The C* Algorithm for AND/OR Tree Search and its Application to a Tsume-Shogi Program

C*%'%k%4%j%:%'Kh$k AND/OR LZ$NC5:w

$*$h$S5M>-4}%w%m%0%i%'$X$N1~MQ

by

Masahiro Seo

@THx>;9(

A Master's Thesis

=$;ND@J8

Submitted to

The Graduate School of Information Science

Faculty of Science

The University of Tokyo

on February 17, 1995

in Partial Fulfillment of the Requirements

for the Degree of Master of Science

Thesis Supervisor: Hiroshi Imai :#0f9@

Title: Associated Processor of Information Science

ABSTRACT

An AND/OR graph represents the problem-solving process of reducing the given problem to more simple ones in a field of Artificial Intelligence. For the problems with enormous search space, it is important to avoid combinatorial explosions by useful heuristic functions. The AO* algorithm, which outputs the optimal solution in an AND/OR graph, is widely researched.

This paper presents the C* algorithm which solves fast the AND/OR tree search problems having a unique solution or when any solution suffices. It is based on conspiracy numbers proposed by McAllester, and solves efficiently by a human-like search method. It explores first one of the potential solution trees (pst) having the least number of unsolved tip nodes. It is constructed as an iterative deepening method by making the choice of a pst from those having the same cost and of the node to be expanded in the selected pst in depth-first order. It is enhanced with a transposition table for multiple iterative deepening, dynamic evaluation, etc. It is shown to be optimal in both time and space complexity.

We also implemented the C* as a Tsume-Shogi (the mating problem of Japanese Chess) solver whose ability is far superior to human experts.

O@J8MW;]

?M9)CNG=8&5f$N0lJ,Ln$G$"$kLdBj2r7hK*$$$F!'M?$($il?LdBj$r$$$/D+Nh$j4JC1$JItJ,LdBj$KJ,2r$7$F2r$/2aDx$O!"AND/OR%0%i%U$NC5:wH7$F%b%G%k2=$G$-$k!#KDBg$JC5:w6u4V$r;}$DLdBj$G$O!'AH9g$;GzH/rHr1k?aKM-MQ$J%R%e!<%j%9%F%#%C%/4X?t$rMQ$$$FC5:w$r8zN(2=$9k3H,=EMWG"$k!#AND/OR%0%i%U$NC5:w$G:GE,2r$r5aakLdBjKBP7FO!"AO*%'%k%4%j%:%'$,9-$/8&5f5l$F$$$k!#

K\8&5fGO!"AND/OR LZ$NC5:w$G2r$,0l0UE*$JLdBjdG0U$N2r$r5aalPh$$LdBjKBP7$F!'9bB.$K2r$r5a$a$kC*%'%k%4%j%:%'$r3+H/7?!#3l$OMcAllester$KhjDs0F5l$?6&KE?t$H$$$&35G0$K4p$E$$$F$*$j!"?M4V$N;w9M$K6a$$C5:w<jK!$G8zN(E*$K2r$r5a$akbNG$"$k!#$^$@2r1F$$$J$$@hC<@aE@?t$,>/$J$$%]%P%s%7%c%k2r7hLZ$+$iM%@hE*$KE83+$7$F$$$-!'F10l%3%9%H$N%]%P%s%7%c%k2r7hLZ$NA*Br$*$h$SF10l%]%P%s%7%c%k2r7hLZFb$NE83+@aE@$NA*Br$r?<$5M%@h$N=g=x$K7?H?I|?<2=K!G"$k!#$5iKB?=EH?I|?<2=K!$dF0E*I>2A$JIN%O%C%7%eI=$rMxMQ$7$?<jK!$Khj9bB.2=7F$$$k!#$=$7$F;~4V7W;,NL$HNN0h7W;,NLHb$K:GE,$G$"$k3H$r<($9!#

$^$?C*$r5M>-4}$r2r$/%w%m%0%i%'$H7F<BAu$7!"?M4V$N@lLg2H$rN?$02rEzG=NO$,$"k3Hr3NG'7?!#

Table of Contents

Acknowledgments

I am grateful to the members of our laboratory who have given me much beneficial advice, especially Prof. Imai Hiroshi for letting me have my own way to complete this research. I also thank others who incorporated me, Prof. Noshita Kohei at The University of Electro-Communications, Mr. Ito Takumi at NTT Software Laboratories, and Mr. Kadowaki Yoshio who is an expert of Tsume-Shogi. I am also very thankful to others who have encouraged and supported me directly or indirectly. Without it, this thesis would not have been completed.

Chapter 1

Introduction

This research developed a new search procedure, the C* algorithm, for AND/OR tree search [50]. The algorithm is based on the idea of conspiracy numbers proposed by McAllester, and it can find the solution tree of an AND/OR tree surprisingly fast. We implemented the C* as a Tsume-Shogi solver whose ability is far superior to human experts, and evaluated the performance by solving Tsume-Shogi problems, and compared the experimental results with other Tsume-Shogi programs. The search process of C* is similar to the way of human experts to solve problems.

1.1 Problem Solving by Searching AND/OR Graphs

The process of solving a given problem can be represented as searching an AND/OR graph. The AND/OR graph search has been studied by many AI researchers [7] [8] [9] [29] [31] [40] [41] [45] [52]. The AO* is a basic best-first search algorithm for AND/OR graphs [41], and some variants of AO* have been developed by others later. AO* search for a solution graph in best-first fashion, expanding the tip nodes in the potential solution graph of the minimum cost. The cost of non-terminal tip nodes is evaluated by a heuristic function which is constructed by using some application-dependent knowledge, and the efficiency of AO* depends on the exactness of the heuristic function. The AND/OR tree search using conspiracy numbers can be generalized as the AO*, but using no such application-dependent heuristic functions. The generalization of conspiracy numbers search as the AO* algorithm is described in Chapter 3.

There are many applications of AND/OR graphs (or trees) search, such as automated theorem proving, game, puzzle, etc [1] [29] [51] [52]. This paper takes

up Tsume-Shogi problems as an application of the C* algorithm for AND/OR tree search. C* is based on the conspiracy numbers [34] [49], and the algorithm using conspiracy number is inefficient for graphs, not trees. The search space of a Tsume-Shogi problem is strictly a graph, not a tree, but by regarding the nodes reached through different paths (move-sequences) as different nodes, it can be regarded as a tree. Tsume-Shogi is a good example of AND/OR tree search because of the following reasons.

1. A Tsume-Shogi problem has always a solution, and the solution is unique as far as the program is complete. If the solving program finds a solution (represented as a solution tree), then it can be terminated, and outputs the solution.

2. Tsume-Shogi has a enormous search space except trivial problems which is easy to solve, so simple brute-force search is not adequate to it because of combinatorial explosions.

3. Many Tsume-Shogi problems have been created from the Edo era. There are various types of problems containing those very hard to solve, those having a very long solution-sequence, and those give artistic impressions to human solvers, etc.

4. Many Tsume-Shogi solving programs of high performance have been implemented till today, so the performance of a program can be measured by solving well-known Tsume-Shogi problems and comparing the ratio of solved programs or the CPU time taken to find the solutions with other programs.

5. Many people have been enjoying Tsume-Shogi, so the result of Tsume-Shogi programs have an impact upon the world.

1.2 Background

Tsume-Shogi is deviated from the endgame of Shogi (Japanese chess, or Sashi-Shogi). So the algorithm of Tsume-Shogi is relevant to that of Shogi or chess. This section surveys briefly the history of the research of computer chess, computer Shogi, and computer Tsume-Shogi.

