Teye

Captain : Wu Lisheng

No: 5130309793

Catalog

1. Background

I made the name for my team. **Teye** is Typhoon Eye, which means "silence in chaos" to me. I hope that my team would have both power and silence. They are both important in everything to be done.

I didn't proceed to write the program immediately. Starting from the history and simple rules [7] to have an overview of the development of the computer go. Patterns, knowledge database were widely used in the early attempts. Then we have Monte-Carlo and machine learning. As a result, I decided to take the Monte-Carlo tree search as the basic framework.

I was recommended to have a look at AMAF in the same time. Then I found [4, 6] to have further understanding. [4] provides a clear framework to claim the three major things I would apply to **teye** later ---- **MCTS**, **UCT**, **AMAF**. Then I kept in mind that MCTS use simulations randomly to evaluate the positions, UCT adds bonus to the positions less travelled by and AMAF takes advantage of the steps in the simulation to help tree-search towards the good moves. I also watched the lectured given by the leader of the computer go program of **Baidu ---- bingo** to know more details.

Additionally, I would not paste any code of **teye** which seems redundant to me in this report.

2. Base

We have the **brown** as the basic program to modify. The program **brown** plays the go randomly but would not violate the rules and fill its own eyes. I took one morning to understand its structure and the important parts of the **gtp** communication.

All its functions only use the board that is defined as global variable. The board represents the current situation of the program. The key point is, if I want to take some simulations, I must change the board used as the parameters passed to the function. Thus, I added the board as the parameters to almost all functions in **brown**.

3. MCTS

In the early exploration, I had no clear ideas about how the MCTS plays. I was puzzled at some points like how many times when one node is visited is enough to expand the tree. When I had hesitated for a long time, I decided to just have a try. Then, actually, I have some wrong versions of MCTS. Moreover, the first version is more like a stupid history for me when I look back now.

3.1 Failed MCTS

The papers says when black wins the outcome is 1 in the **playout**. First misunderstanding is that I regarded it as the outcome is 1 when it's black-node's turn and black wins. Thus, I lost some useful information and decreased the winrate of all nodes as a result.

The second major misunderstanding is that I thought when the number of visits necessary to expand is large enough, the outcome can be more reliable. Thus, I set the number to be 8. In my view now, it was really ridiculous. Smaller the number is, the deeper and larger the tree is. Then, the result of MCTS could be more accurate.

