Review of a survey over MCTS-RAVE and HaHaNoGo with Implementation of Eazynogo

Lin Yan-Tong, National Chiao Tung University, 2019.10.16

Contents:

EazyNoGo:

https://github.com/EazyReal/EazyNoGo

Surveyed Resources:

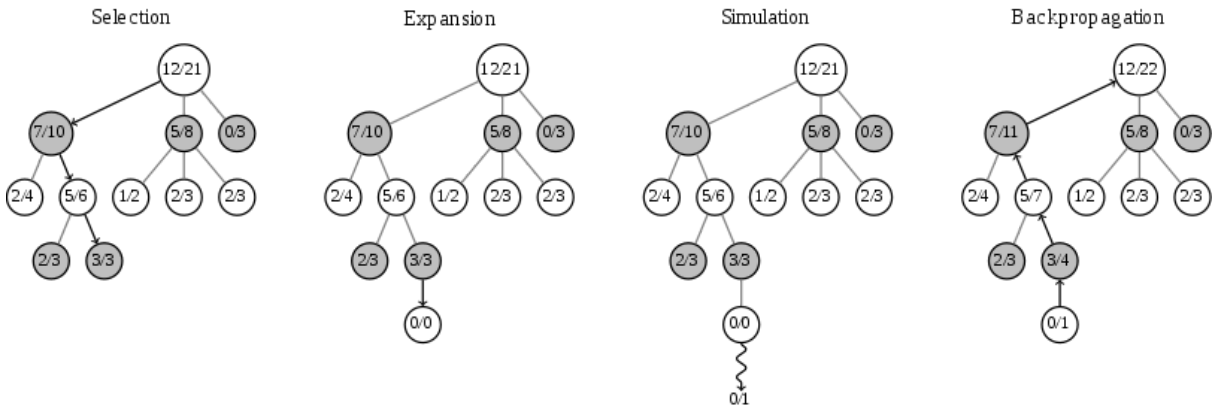
Monte-Carlo Tree Search and Rapid Action Value Estimation in Computer Go Sylvain Gelly, David Silver
http://www.cs.utexas.edu/~pstone/Courses/394Rspring13/resources/mcrave.pdf
HaHaNoGo github repo. by lclan1024
https://github.com/lclan1024/HaHaNoGo
some other paper/website about MCTS and RAVE

Understanding of MCTS(Monte Carlo Tree Search):

MCTS is Tree Search combined with Monte Carlo method.

When trying to solve a game, we are actually searching for “good actions to take in the tree of game states”, so searching algorithms are the basic of AI game study.
However, sometimes the tree may be too large for our current computation resource to search completely, or even just save the state nodes.
This is when Monte Carlo Method comes in.

For the width of the tree: only spend computational resource on “promising” nodes.
For the depth of the tree: “simulation” games to get “data” about how good the action/state is.
combined we get MCTS algorithm.
“promising” => points with “good value(ex:reward/win rate)” or not visited enough time to see how good it is => UCB score is used
“simulation” => use defaut good enough or even random policy to play the game to the end many times to get data
“data” => the result of rollout, the expectation of data shows high variance but low bias information about the value of point by sampling its subtree with default policy.



Understanding of RAVE(Rapid Action Value Estimation):

When applying MCTS, a huge amount of simulation data is required to get good inference about how well an action will perform.

How can we speed up the convergence?Can we use the data more efficiently?

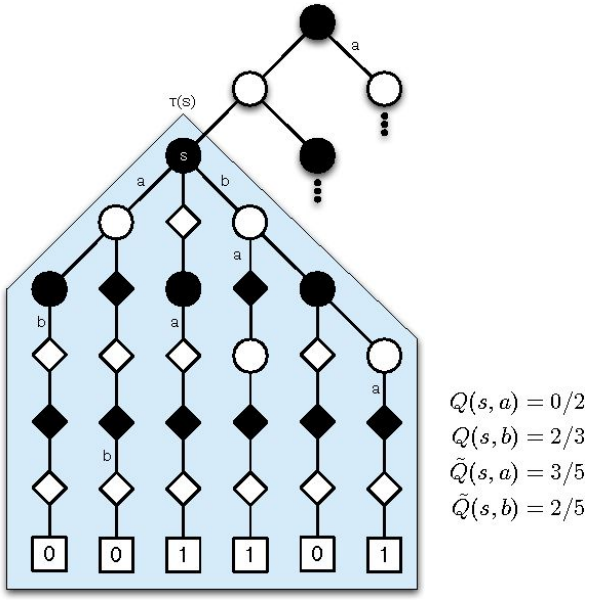
RAVE is a method with the “AMAF”(all move as first) heuristic with a assumption that:

In many cases, an action which is good in the future is often good now.