Computer chess has been studied by many competent researchers of the world till today [2] [4] [5] [6] [12] [19] [20] [30] [39]. Chess can not be solved mathematically, because it has enormous search space. Instead of searching to the end of the game, one searches to a fixed depth, and evaluates the position statically by using chess specific knowledge. At first, forward pruning method is used to increase the search depth. Chess specific knowledge is considered to be important to select the moves to be selectively searched and evaluate positions exactly.

According to the progress of hardware technology, brute-force search (searching all legal moves at each position) becomes to be used among the most chess programs [30]. Most chess programs use an (iterative deepening) alpha-beta search method. Alpha-beta guarantees the same result as minimax, but more effective by pruning nodes unrelated to the root value [21]. The branching factor of chess is about 35, and super-computers or parallel computers for chess specific use are developed, so can be searched about ten depths or more in real time. Variable depth search such as quiescence search and singular extensions are developed in 1980's [2] [3] [19]. Singular extensions are used by chess programs such as DeepThought [2]. The ability of the strongest computer chess programs such as DeepBlue (the successor machine of DeepThought) are the almost the same level as the human chess champion of the world. The computer software Chess Genius II defeated the world chess champion Kasparov at August 1994.

In Japan, computer Shogi has been studied more and more recently by Matsubara, Iida, and others [14] [24] [32]. However the branching factor of Shogi is about 80 [33], so it is impossible to search to adequate depth by brute-force methods like computer chess. Knowledge-oriented approach such as forward pruning or constructing complicate static evaluation functions using domain-dependent knowledge are tried [24]. It is needed to search move-sequences which are important to decide the best move on a current position like human players.

Tsume-Shogi is the mating problem of Japanese Chess. It has usually one solution, and the problem is to find the solution. Tsume-Shogi solving programs have been developed by Morita, Kakinoki, Takada, and others [42]. Till 1990, all Tsume-Shogi programs could not solve the problems whose solution length is longer than 13 because they used the (iterative deepening) depth-first search, so there occurs combinatorial explosions against the Tsume-Shogi problems having long solution-sequences.

Noshita developed the program named T2 in 1992 [42]. T2 uses hashing to detect the same positions, and various Tsume-Shogi specific devices. T2 can solve almost problems whose solution length is less than or equal to 25. However T2 can not solve almost all problems whose solution length is larger than 30.

Ito developed the Tsume-Shogi program named Ito [15]. The algorithm used by Ito is best-first search, and some problems having long solution-sequences can be solved. The heuristic function used by Ito is the number of legal moves to escape from checks (called Oute). The main defect of Ito's program is that his algorithm is a kind of best-first search, and it must store all the generated nodes in memory, so it cannot solve the difficult problems unsolvable within relatively short time because of running out of memory. Recently also Noshita and Kawano developed competent Tsume-Shogi solving programs using best-first search.

The branching factor of Tsume-Shogi has never be calculated yet. The numerical results of this paper shows that the branching factor of Tsume-Shogi is about 5. The number is much smaller than that of Shogi or Chess, but the problems whose solution are longer than 17 can not be solved by simple brute-force methods such as alpha-beta. Also selective search can not be used to solve the problems always correctly.

Other researches on the artistic aspects of Tsume-Shogi exists. Koyama and Kawano constructed a database of Tsume-Shogi problems containing almost all problems created so far, and analized the impression of Tsume-Shogi problems quantitatively [25] [26] [27].

1.3 Conspiracy Numbers

The concept of conspiracy numbers is recently proposed by McAllester [34], and later implemented as chess playing program and its performance is experimented [49]. Conspiracy numbers are the measure of the accuracy and the stability of the root value of a minimax tree, that is, the minimal number of the set of terminal nodes of the minimax tree which must conspire (change their evaluated values) to change the minimax value of the root to other values. Larger the conspiracy numbers of the root value becomes, more reliable the minimax value of the root becomes.

The terminal values evaluated in the minimax trees are not always accurate,

because the static evaluation function only projecting the degree of advantage or disadvantage of one side at the position to a value by synthesizing and analyzing the static information of the position. So the error of the terminal evaluated values are amplified when backing up the minimax values, and the minimax value of the root may be incorrect. But as the conspiracy number of the root becomes larger, the probability of such incorrectness of minimax value decreases.

The search procedure using conspiracy numbers continues to grow the search tree by expanding the node which seems to have the greatest effect on the minimax value of the root node, till the conspiracy numbers to change the root value to another values (usually plus 1 and minus 1 to the current minimax value) becomes greater than a threshold conspiracy number.

The conspiracy number search can be applied to AND/OR tree search [1] [11]. The purpose of AND/OR tree search is to find a solution (in the form of a solution tree) to solve the root node (corresponding to the given problem) or to prove the root value. The conspiracy number to solve the root node in AND/OR tree is the same as the minimum number of unsolved tip nodes of all the unsolved potential solution trees. The search procedure proceeds by selecting the potential solution tree of minimum conspiracy number and expanding a tip node in the selected potential solution tree. This process is similar to the problem solving methods of human experts, searching selectively the moves (or subtrees) whose probabilities of being in the solution tree are expected to be greater than other parts of the search space.

1.4 The C* Algorithm

This paper developed the C* algorithm for AND/OR tree search based on conspiracy numbers and implemented it as a Tsume-Shogi solving program.

Among the existing Tsume-Shogi programs, those using best-first search needs a lot of memory and can not solve difficult problems which require much computational time, because of running out of memory [15]. Also Tsume-Shogi programs using depth-first search like T2, it takes much more time to solve the problems whose solution length becomes greater [42]. This research developed the program that can solve almost all Tsume-Shogi problems containing those having very long solutions.

The C* is a depth-first iterative-deepening search algorithm, so the overhead of reexpansions of nodes already expanded at previous iterations may be seemed to be high. The overhead of reexpansions can be reduced by hashing, the computational results of nodes generated and expanded at previous iterations are stored in a hash table. It needs only linear space, so never running out of memory. We make the C* more effective by such methods as multiple iterative deepening and dynamic evaluation by hashing, etc.

The multiple iterative deepening method does the iterative deepening search not only at the root node but also at all other OR nodes. If the multiple iterative deepening is used, it may seem that the number of reenpanded nodes becomes greater. But by changing the ordering of the successor OR nodes at each iteration when an AND node is expanded, we can reduce the ratio of the reexpansions. The numerical results in chapter 5 shows that the ratio of reexpanded nodes to total expanded nodes is about 20%.

When an AND node is expanded and its successor OR nodes are generated, the dynamic evaluation by hashing counts the conspiracy number of unsolved successor tip node as N instead of 1, if the successor OR node has been expanded at previous iterations, and the information that it could not be solved within conspiracy number N is stored in the hash table.

By implementing both the multiple iterative deepening and dynamic evaluation by hashing, the performance of Tsume-Shogi problems is improved, compared with the cases that either of the two is implemented, or neither of the two are implemented.

1.5 The C*'s Main Results

The contents of experiments in Chapter 5 is that we measured the CPU time for our Tsume-Shogi solving program required to solve the selected Tsume-Shogi problems. As a benchmark set of problems, we selected the book named "Zoku-Tsumuya-Tsumazaruya" in which about 200 Tsume-Shogi problems are printed [18]. The book contains various types of problems, from 11-move (called 11-TE-Tsume, which means that the length of solution-sequence is 11) to 873-move, created by 41 creators from the Edo era to the Showa era. The experimental results of other programs to the problems in this book are as follows. The program named T2

solved about 70 problems, and the program named Ito solved about 135 problems of all about 200 problems in this book [15] [23].

