wu, Wen-Jie Tseng, Shi-Jim Yen, Jr-Chang Chen, An analysis for strength improvement of an MCTS-based program playing Chinese dark chess,** *Theoretical Computer Science*, **Volume 644, 2016, Pages 63-75, ISSN 0304-3975.**
- **Couëtoux A., Hoock JB., Sokolovska N., Teytaud O., Bonnard N. (2011) Continuous Upper Confidence Trees. In: Coello C.A.C. (eds) Learning and Intelligent Optimization. LION 2011.**

Lecture Notes in Computer Science, vol 6683. Springer, Berlin, Heidelberg.

- Chaslot, Guillaume, Winands, Mark, Herik, H., Uiterwijk, Jos, Bouzy, Bruno. (2008). Progressive Strategies for Monte-Carlo Tree Search. New Mathematics and Natural Computation. 04. 343-357. 10.1142/S1793005708001094.