So Q(S,A) and be estimated with Q(S in subtree, A) with acceptable bias to give a good initial guess of Q(S,A) in order to search well when the nodes are just expanded.(work similarly as heuristic function)

So, we can use RAVE value when the node has not enough data and take it less seriously as the visited count rises.

In this way, we can make use of rave heuristic Q(A) and avoid rave value to be considered too much to cause bias when the understanding of Q(S,A) value is good enough.



$$Q(s,a) = 0/2$$
$$Q(s,b) = 2/3$$
$$\tilde{Q}(s,a) = 3/5$$
$$\tilde{Q}(s,b) = 2/5$$

Using RAVE on the task of NoGo:

domain knowledge:

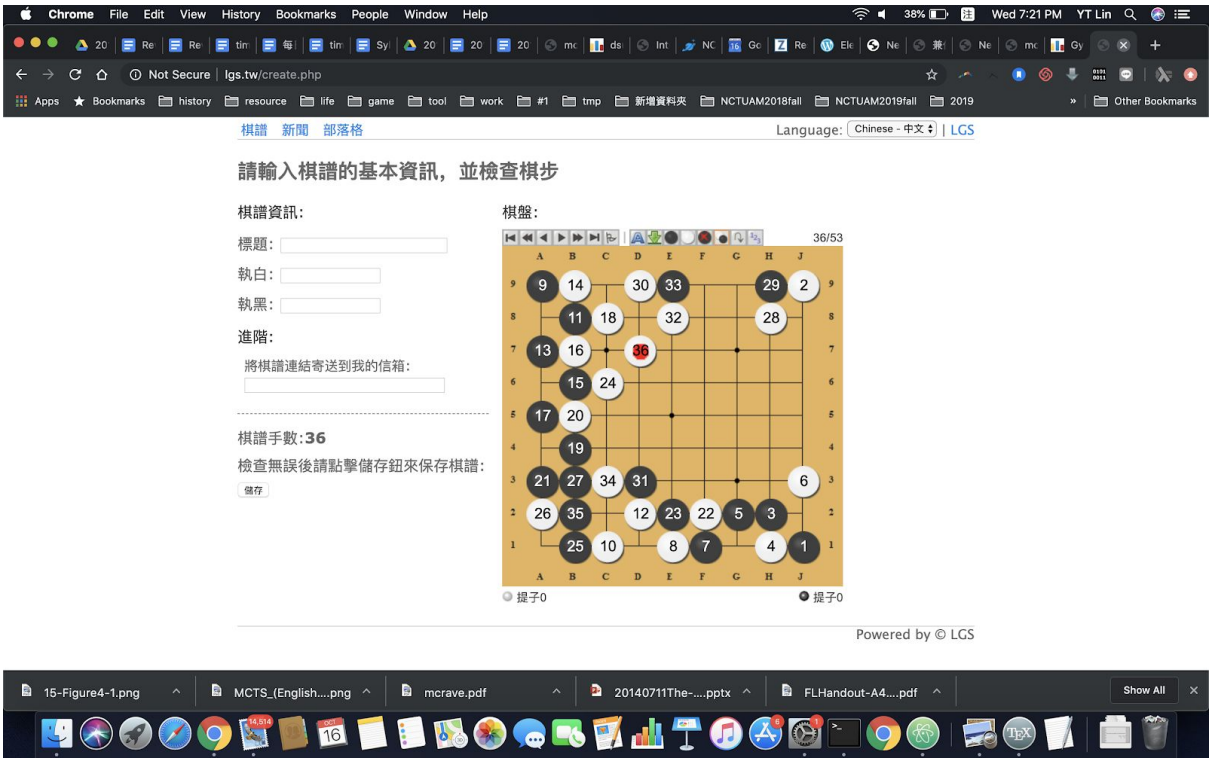
1-go/2-go move=>
going 2-go move before 2-go move is bound to be better is easily proved.
so rave value shouldn't encourage move as 1-go move.

the move which is good "before" is often good "after"=>
update the whole path instead of path in the subtrees

tuning =>
UCB = 0.25 : care less about unvisited move(with low rave heuristic)!!
b = 0.0001 => ~= 0 => Q* = win+rwin/cnt+rcnt : care a lot about rave!!