Our programs solved 190 programs in the book "Zoku-Tsumuya-Tsumazaruya" within 2 hours per a program on Sun Sparc Station 20. Almost all of the unsolved problems are those also seems to be difficult for human experts to solve, judging from the solution-sequences and the remarks of these problems in the book. Our program solved two problems having very long solution-sequences (611-move and 873-move) within 2 hours. The 873-move is the longest solution-sequence of all programs solved by computer [16].

We also tried to solve 100 programs in the book "Shogi-Zuko" which is well-known and have been solved by many people [17]. Our program solved 99 problems in the book.

Chapter 2

The Relation between Searching AND/OR Trees and Solving Tsume-Shogi Problems

The subproblems decomposed by problem reduction techniques in various areas in artificial intelligence are all have to be solved (at an AND node) or exactly one of them needs to be solved (at an OR node) in general [40] [41]. Tsume-Shogi is a typical example of a problem-solving process, and a kind of one-player puzzles, originated from the endgame of Shogi. The goal of Tsume-Shogi is to find the move-sequence of Black (attack) side to mate the King of White (defense) side. They usually not contain the Black's King on the board, except those called "Sou-Gyoku" problems. Tsume-Shogi problems always have only one solution, if they are complete, and can not be solved by simple brute-force searching techniques in real time except easy problems, so it is a good example to be researched in AI.

2.1 AND/OR trees and Tsume-Shogi

The search space of a Tsume-Shogi problem can be seen as an AND/OR tree. The process of solving a Tsume-Shogi problem corresponds to the process of searching an AND/OR tree. Moves at a position can be regarded as reducing a problem to more simple problems. The given problem corresponds to a root OR node. The successor nodes of an OR node are AND node, and the successor nodes of an OR node are AND nodes. The Black to move position corresponds to an OR node, and the White to move position corresponds to an AND node. All legal moves of Black side at a position correspond to the edges between the OR node and all its

Tsume-Shogi	an AND/OR tree
a given initial position	a root node (OR node)
a Black to move position	an OR node
a White to move position	an AND node
a mated position	an AND node having no successors
a Black to move position having no checking moves	an OR node having no successors
a position not searched until a mated position yet	an unsolved position

Table 2.1: The correspondences between Tsume-Shogi and AND/OR tree.

successor AND nodes, and all legal moves of White side at a position correspond to the edges between the AND node and all its successor OR nodes. An AND node is solved if it has no successors, and an OR node having no successors is unsolvable. An internal OR node is solved if any one of its successors is solved, and it is unsolvable if all its successors are unsolvable. An internal AND node is solved if all of its successors are solved, and it is unsolvable if any of its successors are unsolvable. A node which is neither solved nor unsolvable is called unsolved.

Definition 1 *A potential solution tree pst in an AND/OR tree is a subtree such that,*

1. *A root node of the tree is in pst.*
2. *If a nonterminal OR node is in pst, then exactly one successor node of it is in pst.*
3. *If a nonterminal AND node is in pst, then all its successor nodes are in pst.*

The above definition suggests that to solve an internal OR node, one needs to solve only one successor node, and to solve an internal AND node, one must solve all its successor nodes. An unexpanded node of *pst* is called a "tip node". When the root node of potential solution tree *pst* is unsolved, the *pst* would become either solved or unsolvable by successively expanding the pst's tip nodes. If all the tip

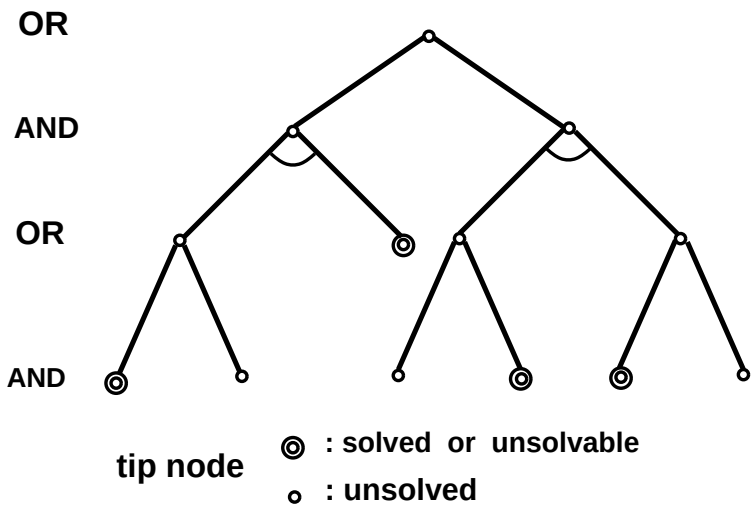


Figure 2.1: An example of an AND/OR tree.

nodes of an unsolved *pst* are solved, then the *pst* becomes a solution tree. The definition of a "solution tree" is as follows.

Definition 2 *If all the terminal AND nodes of a potential solution tree pst are solved, then the root node of the pst is solved, and the pst becomes a solution tree.*

The search process terminates when one solution tree of an AND/OR tree is found. The main theme of this research is to find the solution tree as far as fast.

An AND/OR tree representing the decomposition of a given Tsume-Shogi problem may be seemed to be a special case of searching an AND/OR tree, because the parent of an OR node is always an AND node, and the parent of an AND node is always an OR node. Any AND/OR tree can be transformed to the AND/OR tree of this type equivalent to the original one. So Tsume-Shogi problems are general examples of AND/OR tree search, and the algorithm of Tsume-Shogi can be applied to other areas of AI.

2.2 About Tsume-Shogi

The answer of a Tsume-Shogi problem is normally given as a solution-sequence of moves, namely, a path of move-sequence from the given root node to a mated node. The answer is the longest path of all paths from the root to the terminal AND nodes in a solution tree. The defense side (White) have to make the best

defense to live as long as possible. The solution-sequence of moves satisfies the following conditions[32]:

1. every move of the attack side (Black always moves first) must be check, and the attack side must mate with the shortest sequence.

2. the defense side (White) must select his move which makes the mating sequence as long as possible, but he can not take useless-moves ('Muda-ai').

So the solution-sequence corresponds to the sequence of best moves of both sides. The following characteristic of Tsume-Shogi are different from that of mating problems of chess [32].

1. Every piece not otherwise accounted for, except the Black's King, is in White's hand. White can drop these pieces on the board in his turn.

2. The length of a correct solution-sequence is not given in advance.

The conditions of completeness of Tsume-Shogi problems is defined as follows [32].

1. A problem that has no solution-sequence (that is, it is never mated) is disqualified (called "no-mate" or "Fu-Tsume").

2. There is only one solution-sequence of moves in the sense that:

   (a) if the attack side selects other moves, he will fail to mate the problem.

   (b) if the defense side selects other moves, the attack side can mate more easily (that is, with a shorter sequence or leaving some pieces in hand unused).

   (c) According to the solution-sequence, the attack side has no pieces in hand at the last (mated) situation.

The solution-sequence of a complete Tsume-Shogi problem is unique, but even the complete problem usually has several solution trees. The longest move-sequence shared by such solution trees is the solution-sequence intended by the creator of the problem. If there exists no solution-sequences shared by all the solution trees, then the problem has more than one solution-sequence, called "Yo-Tsume", regarded as an incomplete problem. If a program finds a solution tree not intended by

the creator, then it may answer another move-sequence different from the solution-sequence, called "Henka-Betsu-Tsume". The creator of Tsume-Shogi problem must investigate all possibilities of Yo-Tsume to confirm the completeness of the problem after creating the problem. So Creating a complete Tsume-Shogi problem is much more difficult in general than solving the same problem correctly.