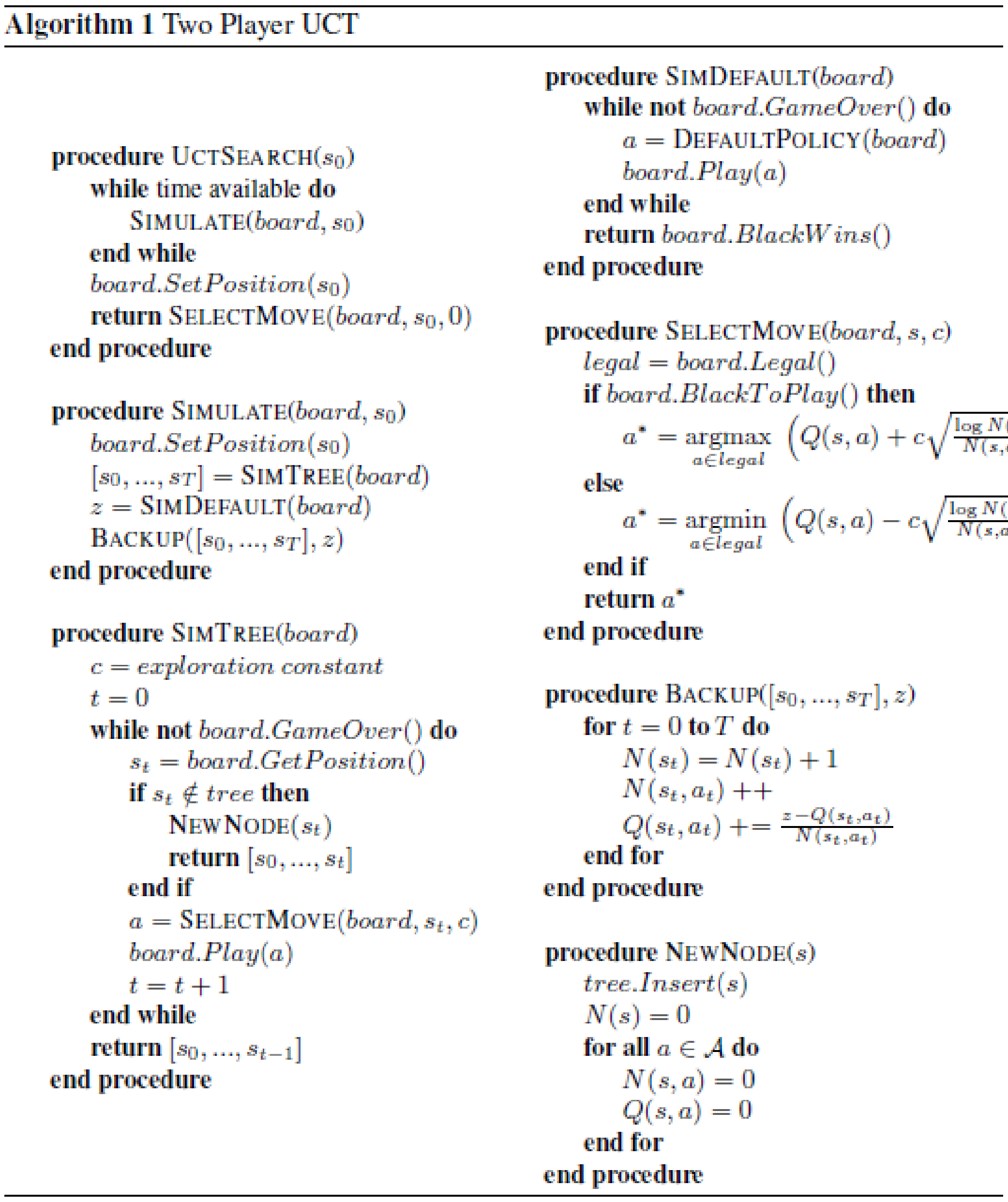
3.2 Successful MCTS

Due to the early version equipped with UCT are too weak in contrast to my roommate's one, I start to find reasons. Thanks to [1] , its clear pseudocode and graph helped me a lot to have a deep understanding of MCTS**. [*Figure 1*]**

Moreover, I set the number of visits to expand as one.

4. UCT

UCT can be said as the easiest part in the process. The main time of debugging in this part is spent on alternating the choice between **UCB1** and **UCB1-tuned**. ***[Table 1]*** After some changes, it seems **UCB1-tuned** can outperform **UCB1** in most cases**.** Then just replace the **UCB1** in the pseudocode in the following ***[Figure 1]*** with **UCB1-tuned**.



