Optimization of VCDTS Algorithm in Connect6 Game

Chen Zhang*[1], Hong Huang[2], Zhouyu Zhang[3], Shicong Liu[4]

1. School of Automation, Beijing Institute of Technology, Beijing, 100081, China
E-mail: gene@bit.edu.cn

2. School of Automation, Beijing Institute of Technology, Beijing, 100081, China
E-mail: honghuang@bit.edu.cn

3. School of Software Engineering, Beijing Institute of Technology, Beijing, 100081, China
E-mail: zhangzhouyu@bit.edu.cn

4. School of Information and Electronics, Beijing Institute of Technology, Beijing, 100081, China
E-mail: 1120161380@bit.edu.cn

Abstract: The game of Connect6 was invented to be viewed as a complicated game owing to its enormous search space. In this paper, an optimized approach of Victory of Continuous Double Threat Search(VCDTS) algorithm based on Aho-Corasick Automaton - called as Automaton Double Threat Search(ADTS) - was proposed. Generating moves using line patterns, ADTS has achieved ten times the efficiency of Brute-force VCDTS. Through repeated experiments, the effectiveness of the algorithm has been validated.

Key Words: Connect6, VCDTS, Line Pattern, Aho-Corasick Automaton

1 INTRODUCTION

Connect6[3] was first introduced by Prof.Wu in 2005. At the same time, Wu proposed a *Threat-based Method*[3] for Connect6, which was later on developed as *VCDTS(Victory of Continuous Double Threat Search)*[3, 7] to make continuous double threats in the game so that the attacker can dominant the game and finally win.

X6[7] is a Connect6 program, where VCDTS is a very important part, implemented by Liu. He commented that what counted most in VCDTS was accuracy while it would lead to a high time complexity since the search tree was too large. So Liu came up with the *Conservative VCDTS Algorithm*[7] which brought a higher efficiency yet a sacrifice of accuracy.

In terms of the problems mentioned above, here we announce an optimization method of VCDTS algorithm to improve the efficiency making use of line patterns and pattern matching algorithms while maintaining a precise move generation, called as *Automaton Double Threat Search(ADTS)*.

The paper is organized as follows: in section 2, we cover some preliminaries. Section 3 dives into how to optimize VCDTS and tricks in ADTS. The experimental performance of ADTS will be discussed in section 4 while we will talk about some related works at the same time. And finally we come to the conclusion in section 5.

2 PRELIMINARIES

2.1 Line Pattern

The same as Gomoku, Connect6 also has some line patterns to help player or program better understand what is going on currently and in the near future. And here we just consider the *Line Pattern*[3] could be placed with stones to make *Single Threat, Double Threats*[3], and winning steps.

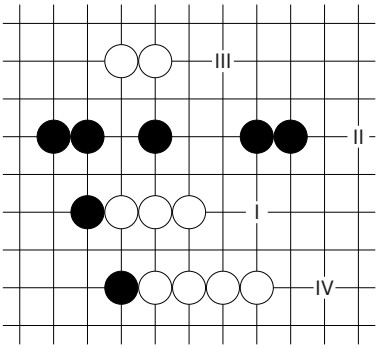


Figure 1: Line pattern.

Definition 1. *Line patterns are categorized into 4 types(See Figure 1) according to our design.*
I. *One stone to make a single threat;*
II. *One stone to make double threats;*
III. *Two stones to make double threats;*
IV. *Two stones to win(Actually contains the type that one stone to win).*

2.2 General VCDTS Algorithm

Through all these years, search tree seems to be one of the best ways to construct search algorithms in computer games. Such as *Monte Carlo Tree Search(MCTS)*[1], *Minimax Search*[2], and *Alpha-Beta Search*[2]. VCDTS is a tree search algorithm similar to Minimax Search. In VCDTS proposed by Wu, program must first come up with the positions where it could build up threats and are with considerable values evaluated by evaluation function as attacker

in the game, then it shall place our stones in the positions that are regarded as winning steps. Our VCDTS has followed the main idea, though, we have get rid of complex construction of evaluation.

As is called each time, the algorithm proceeds as follows: If the board state shows that our stone has won, then return the move we've already saved in move table that leads to such situation. Otherwise, if the search depth or the number of searched nodes exceeds pre-set limit, flag that searching fails will be returned. Under the circumstance that above all conditions are not satisfied, it will generate moves whose method shall be talked about later and continue searching recursively with modified search depth, number of searched nodes, and other information. As long as it finds more than two threats of our stone existing on the board, the path to win will be determined and the best move will be given.