2.3 Tsume-Shogi Programs

Tsume-Shogi programs have been developed by Morita, Kakinoki, Takada, and others in 1980's [42]. Till 1990, all Tsume-Shogi programs have not been able to solve almost all of the problems whose solution length is greater than 13-move in real time. The reason is that they used (iterative deepening) depth-first search, so combinatorial explosions occurs against the Tsume-Shogi problems having long solution-sequences.

Noshita developed the program named T1 in 1991 [42], and T1 was improved and developed to T2 in 1992 [15]. T2 uses hashing to detect the same positions, and various Tsume-Shogi specific devices. The algorithm used by these programs is the so-called depth-first iterative deepening. It does depth-first search of depth 1 for 1-move mate, and if no solutions are founds, then does depth-first search of depth 3 for 3-move mate, and if no solutions are founds, then does depth-first search of depth 5 for 5-move mate, and so on. T2 can solve almost all problems whose solution length is less than or equal to 25. T2 is very strong against Tsume-Shogi programs of relatively short solution-sequence. Its performance would be highest among all programs using (iterative deepening) depth-first search. However T2 can not solve almost all problems whose solution length is larger than 30, because it takes exponential time of the length of the solution-sequence.

Ito developed a Tsume-Shogi solving program named Ito in 1992 [15]. Before the publication of his program, all Tsume-Shogi programs used depth-first iterative deepening, so the problems having long solution sequences can not be solved within time constraint because of combinatorial explosions. However, Ito uses best-first search, selectively expanding the tip nodes of the potential solution tree whose probability of becoming a solution tree is the highest. It can be said that his algorithm is more similar to the solving process of human experts than other programs.

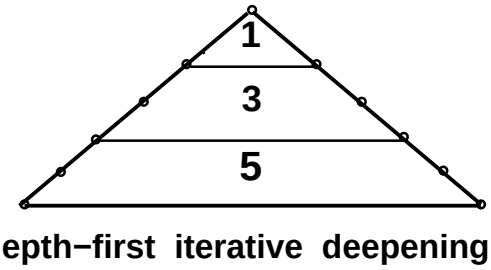


Figure 2.2: Depth-first iterative deepening.

The main disadvantage of Ito's algorithm is its memory usage. All the nodes generated must be stored in memory. Memory space is running out before combinatorial explosion in time. When running out of memory, it may be possible to continue the search by selecting the removed nodes seemed to be useless, but it has a risk of never finding a solution forever when the node in a solution tree is removed unfortunately.

We aimed to develop a Tsume-Shogi program which overcomes both T2's and Ito's defects and combines both T2's and Ito's advantages.

Chapter 3

Conspiracy Numbers

Recently the branching factor of Shogi (Sashi-Shogi) is calculated by the analysis of games between human professional players, and it turned out to be about 80 [33]. Branching factor means the average number of successor nodes in the whole search tree. We calculated the branching factor of Tsume-Shogi by averaging the number of successor nodes of all expanded nodes of Tsume-Shogi problems in the set "Zoku-Tsumuya-Tsumazaruya" (See Chapter 5), and it turn out to be about 5. The branching factor 5 of Tsume-Shogi is much smaller than that of Shogi, so you may feel that solving Tsume-Shogi problems is much easier than winning at Shogi.

However, to solve a Tsume-Shogi program, you must search at least all nodes in a solution tree, and the length of the solution-sequence becomes often to be greater than twenty or sometimes some hundreds. For example, the length of the longest solution-sequence of problems solved by our program is 873-move, so to put it simply we must generate $5^{873} \simeq 10^{600}$ nodes. Except trivial programs, Tsume-Shogi has enormous search space, so it is impossible for simple brute-force algorithms to solve almost all programs in real time (within a few hours).

So it needs to solve problems effectively like human experts. We turned our attention to the concept of conspiracy numbers proposed by McAllester [34]. It becomes possible to identify the potential solution tree whose probability of becoming a solution tree is the highest of all pst's.

3.1 The Conspiracy Numbers for Minimax Tree Search

The difficult games such as chess, Go, and Shogi can not be solved mathematically. It is impossible to search to the end of the game, so the problem that the initial position of the game is whether black to win or white to win or draw has never been solved, or at least it can never be proved. So the unsolved positions of some depths are virtually regarded as terminal nodes, and evaluated by a static evaluation function created according to some application-dependent knowledge of human experts. The terminal nodes in minimax trees correspond to the tip nodes in AND/OR trees. The player MAX selects the move reaching the successor node of the maximum evaluated value, and the player MIN selects the move reaching the successor node of the minimum evaluated value. The minimax value of the root can be determined by minimax back-up procedure. The root value is the same as the static evaluation value of the terminal node reached from the root through the sequence of best moves of both sides.

A static evaluation function only projects the degree of advantage or disadvantage of either side at the position to a value by synthesizing and analyzing the static information of the position. So static evaluation functions are difficult to evaluate terminal positions always accurately in games seemed to be difficult in general, such as chess, GO, and Shogi. Though alpha-beta search always calculates the accurate value of a minimax tree, the errors of the evaluation of terminal values are amplified more and more by searching deeper. Reaching deeper consistently degrades the quality of a minimax value. The pathology was discovered by Nau and Beal independently, and well-known as the theoretical nature of game trees [35][36] [37] [44] [45]. The deeper we search, the worse we play.

Conspiracy numbers search is proposed by McAllester [34]. The conspiracy number is a measure of the accuracy of the root value of a minimax tree. It aimed to search until changing the root value becomes unlikely. It is the minimal number of terminal nodes to be changed needed to change the root value to other values. The root value of a minimax tree becomes more stable by searching selectively the subtree rooted at the node which is one of the conspirators for the root value.

In situations that a player A forces a sequence of moves to a player B, that is, there are only few choise of a move in each positions of the sequence to avoid clear or immediate disadvantage. The static evaluation function of the reached position

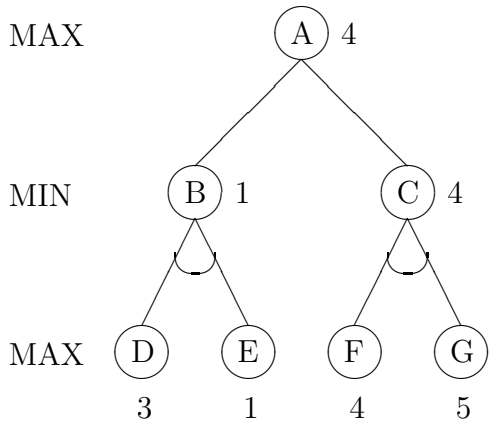


Figure 3.1: A minimax tree whose root value is 4.

"- CN(n,v)	0	1	2	3	4	5	6	", CN(n,v)	0	1	2	3	4	5	6
A	2	1	1	1	0	0	0	A	0	0	0	0	0	1	2
B	1	0	0	0	0	0	0	B	0	0	1	1	2	2	2
C	1	1	1	1	0	0	0	C	0	0	0	0	0	1	2
D	1	1	1	0	0	0	0	D	0	0	0	0	1	1	1
E	1	0	0	0	0	0	0	E	0	0	1	1	1	1	1
F	1	1	1	1	0	0	0	F	0	0	0	0	0	1	1
G	1	1	1	1	1	0	0	G	0	0	0	0	0	0	1

Table 3.1: Conspiracy numbers of nodes in Figure 3.1.