Algorithm 1 Two Player UCT

```
procedure UCTSEARCH($s_0$)
    while time available do
        SIMULATE($board, s_0$)
    end while
    $board.SetPosition(s_0)$
    return SELECTMOVE($board, s_0, 0$)
end procedure

procedure SIMULATE($board, s_0$)
    $board.SetPosition(s_0)$
    $[s_0, ..., s_T]$ = SIMTREE($board$)
    $z$ = SIMDEFAULT($board$)
    BACKUP($[s_0, ..., s_T], z$)
end procedure

procedure SIMTREE($board$)
    $c = exploration\ constant$
    $t = 0$
    while not $board.GameOver()$ do
        $s_t = board.GetPosition()$
        if $s_t \notin tree$ then
            NEWNODE($s_t$)
            return $[s_0, ..., s_t]$
        end if
        $a$ = SELECTMOVE($board, s_t, c$)
        $board.Play(a)$
        $t = t + 1$
    end while
    return $[s_0, ..., s_{t-1}]$
end procedure

procedure SIMDEFAULT($board$)
    while not $board.GameOver()$ do
        $a$ = DEFAULTPOLICY($board$)
        $board.Play(a)$
    end while
    return $board.BlackWins()$
end procedure

procedure SELECTMOVE($board, s, c$)
    $legal = board.Legal()$
    if $board.BlackToPlay()$ then
        $a^* = \operatorname{argmax}_{a \in legal} \left( Q(s,a) + c\sqrt{\frac{\log N(s)}{N(s,a)}} \right)$
    else
        $a^* = \operatorname{argmin}_{a \in legal} \left( Q(s,a) - c\sqrt{\frac{\log N(s)}{N(s,a)}} \right)$
    end if
    return $a^*$
end procedure

procedure BACKUP($[s_0, ..., s_T], z$)
    for $t = 0$ to $T$ do
        $N(s_t) = N(s_t) + 1$
        $N(s_t, a_t) ++$
        $Q(s_t, a_t) += \frac{z - Q(s_t, a_t)}{N(s_t, a_t)}$
    end for
end procedure

procedure NEWNODE($s$)
    $tree.Insert(s)$
    $N(s) = 0$
    for all $a \in \mathcal{A}$ do
        $N(s,a) = 0$
        $Q(s,a) = 0$
    end for
end procedure
```

Figure 1 Two Player UCT [1]

UCB1	$\bar{X}_j + \sqrt{\frac{2\ln n}{n_j}}$
UCB1-tuned	$\bar{X}_j + \sqrt{\frac{\ln n}{n_j} min(0.25, (\sigma_j + \frac{2\ln n}{n_j}))}$

Table 1 UCB For each action **j** record the average reward $\bar{X}_j$ and number of times we have tried it n_j . We write n for total number of actions we have tried. σ_j is the sample variance for each action.

Just apply the computation of values including the bonus to the process of choosing the best child of one node to traverse the tree to the leaf in MCTS.

5. AMAF

All moves as first, we can understand it literally. We can consider all the moves in the simulation as the first move to evaluate the performance of each position. However, there are also some misunderstandings during realizing the AMAF.

5.1 Failed AMAF

[4] doesn't tell the details. Then, I still wanted to figure it out by myself. So something wrong occurred. What puzzled me at first was how to combine AMAF values with standard values and UCT. For UCT takes account of the visit times to the parent and the children, I didn't know whether the visit times updated by AMAF should be considered in the process of UCT or not.

In the first version, I try both considering the AMAF visit times and not considering them in the UCT. However, in this time, I have the wrong version of MCTS. Thus, I couldn't determine which is better to choose, they are both not good as the paper says.

5.2 Successful AMAF

I corrected my program according to [1] in the same time when I corrected the MCTS.

I realized that AMAF values are not reliable when **n** is large enough. Thus, there is no need to add the AMAF visit times to **UCB** computation. However, the standard update is not reliable when **n** is small. We need always to adjust the impact of them according to **n.**

When we traversed the tree, we compute the values as:

$$(1 - \beta) * q + \beta * rq + UCB_{part}$$

β decreases from 1 to 0 as **b** increases:

$$\beta = \sqrt{\frac{k}{n * 3 + k}}$$

The value **k** is a scalar. When **k** is equal to **n**, **β** is 0.5. It means that the AMAF values and standard values take the same influences. I couldn't process so many times to determine which scalar is the best to have so many contests to evaluate the scalar. So I set k as 1200 according to

[1]. *([1] says 1000 is the best, but I considered multithread may have some influences. I increased it by a little.)*

Though [1] indicates that MC-RAVE reduces the need for explicit exploration and UCT-RAVE doesn't outperform it, I still decided to use UCT because the go always followed the opponent's moves without UCT.

6. Patterns

MoGo is a good computer go program. We thought highly of the patterns it used in MC [3] ---- **Hane**, **Cut1**, **Cut2** and patterns for movers on **Go board side**. Thus, we added it to our simulation before playing on one random position.