Implementation of EazyNoGo in steps:
survey hahanogo code in depth, use its board part
structure(code begin with what "function" and "data" is needed) by scratch
wrong implementation(note viewing the state as "afterstate" correctly) + debug
using simulation counter to end loop instead of time limit(improve the performence a lot)
afterstate/theorem adjustment(sqrt(k/3n+k) =>n~/n+n~+4*n*n~*b^2, b = 0)
board copy => board reference to improve program efficiency
mistype error discovered by white filling in its own eyes in the end game
add BASENUM+1 expand threshold & onestep lookahead (hahanogo trick, expand when visit twice)

When implementing, printing out policy of the MCTS tree node is important.
And during experiments, theoretical bugs or wrong implementation can be caught by weird moves in the sgf game logs.



ex:move 35 36 shows that eznogo patch 3.4 is a lot worse then hahanogo(bad priority)

Expreriments:

(final results, games played with hahanogo)

simu/step | eznogo as w/b | ezbogo win:lose

1k w 7:3

1w b 5:0, w 3:2

5w w 6:4

10w b 19:11, 10w w 12:18

50w simu/step: (black)eznogo vs (white)hahanogo 7:3

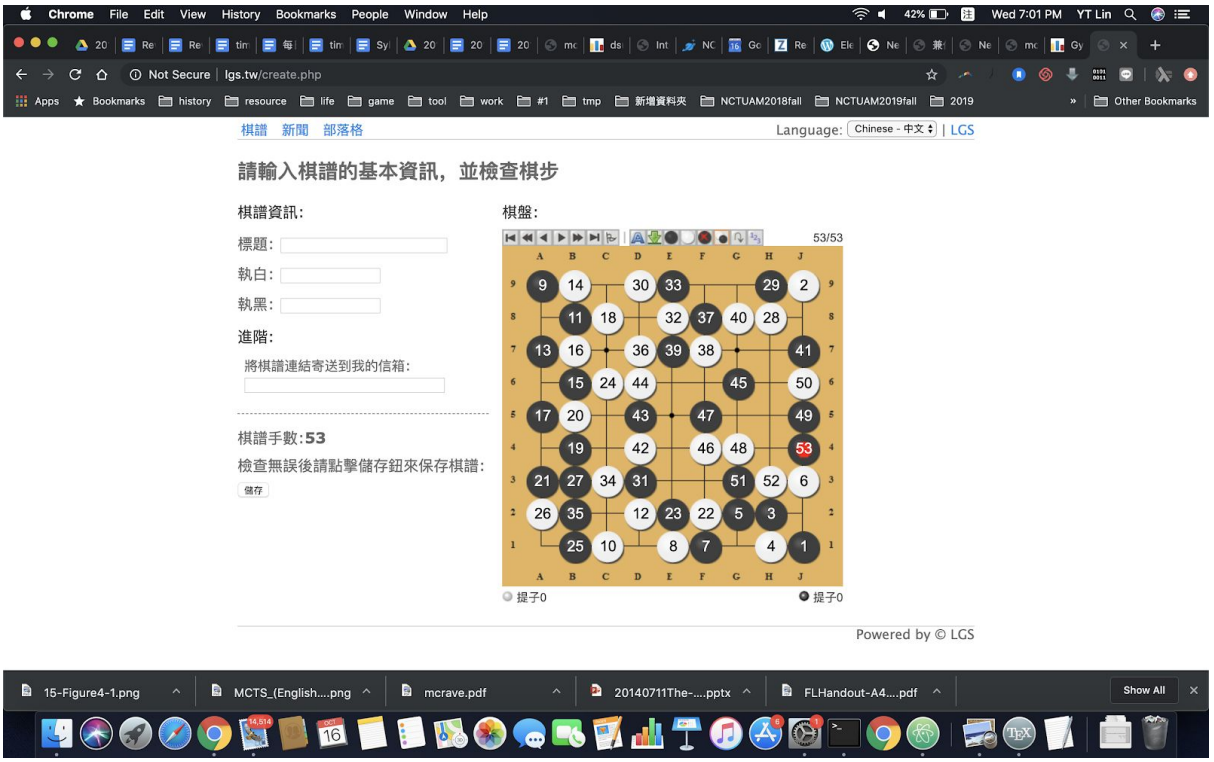
etc.

(simple unrecorded experiments on eznogo middle version, to try hahanogo tricks)

expand when (first visit/ second visit) : second visit >> first visit

rave update with(tp = 0/t/2) : 0 > t/2

etc.



(black) eznogo vs (white) hahanogo 50w simu B+R

Inference:
the final version of eazynogo uses a lot of tricks from haha nogo, so the programs are quite similar in essense.
However, in large simulation count test cases, eazynogo wins a lot more than hahanogo, my inference is that : hahanogo program records cnt and mean(win rate) as float in its node, while eazynogo use cnt and num(win number) as double. so when the number goes large, its possible that eazynogo have better performance for higher precision than hahanogo does.

Possible Improvement:
These are some of my ideas to improve eazynogo program:

Save information of the nodes during/across the game:
Since the data is unbiased, high variance estimation of win rate of game node,
saving the past data, instead of searching from the current node scratch every time would give us a lot of extra usable data.
However, the implementation would cost a lot of code work and space, so is rarely used in practice, said by TA.

Save heuristic table in different stage of the game(across games) might help:
Can determine the “stages” by depth. And use past game data as heuristic in searching.