P as a result of these forced move-sequences may be good for the player B, but the exactness of it is not guaranteed. If the position P is searched deeper, then it may turns out to be bad for the player B, and the probability of varying the root value also to be bad for the player B is relatively higher. In such a case, searching the position P more deeper is important to decide the best move at the current position. Conspiracy numbers is a mesure of stability or reliability of minimax values to detect such critical positions.

The notation of conspiracy numbers for minimax tree search is defined as follows. "- $CN(n, v)$ denotes the minimum number of terminal nodes whose value must be decreased to change the minimax value of the node n to the value less than or equal to v. ", $CN(n, v)$ denotes the minimum number of terminal nodes whose value must be increased to change the minimax value of the node n to the

value greater than or equal to v. Table 3.1 shows the conspiracy numbers of nodes in the AND/OR tree of Figure 3.1. For example, the Table 3.1 shows that "-$CN(A, 0) = 2$ which means that to decrease the minimax value of node A to 0, one must decrease the values of two terminal nodes to 0 (or less than 0). In this example, the conspirators of "-$CN(A, 0)$ is [(D *or* E) *and* (F *or* G)].

The conspiracy number to change the minimax value m of node n to value v is calculated as follows.

1. If n is a terminal node, and

   (a) if m is a game theoretical value of node n and can never be changed, then

   $$\text{"-CN}(n, v) = \infty \qquad \forall\, v < m$$
   $$\text{"-CN}(n, v) = 0 \qquad \forall\, v \geq m$$
   $$\text{",CN}(n, v) = \infty \qquad \forall\, v > m$$
   $$\text{",CN}(n, v) = 0 \qquad \forall\, v \leq m$$

   (b) if m is a value of node n evaluated by a static evaluation function and can be changed by searching deeper, then

   $$\text{"-CN}(n, v) = 1 \qquad \forall\, v < m$$
   $$\text{"-CN}(n, v) = 0 \qquad \forall\, v \geq m$$
   $$\text{",CN}(n, v) = 1 \qquad \forall\, v > m$$
   $$\text{",CN}(n, v) = 0 \qquad \forall\, v \leq m$$

2. If n is a nonterminal MAX node whose successor nodes are $n_i, 1 \leq i \leq k$, then

   $$\text{"-CN}(n, v) = \sum_{1 \leq i \leq k} \text{"-CN}(n_i, v)$$
   $$\text{",CN}(n, v) = \min_{i \leq i \leq k} \text{",CN}(n_i, v)$$

3. If n is a nonterminal MIN node whose successor nodes are $n_i, 1 \leq i \leq k$, then

   $$\text{"-CN}(n, v) = \min_{1 \leq i \leq k} \text{"-CN}(n_i, v)$$
   $$\text{",CN}(n, v) = \sum_{1 \leq i \leq k} \text{",CN}(n_i, v)$$

From the Monotonicity Theorem of comspiracy numbers in [34], the following relations holds.

$$\text{"-CN}(n, v_1) \geq \text{"-CN}(n, v_2) \qquad \forall\, v_1 < v_2$$

", $\mathrm{CN}(n, v_1) \leq$ ", $\mathrm{CN}(n, v_2) \quad \forall\, v_1 < v_2$

An implementation of conspiracy numbers in computer chess is presented and some experiments are shown by Schaeffer [49]. There are some defects in searching minimax trees using conspiracy numbers. The first is its memory requirement. A conspiracy numbers search is a kind of best-first search, and required to store all the nodes generated previously and also their conspiracy numbers, so difficult positions cannot be searched to reliable conspiracy numbers because of running out of memory. The second is that the values of decomposed nodes are assumed to be independent, but the assumption does not hold in most of its applications such as chess, GO, and Shogi. And also the branching factor of these games is relatively greater, so the conspiracy numbers may not reflect accurately the stability or reliability of minimax values.

3.2 The Conspiracy Number for AND/OR Tree Search

The conspiracy numbers for minimax tree search can be applied to the algorithm on searching AND/OR trees [1] [11]. The conspiracy number for AND/OR tree search is the minimum number of unsolved tip nodes to be solved in order to solve the root node. It can be said that greater the conspiracy number of the root to be solved, more difficult it becomes to solve the root node. We call the set of unsolved tip nodes which constitute the minimum conspiracy number "conspirators". The conspirators correspond to the tip nodes of the potential solution tree of the minimum conspiracy number at the root node. The search proceeds by continuously expanding the conspirators.

The conspiracy number of an unsolved potential solution tree and that of an unsolved node are defined as follows.

Definition 3 *In an AND/OR tree, the conspiracy number of an unsolved potential solution tree pst is equal to the number of unsolved tip nodes in pst.*

Definition 4 *In an AND/OR tree, the conspiracy number of an unsolved node n is equal to the minimum conspiracy number of potential solution trees rooted at n.*

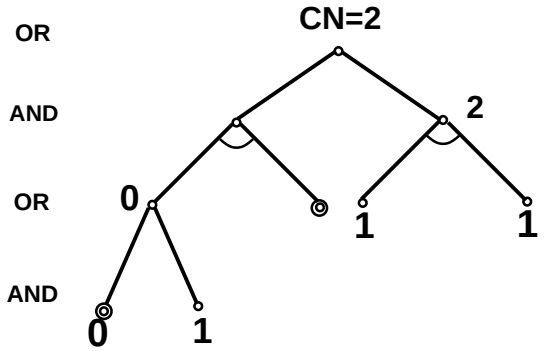


Figure 3.2: The conspiracy numbers of nodes in an AND/OR tree.

$CN(n)$ denotes the conspiracy number of a node n in an AND/OR tree. The calculation method of the conspiracy number to solve a node n in an AND/OR tree is as follows.

1. If n is a tip node, then

   (a) If n is a solved AND node, then
   $\mathrm{CN}(n) = 0$

   (b) Else if n is an unsolvable OR node, then
   $\mathrm{CN}(n) = \infty$

   (c) Else n is an unsolved tip node, and
   $\mathrm{CN}(n) = 1$

2. Else if n is an OR node whose successor nodes are $n_i, 1 \leq i \leq k$, then
   $\mathrm{CN}(n) = \min\limits_{1 \leq i \leq k} \mathrm{CN}(n_i)$

3. Else if n is an AND node whose successor nodes are $n_i, 1 \leq i \leq k$, then
   $\mathrm{CN}(n) = \sum\limits_{1 \leq i \leq k} \mathrm{CN}(n_i)$

3.3 The Generalization of Conspiracy Number Search for an AND/OR Tree as the AO* Algorithm

The AO* algorithm has been widely researched till today [7] [8] [9] [29] [40] [41]. The AO* is a best-first search procedure for AND/OR graph (or tree) search, it is named by Nilsson[41], and can be generalized as a branch and bound procedure

for searching an optimal solution tree [28] [38]. The A* is an algorithm to find an optimal solution path in an OR tree, so A* can be seen as a special case of AO* [40] [41]. AO* is admissible in the same reason as A*, that is, AO* always terminates and outputs an optimal solution, if there is a heuristic function for estimating the cost of the node which never overestimates the true cost of the solution rooted at the node. We define the heuristic function for estimating the cost of a node n as $h(n)$, and the cost function of a node n as $f(n)$. It is assumed in general that each edge from a node m to a node n has a finite non-negative cost $c(m, n)$. $f(n)$ is recursively calculated as follows [40] [41].