Actually, it made the simulation results more reliable. For our group doesn't achieve dozens of thousand simulations like other groups, it is an important part to gain more accurate results.

7. Parallelization

I used the library **<thread>.** Due to the course ***operating system concepts*** this semester, it was easy to operate on modifying the single thread to multithreads by adding some locks to the tree.

As a result, the simulations has increased 2x to 3x by 4 using threads when running.

8. Monte Carlo process acceleration

In fact, the most time consuming part is the playout. **Brown** scans the board to list all the available positions and then randomly choose one. Between two times one player play, the board only have increase 2 stones. Most of the status of them stay unchanged. Thus, it is necessary to change the method used in **brown**. What is need to be done is to detect the status of each string on the board and change the status when one player has played.

8.1 Failed try

I want to use CFG [5] to maintain the status of string. ***[Figures 2]***

I maintain all the neighboring string for each string. Each empty position was regarded as a string. Then update them when one move is made and one string are moved. Each string has one position as the core to mark the ID of the string. However, in this first try, there are too many redundant steps.

1. Mix strings with liberties (empty neighbor), it is difficult to determine whether one eye can be played when the board changes.
2. Each union of strings changes all stones of one string iteratively to follow another string's core.

There were too many complicated steps to judge the changes in the status of the eye. Thus, I gave up the CFG as the solution.

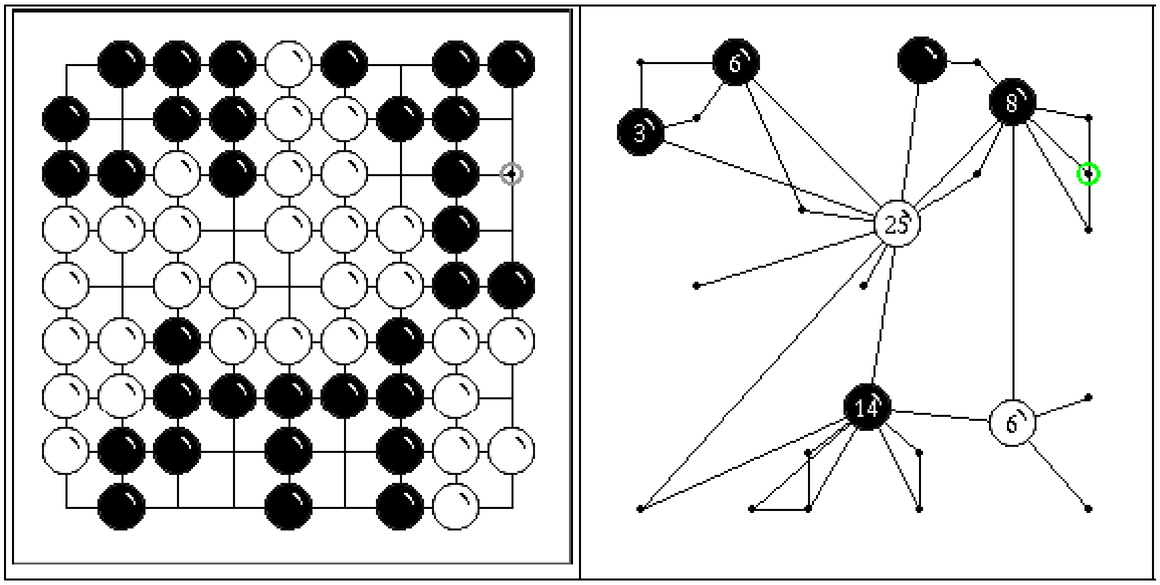


Figure 2 transformation to CFG

8.2 Second try

After the first try, I realized that I only needed to maintain the liberties rather than considering both string and liberties. I used similar methods in the first try and remembered the size of liberty. When any action changes the liberty of some string to be 1, it is noticed to check the status of the corresponding liberty and update the set of available moves for both players.