2.3 Move Generation

Definition 2. *Note what we are going to raise in the next few paragraphs:*
1. *Line patterns mentioned above are referred to as* ***Patterns****;*
2. *All lines on a given board state are referred to as* ***Lines****;*

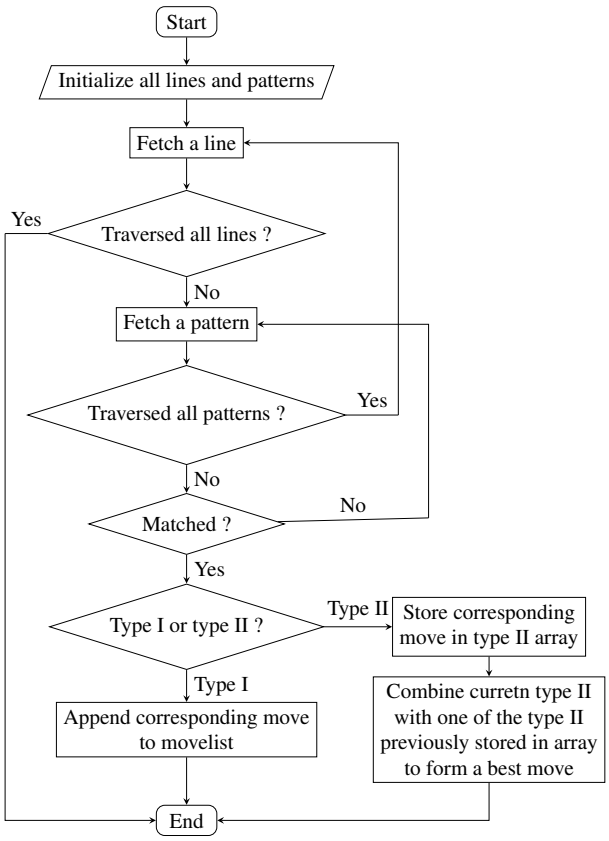
The importance of generating moves in board games is like water to fish. Here a new strategy of generating moves based on line patterns on the board was introduced, i.e. starting from the edges of the board, collecting all lines, then searching these lines with the patterns learned from experience. Once it's noted that the pattern exists in a line, the corresponding move will be appended into move list. What we need to pay attention is the probability that one pattern may be related to multiple moves although usually they works the same. However, since the crosses of lines are always significant for us to win, they must be considered differently.

Note. In this paper, only offensive move generator are analysed as defensive moves are so limited in VCDTS.

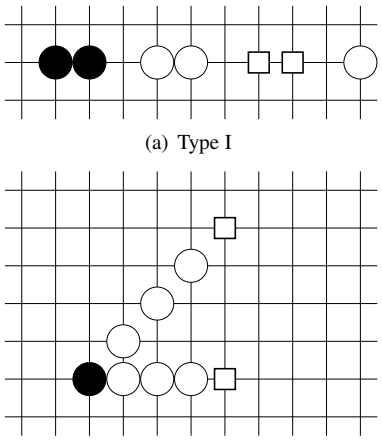
Definition 3. *Taking the line type into account, the move list could consist of:*
I. *Both two stones placed in a line pattern to make two threats;*
II. *One stone places in a line pattern to make one threat while the other one places in another pattern to make another one;*
III. *Two stones placed in a line to make two threats and one of the two deliberately makes the third threat in another line pattern.*

Lemma 1. *Moves generated by VCDTS come from recognizing patterns on the board. From these patterns VCDTS shall learn what type they are and extract moves accordingly.*

To organize this well, pattern match algorithm was set. Procedure follows.



Samples are displayed below(See Figure 2(a) 2(b)), where the grey squares represent the move according to a line pattern.



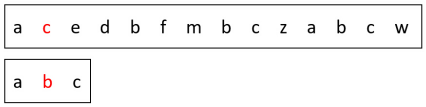
(a) Type I

(b) Type II

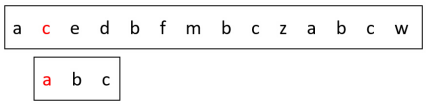
Figure 2: Move according to line pattern.

A naive way to search pattern in a line is *Brute-force Search*. Following is the main process:

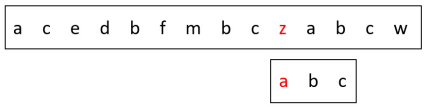
Step 1 - Align the head of **pattern** to the leftmost of **line**, and compare the first member of **pattern** with the **line**; If matched, continue with the next member of **pattern** and **line** until not matched.(See Figure 3)



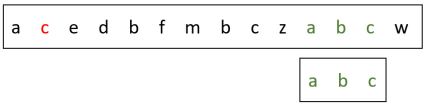
(a) Not matched



(b) Shift and restart



(c) Not matched at beginning member



(d) Finally matched

Figure 3: Brute-force Search.