1. If n is a tip node, then

   (a) If n is a terminal AND node, then
   $$f(n) = 0$$

   (b) Else if n is a terminal OR node, then
   $$f(n) = \infty$$

   (c) Else if n is a nonterminal node, then
   $$f(n) = h(n)$$

2. Else if n is a nonterminal OR node whose successor nodes are $n_i, 1 \leq i \leq k$, then
   $$f(n) = \min_{1 \leq i \leq k} \{f(n_i) + c(n, n_i)\}$$

3. Else if n is a nonterminal AND node whose successor nodes are $n_i, 1 \leq i \leq k$, then
   $$f(n) = \sum_{1 \leq i \leq k} \{f(n_i) + c(n, n_i)\}$$

The cost $f(n)$ defined above corresponds to the cost of the potential solution tree of the minimum cost rooted at n. When all tip nodes in a potential solution tree become terminal nodes by expanded and searched deeper, $f(n)$ becomes the true cost of the solution tree.

In the above definition, the AND/OR trees are assumed to be additive AND/OR trees, but according to applications, the $\sum_{1 \leq i \leq k}$ in the definition of $f(n)$ at a nonterminal AND node is replaced with $\max_{1 \leq i \leq k}$ [40] [41].

The algorithm using conspiracy numbers for searching an AND/OR tree can be generalized as the AO*. The conspiracy number for AND/OR tree search can

be defined as a function $f(n)$ by adding some modification to the above definition of $f(n)$. Here the term "cost" in the above definition means a comspiracy number. The conspiracy number $CN(n)$ for a node n to be solved in an AND/OR tree is calculated as follows (See Section 3.2).

1. If n is a tip node, then

   (a) If n is a solved AND node, then
   $\mathrm{CN}(n) = 0$

   (b) Else if n is an unsolvable OR node, then
   $\mathrm{CN}(n) = \infty$

   (c) Else n is an unsolved tip node, and
   $\mathrm{CN}(n) = 1$

2. Else if n is an OR node whose successor nodes are $n_i, 1 \leq i \leq k$, then
   $\mathrm{CN}(n) = \min_{1 \leq i \leq k} \mathrm{CN}(n_i)$

3. Else if n is an AND node whose successor nodes are $n_i, 1 \leq i \leq k$, then
   $\mathrm{CN}(n) = \sum_{1 \leq i \leq k} \mathrm{CN}(n_i)$

In the above calculation procedure of conspiracy numbers, the cost of nonterminal tip node is always estimated as 1. So it can be said that the conspiracy number uses no heuristic function constructed by using some application-dependent knowledge, instead it assigns the constant value 1 to all the unsolved tip nodes.

Also in the above procedure, the cost of an edge is defined to be 0, and every unsolved tip node is assigned the cost 1. But it is possible to give some cost, 1 for example, to each edge [11]. The merit of this is that the termination of the algorithm is guaranteed, without searching a long path forever. But our implementation of a Tsume-Shogi program, the cost of an edge is defined as 0 to avoid being restricted by search depths. And the C* is depth-first iterative deepening algorithm, so the following merit of making the cost of an edge 0 exists. We can decrease the waste of the calculation of reexpanded nodes, because the number of expanded nodes at each iteration increases.

Chapter 4

The C* Algorithm for AND/OR Tree Search

We developed the C* algorithm for AND/OR tree search based on conspiracy numbers. The structure of C* algorithm described in this chapter is showed roughly in Figure 4.1. Here we assume some points about problems.

1. The goal is to find a solution tree, and the search process can be terminated when finding any solution tree. At least one solution tree always exist in the search space.

2. The successor nodes of an OR node is type AND, and the successor nodes of an AND node is type OR, if any.

3. An AND node having no successors is a solved node, and an OR node having no successors is an unsolvable node, and other tip nodes never expanded before are unsolved nodes.

The above assumptions are general, not application-dependent.

4.1 Depth-first Iterative Deepening using Conspiracy Number

There are two simple search algorithm, depth-first search, and breadth-first search.

Depth-first search expands the most recently generated node first. It needs only linear space, but takes a lot of time unless the ordering of successor nodes is fortunately optimal. The depth where the solution exists is not known in advance,

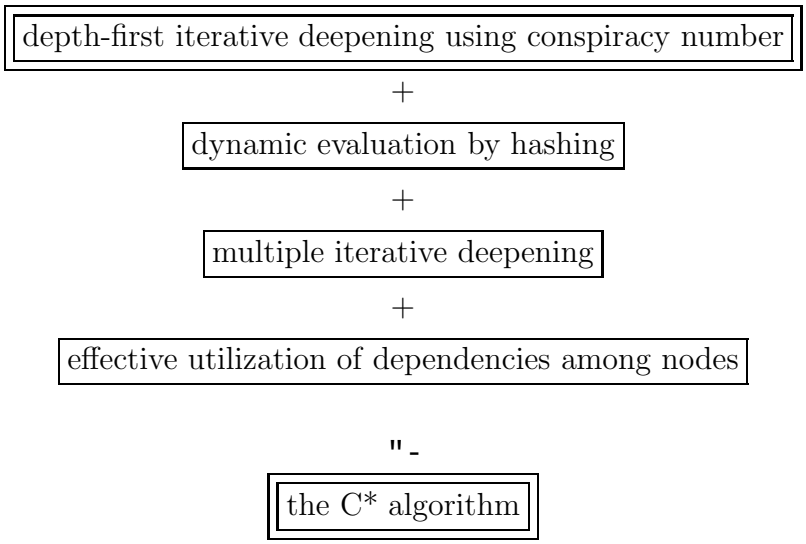


Figure 4.1: The structure of the C* algorithm.

so we must estimate it before starting depth-first search. If the estimate is lower than the depth of an optimal solution, the algorithm terminates without finding any solutions. If the estimate is much larger than the optimal solution's depth and the ordering or successor nodes turned out to be bad, it takes a lot of time than that needed by other search methods. The selection of the depth bound is hard to be predetermined, and the ordering of successor nodes is important for its efficiency.

Breadth-first expands the most previously generated node first. All nodes at depth less than or equal to d have been already generated when any nodes at depth $d+1$ are generated. All the generated nodes must be kept in memory in order to generate nodes of greater depth. It takes a lot of space, and if the depth of the solution is large, its time complexity also becomes large. So breadth-first search is not used in many applications today.

The iterative deepening search suffers neither defects of depth-first search nor breadth-first search [10] [22]. It does the depth-first search to depth 1 at the first iteration, and if a solution does not found at the first iteration, then search to depth 2 at the next iteration, and so on. The all nodes generated at the previous iterations are discarded. Depth-first iterative deepening is shown to be optimal by Korf [22], and it can be enhanced by hash tables, killer heuristic, history heuristic, and so on [46] [48].

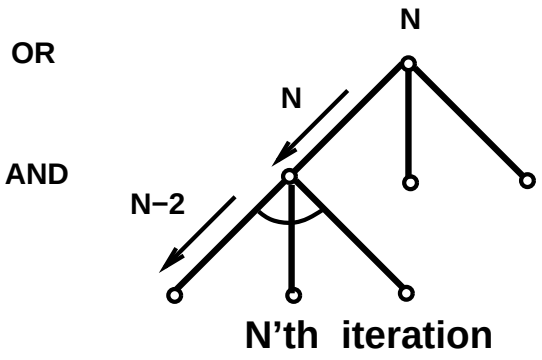


Figure 4.2: An example of iterative deepening using comspiracy number.