The second redundancy is dealt by using union find set to reduce it. I only need to operate on the two cores.

Then, I changed the method to generate moves in the playout. Then I gained almost 65% speed-up. For monothread, the number of simulations increase from 3000 to 5000 or so. However, it is not enough actually compared with other groups. The reason may lie in the operation of set operations which is used to maintain the positions of liberties.

9. Shortage

There are still a lot of places that can be improved.

1. The speed of simulation can be improved faster. Better data structures are needed to be used.
2. We doesn't apply any tactics except for the patterns to the program. Tactics can improve the performances especially in the beginning of the game.
3. In the beginning of the game, if the simulation is bounded to a local place, it can be more aggressive. Then if we apply some simple tactics to occupy the corner and edge, the program can be more powerful. This point is got from the contest with the champion **BaukGo** and the discussion with them after the finals.

10. Some problems in programming

I originally wanted to use dynamic arrays to represent the board, however, no matter how I ensured all the dynamic space had been deleted, there was always memory leakage increasing as the game went on. Moreover, some problems would occur in the static functions when process the dynamic arrays. Finally, I changed all the dynamic arrays to the static arrays.

I followed the above steps gradually. Thus, the parameters added to the function become more and more. They seems annoying. Finally, a Board class is used to include all the parameters.

11. Conclusion

This AI program is the program I spent the most energy on in the college life so far. I devoted more than half a month totally to it by specific reason to avoid losing the competition completely. The failures above are described with only a few lines, but they take a lot of time.

I realize something important in the process:

1. There are a lot of work useless in figuring out how to realize some ideas. Moreover, all of them should have been avoided by thinking carefully.
2. A good paper like [1] is important. However, there may exist a lot of different opinions in the world. Then what is more important compared to a good paper is **carefully thinking**. I have great improvement in self-thinking in the process. In the past, I just become irritable but do nothing in the face of difficult problems.
3. A good team is necessary for any teamwork.
4. Some small bugs can determine the whole performances. When I found that I evaluated the opposite result of the playout resulting in the poor performances that it can’t beat brown. It actually took a long time to find it and it was found accidently. When I fixed it, the performance is improved a lot. **Details determine success or failure.**
5. When it comes to indefinite things, **analyze not guess**.
6. Complicated things can be easier when I calm down.
7. A good structure is needed to be planned in the beginning. For example, if I write the class Board in the early time, a lot of time could be saved in changing the parameters for functions when I added new status parameter.

I am proud of that **teye** took the 3rd place in the final round though there may be some luck.

I tried my best in the limited time.

I found a lot of disadvantages of me.

I made a lot progress.

I have more confidence.

I am satisfied with the past fighting time. That’s enough.

Thanks.

I need to thanks for **the teacher of my lab**. He told me to read many papers early in this semester which helped me a lot in reading some papers referred in the report.

Thanks for those people believing in me.

Thanks for the teacher of the AI course to for giving this chance to me.

I do believe I can do something in the future. Thank you.

References

[1]S. Gelly, D. Silver, Monte-Carlo three search and rapid action value estimation in computer Go.

[2]L. Kocsis, C. Szepesvari, Bandit based Monte-Carlo planning.

[3]S. Gelly, Y. Wang, R. Munos, O. Teytaud, Modification of UCT with patterns in Monte-Carlo Go.

[4]D P.Helmbold, A Parker-Wood, All-Moves-As-First Heuristics in Mone-Carlo Go.

[5] T. Graepel, M. Goutri´e, M. Kr¨uger, and R. Herbrich. Learning on graphs in the game

of go. Lecture Notes in Computer Science, 2130:347–352, 2001.

[6]S. Gelly, L. Kocsis, M. Schoenauer, M. Sebag, D. Silver, C.Szepesvari, O. Teytaud, The Grand Challenge of Computer Go: Monte Carlo Tree Search and Extensions.

[7]X. Cai, D C. Wunsch II, Computer Go: A Grand Challenge to AI