Step 2 - Align the head of **pattern** to the next leftmost member of **line**, and continue with right shifting one member to be compared if matches else with a new aligning similarly. If met with a position in the **line** totally matching pattern, process stops and return success.
Step 3 - However, if the process has visited all the members in the **line**, process also stops, with a failure.

3 QUICKLY FIND PATTERNS

3.1 With Automaton

Now that the whole structure is built, the VCDTS could already solve the board state that one stone could win in the near future. Making use of Brute-force search as the core pattern matching method, VCDTS shall attempt to solve these puzzles. It's not difficult to realize there exists a problem that Brute-force method is somehow too slow because of numerous unnecessary trials.
To mitigate this problem, *Knuth-Morris-Pratt Algorithm(KMP)*[5] was presented, whose intrinsic purpose is to establish a *Deterministic Finite Automaton(DFA)*[6], namely, building automaton with pattern and using automaton to judge next state after receiving members of line one by one instead of time-wasting backing up directly. If the terminated state indicates matching, it demonstrates that the line matches pattern.
Two means could be applied to realize KMP, one is constructing partial match table and the other is based on next array. Both two are founded on the *Longest Common Prepostfix*[5]. Nevertheless, the former one is much better than the latter one in time complexity and easier to understand. Therefore, we will put emphasis on how to get a table we long for.
Definition 4. *For a pattern p, we define a auxiliary function σ named as postfix function which satisfies $\sigma(x)$ is the longest length of postfix of x one-to-one corresponding to prefix of p.*

Lemma 2. *If ϕ is a transition function, a.k.a halt function, of a given pattern based on automaton, and $t[1..n]$ is the input text of the automaton, then for $i = 0, 1, ..., n$, $\phi(t_i) = \sigma(t_i)$*
Hence, considering a pattern ABABAC, partial match table DFA could be drawn with automaton mechanism(See Figure 4).

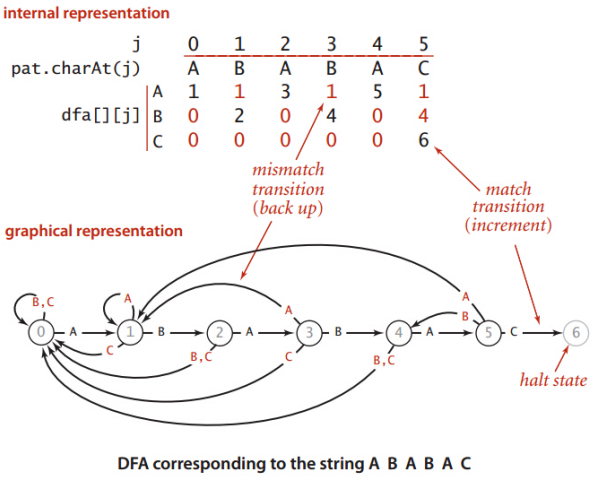


Figure 4: DFA mechanism.

Briefly, we set up a table by trying all possible inputs, during which process mismatch transition as Lemma 1 mentioned and match transition, i.e. stepping forward to next state are used.
Meanwhile, as partial state table presents above, when KMP recieves a member from the line, it will judge which state it goes to according to DFA. Take the pattern in Figure 4 for example, if currently KMP is in state 5, then it must go to state 4 after B has been received while to state 6 after receiving C, which indicates the line matches the pattern.

3.2 Automaton Double Threat Search — ADTS

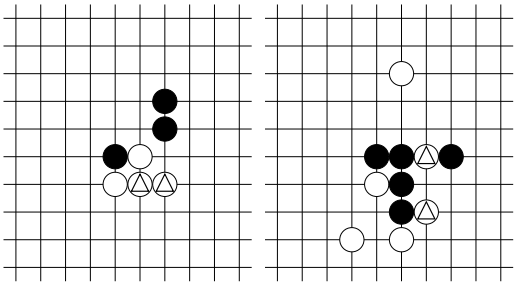
Having reached this stage, the solving time with KMP employed as the core of VCDTS is around two times the efficiency of Brute-force, say, much faster even though the time of initialization increases. Yet there do exist a modification reducing initializing time and improving search time complexity as well, in which *Trie Tree*[5] is used to integrate all patterns in a tree structure and add transition pointers between different patterns. This algorithm is called *Aho-Corasick Automaton(AC Automaton)*[5], initially designed for multi-pattern situation.
AC algorithm and DFA are distinct from one another in dealing with unmatched circumstance. Former one will pre-build a failure pointer across different patterns while later one just establishes table for only one individual pattern. To a certain degree, AC will be quicker saving many loops paid on each pattern table although it occupies more space pre-hand, which is acceptable. Just like humans will always learn and memorize a lot in his or her mind to purchase a wonderful decision in a short time when picking up a computer game.
Steps of how to build AC Automaton are given below.