The original conspiracy number search presented by McAllester is a kind of best-first search, expanding selectively the potential solution tree of the minimum conspiracy number. It needs large memory space to store all nodes generated so far. Since the number of nodes generated during the course of search is often very large, it is impossible to retain all the generated nodes of the problem having very large search space.

Our search algorithm uses a depth-first iterative deepening method, incrementing the conspiracy threshold at each iteration. Best-first search using conspiracy numbers can be arranged to depth-fist iterative deepening by making the selection of *pst* from those having the same conspiracy number and the selection of the node to be expanded in the selected *pst* in depth-first order. The search process of iterative deepening algorithm using conspiracy numbers is as follows (See Figure 4.2). At the beginning of N'th iteration, all the potential solution trees at the root node have been generated and proved to be unsolvable within a conspiracy number $N-1$. Then the conspiracy threshold number N is given to the root node, and the root (type OR) node is expanded and some potential solution trees of depth 1 whose conspiracy numbers are 1 are generated. The conspiracy number 1 does not exceeds the threshold N, so the leftmost successor AND node is expanded, and counts the number of the successor OR nodes ($= N_1$). For example, in Figure 4.2, $N_1 = 3$. Then the conspiracy number of leftmost potential solution tree becomes N_1. If N_1 exceeds the threshold number N, then we abandon the expansion of the leftmost *pst*, and return to the parent OR node (the root node in this case), and expand the next *pst* rooted at the OR node. Else if N_1 does not exceed the

threshold N, then we continue to expand the leftmost *pst* by giving the conspiracy threshold $N - N_1 + 1$ to the leftmost successor OR node.

4.2 Some Hashing Methods

The depth-first iterative deepening method discards all the nodes generated in previous iterations except a root node, and resumes the next iteration by expanding the root node. So it needs to utilize the results of previous iterations effectively to avoid useless re-calculations. The information of the node n expanded at least once and searched deeper is stored in the hash table after n is solved or turned to be unsolvable within the given conspiracy threshold.

4.2.1 The Information Stored in Hash Table

When a node n is turned out to be unsolvable within a given threshold, the conspiracy number stored in hash table is calculated as follows. If n is an OR node, the minimum conspiracy number of the successor AND nodes becomes the conspiracy number of n. If n is an AND node, the sum of the conspiracy number of the successor OR nodes becomes the conspiracy number of n.

The conspiracy number of an unsolve node n is larger than or equal to the conspiracy threshold given to n. The difference of the conspiracy number and the given conspiracy threshold came from the counting method when an AND node is expanded, that is, all the successor nodes are generated and counted even after the node generation of a few successor nodes suffices to exceed the given conspiracy threshold. This is efficient to minimize the effort of node reexpansions.

4.2.2 Dynamic Evaluation by Hashing

Next we consider the counting method of conspiracy numbers of tip nodes which have been expanded at previous iterations. When a tip node n of type AND is expanded, the number of the successor OR nodes generated is added to conspiracy number of n, because the conspiracy number of unsolved tip nodes are defined as 1. It is natural for the tip nodes never expanded so far to be counted 1.

But when using iterative deepening method, some of the successor OR nodes may be previously expanded and turned out to be unsolvable within the given conspiracy threshold. The previous results of expansions can be obtained from a hash

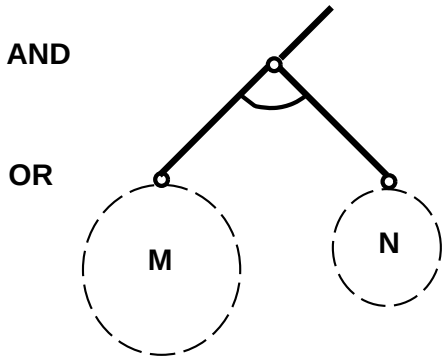


Figure 4.3: Dynamic evaluation by hashing.

table. For the tip nodes which have been previously expanded, we can give another conspiracy numbers to the nodes. If the tip node n has been expanded previously and turned out to be unsolved with a conspiracy number N, it is reasonable and efficient for the node n to be counted N. Else if n has been already solved at the previous iteration, it is counted 0. Else if n turned out to be unsolvable previously, conspiracy number ∞ is given to n. Else if n has never been expanded before, then it is counted 1. We call the dynamic counting method of conspiracy numbers "dynamic evaluation by hashing".

We can count the conspiracy numbers of tip nodes by using some static evaluation function, to reflect the difficulty of the tip nodes to be solved. But such a static evaluation function needs application-dependent knowledge, and wastes time to evaluate nodes, and such heuristic is unreliable. Instead of it, we utilize experience, the result of previous iterations. It is trustworthy than static evaluation functions, because it evaluates dynamically by expanding nodes.

4.2.3 Efficient Ordering by Hashing

The following theorems are easily obtained from the definition of conspiracy numbers for AND/OR tree search.

Theorem 1 *The number of iterations required to find a solution tree is more than or equal to the maximum number of the successor nodes of all AND nodes in the solution tree.*

Theorem 2 *The number of iterations required to find a solution tree is less than or equal to the number of the tip nodes in the solution tree.*

The number of iterations needed to find a solution tree depends on the successor ordering of AND nodes. The optimal ordering at the AND node is the ordering by which the number of nodes generated to solve the AND node is minimized. The optimal ordering is as follows. At the AND nodes in the solution tree, it is efficient to order the successor OR nodes in the worst-first order which means that the successor node solved within the smallest conspiracy number is ordered first, and the largest is ordered last. At the AND nodes out of the solution tree, it is efficient to order the successor OR nodes in the best-first order which means that the successor nodes solved within the smallest conspiracy number is ordered last and the successor node which is unsolvable is ordered first. But it is difficult to expect the conspiracy number needed to solve the successor OR nodes.

Hashing is also used to the successor ordering of AND nodes. It is efficient to order first the successor OR nodes which have never been expanded at the previous iterations. Reexpanding the successor nodes which have been searched deeper at the previous iterations is inefficient, because the overhead of the node reexpansions is expected to be high.

The successor ordering of AND nodes is difficult in general [40], so it is opened as a previous work.

4.3 Multiple Iterative Deepening

The conspiracy number search using iterative deepening method can not always select the optimal potential solution tree whose conspiracy number is minimum not only in global but also in local. The conspiracy threshold more than 1 may be given to the node which has been expanded previously. It comes from the nature of conspiracy number, that is, it may overestimate the conspiracy number of nodes. For example, when the OR node which has been expanded and unsolved with conspiracy number N at the previous iterations is solved at the current iteration, the conspiracy threshold $N+1$ is given to the next OR node in the same potential solution tree. If the difference between the conspiracy number needed to solve the next OR node and conspiracy threshold given to the node is large, then the

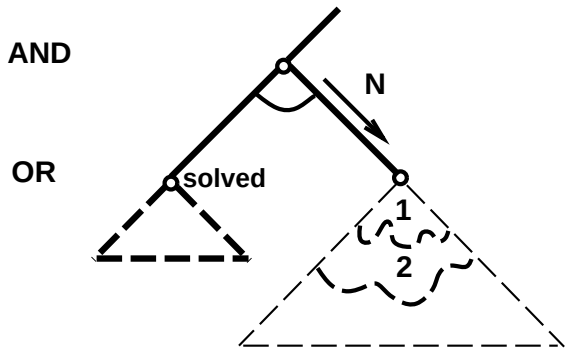


Figure 4.4: Multiple iterative deepening method.

successor ordering of the OR node becomes crucial. The reason is that if the solution tree rooted at the OR node n lies in the right side of the subtree rooted at n, and the conspiracy number of the *pst* is relatively small, then it takes much more time than expected from the difficulty to solve n.