(1) Insert all patterns to trie tree;
(2) Build failure pointers on the trie tree, which is based on DFA mentioned above;
(3) Given a line, search the tree with failure pointer.

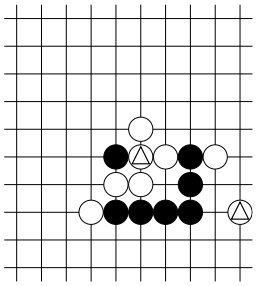
4 EXPERIMENTS

4.1 Visualizing Benefits of Optimization

Our technique for VCDTS could finally be realized. Now the average time spent by Brute-force, KMP, and AC Automaton with searching nodes increasing will be compared provided several puzzle board states could be solved by VCDTS(See Figure 5(a) 5(b) 5(c)).



(a) Puzzle 1 (b) Puzzle 2



(c) Puzzle 3

Figure 5: VCDT sample puzzles.

Figure 6 illustrates how time changes according to nodes in diagram.

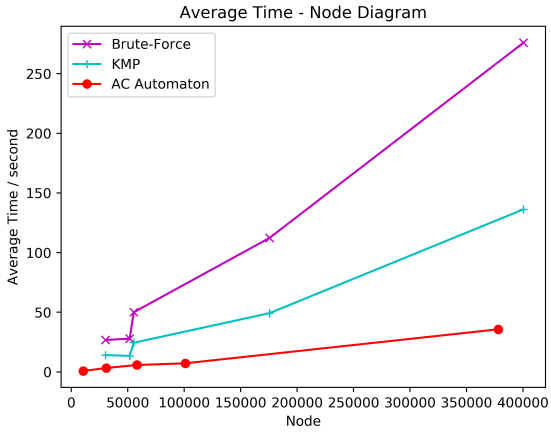


Figure 6: Time - Node Diagram.

It's clear that with nodes increasing, AC Automaton outperforms KMP and certainly Brute-force, roughly five times the efficiency of KMP and ten times of Brute-force(See Table 1).

Table 1: Average Solving Time of Sample Puzzles

Puzzle #	Brute-force	KMP	AC Automaton
Puzzle 1	50.010 s	24.449 s	5.794 s
Puzzle 2	26.753 s	14.104 s	3.306 s
Puzzle 3	275.919 s	136.268 s	35.706 s

4.2 Performance Against State-of-the-Art

There are a lot of state-of-the-art programs, some of them are accessible such as *Mobile6, X6*[4, 7]. To eliminate the difference between hardware as far as possible, programs are running on the same machine except for *Mobile6*. Our program coupled with Alpha-Beta Search wins 25% of games against *Mobile6* at level 5 and 75% of games against *X6* at level 10(See Table 2).

5 CONCLUSION

We have presented a brand new design of VCDTS algorithm - ADTS - inspired by Prof.Wu's Threat-based Method and Double Threat Search of Liu. With the help of ADTS, programs are able to find approach to win in a flexible way. Our design decomposes the problem by transferring forming threats problem to pattern matching problem and utilizing automaton to boost the efficiency and accuracy. A terrific result shows ADTS is effective.

Table 2: Game Results

Program	Black(Our)	White(Our)
Mobile6	0%	50%
X6	100%	50%

REFERENCES

[1] Cameron B. Browne, A survey of monte carlo tree search methods, IEEE Trans. on Computational Intelligence & Ai in Games, Vol.4, No.1, 1 - 43, 2012.

[2] Peter Norvig and Stuart Jonathan Russell, Artificial intelligence :a modern approach, Applied Mechanics & Materials, Vol.263, No.5, 2829 - 2833, 1995.

[3] I Chen Wu and Dei Yen Huang, A new family of k -in-a-row games, Advances in Computer Games, 180 - 194, 2005.

[4] Ji Hong Zheng and et al., Connect6 programs on mobile devices, Technologies and Applications of Artificial Intelligence, 304 - 307, 2013.

[5] T H Cormen, C E Leiserson, R L Rivest, and C. Stein, Introduction to Algorithms, Second Edition, Chap. 12, 2001.

[6] Robert Sedgwick and Wayne Kevin, Algorithms Fourth Edition, Addison-Wesley Professional, 2011.

[7] Sih-Yuan Liou, Design and implementation of computer connective 6 program x6, 2006.