We use the iterative deepening method not only at the root node but also at all the other OR nodes. We call it "multiple iterative deepening method". Now you may feel that the overhead of recalculation of previously expanded nodes may be high. But we can reduce the ratio of the reexpansions by changing the successor ordering of AND nodes at each iteration, that is, order the successor OR node which has never been expanded first. Our experimental result in Chapter 5 shows that the ratio of reexpanded nodes to all expanded nodes is about 20 %. By using multiple iterative deepening method, we can always select the optimal potential solution tree not only in global but also in local.

4.4 The Program List of the C* Algorithm

Next we show the procedure of the C* algorithm in the form of ALGOL-like program list.

comment Iterative deepening at the root node;
procedure ITERATE:
  comment Initialize the conspiracy threshold at the root;
  $CN(r) \leftarrow 1$

while do begin
if (EXPAND$(r, CN(r)) = 1$) **then**
return
$CN(r) \leftarrow CN(r) + 1$
end

comment Expand the tip nodes of the selected optimal potential solution tree;
procedure EXPAND$(n, CN(n))$:

1. **comment** Search for the entry of n in the hash table;
if FINDTABLE$(n, c) = 1$ **then**
begin
if $c = 0$ **then return** 1
else if $c \geq CN(n)$ **then return** 0
end

2. **comment** Expand a node n to generate all its successor nodes;
GENERATE(n)
if n has no successor nodes **then begin**
if n is an AND node **then begin**
PUTINTABLE$(n, 0)$
return 1
end
if n is an OR node **then begin**
PUTINTABLE(n, ∞)
return 0
end
end

3. **comment** To detect cycles, the conspiracy threshold given to n is stored in the hash table;
PUTINTABLE$(n, CN(n))$

4. **comment** Count the conspiracy numbers of each successor OR node by hashing;
if n is an AND node **then begin**

for each n_i **do begin**

if FINDTABLE$(n_i, c) = 1$ **then**

begin

if $c = 0$ **then**

n_i is removed from the set of unsolved successor OR nodes;

else $CN(n_i) \leftarrow c$

end

else $CN(n_i) \leftarrow 1$

end

comment When the sum of the conspiracy numbers of the successor OR nodes exceeds the conspiracy threshold given to n;

if $\sum_{i=1}^{k} CN(n_i) > CN(n)$ **then begin**

$CN(n) \leftarrow \sum_{i=1}^{k} CN(n_i)$

PUTINTABLE$(n, CN(n))$

return 0

end

else

$CN(n) \leftarrow CN(n) - \sum_{i=1}^{k} CN(n_i)$

end

5. **if** n is an OR node **then begin**

comment The recursive call of the successor AND nodes;

for each n_i **do begin**

$CN(n_i) \leftarrow CN(n)$

if EXPAND$(n_i, CN(n_i)) = 1$ **then**

begin

PUTINTABLE$(n_i, 0)$

return 1

end

end

comment $CN(n)$ is revised and the information of n is stored in the hash table;

$CN(n) = \min_{1 \le i \le k} (CN(n_i))$

PUTINTABLE$(n, CN(n))$
return 0
end

6. **if** n is an AND node **then begin**
comment The recursive call of the successor OR nodes;
for each n_i **do begin**
$CN(n_i) \leftarrow CN(n) + CN(n_i)$
comment Multiple iterative deepening method;
while do begin
comment In case n_i is solved;
if EXPAND$(n_i, CN(n_i)) = 1$ **then**
break
comment In case n_i is unsolved within the given conspiracy threshold;
if $CN(n_i) > CN(n)$ **then begin**
comment $CN(n)$ is revised and the information of n is stored in the hash table;
$CN(n) \leftarrow \sum_{j=i}^{k} CN(n_j)$
PUTINTABLE$(n, CN(n))$
return 0
end
$CN(n_i) \leftarrow CN(n_i) + 1$
end
end
PUTINTABLE$(n, 0)$
return 1
end

comment The procedure of searching for the entry of n in the hash table;
procedure FINDTABLE(n, c):
if the entry of n is in the hash table **then begin**
$c \leftarrow table(n)$
return 1
end

else

return 0

comment The procedure of entering the information of n to the hash table;
procedure PUTINTABLE(n, c):

if the entry of n is in the hash table **then**

if $c = 0$ or $c \geq table(n)$ **then**

$table(n) \leftarrow c$

else

$table(n) \leftarrow c$

4.5 The Dependence among the Decomposed Subproblems

The basic assumption of the conspiracy number search is that the decomposed problems are independent each other. The probability that the each unsolved subproblem becomes solved is regarded to be independent each other. There is a strong correlation between the conspiracy number of a node n and the probability of n to be solved. So it is reasonable to expand primarily the tip nodes of the potential solution tree of minimum conspiracy number. But such dependence does not hold in many applications [29].

In this section, the dependence between nodes and the effective search utilizing such dependence are described. There are two kinds of dependence between nodes. One of them is that the dominance relation holds among the nodes, and the other is that such a relation does not hold but the nodes are similar. The hash table is used not only to prevent the redundant reexpansions of nodes when using multiple iterative deepening, but also to detect such dependence among nodes.

4.5.1 Dominance Relations

Even if two decomposed subproblems P_i and P_j are not equal, a dominance relation may hold [13]. If P_i dominates P_j, the dominance relation between P_i and P_j is denoted $P_i D P_j$ which represents that P_i always has a better solution than P_j. If P_j is solved, P_i can be determined to be solved without searching it further.

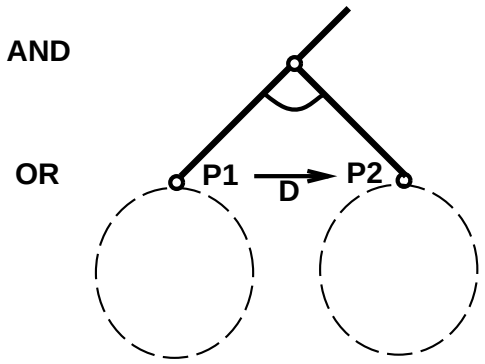


Table 4.1: A dominance relation between two nodes.

Conversely if P_i is unsolvable, P_j is also unsolvable, so P_j need not to be expanded further. So if the dominance relations are checked before expanding a node, the total number of generated nodes becomes smaller.

When an AND node is expanded, between the successor nodes P_i and P_j the dominance relation P_iDP_j may hold. In such a case, it would be better to expand first P_j before expanding P_i. The reason is as follows. If P_j is solved, P_i can be determined to be solved without being expanded from the dominance relation P_iDP_j, and also if P_j is unsolvable, P_i need not to be expanded because the parent node is type AND.

In Tsume-Shogi, the dominance relation between two positions holds only in the following case. Let the set of captured pieces of Black side at a position P_i be B_i, and the set of captured pieces of White side at a position P_i be W_i. The dominamce relation holds only in the case that the board which represents the position P_i is the same as that of P_j, except that B_i is a superset of B_j, which also means that W_i is a subset of W_j. In such a case, a dominance relation P_iDP_j holds, and if P_j is solved, P_i can always be solved with the same move-sequence as P_j, so P_i need not to be expanded.

In Tsume-Shogi, other dominance relations never hold. The reason is that "Uchi-Fu-Tsume", mating the White's King by dropping a Pawn, is forbidden by the rule of Shogi (also applied to Tsume-Shogi similarly). In such a case, it may be good to weaken the power of Black's pieces intentionally to avoid Uchi-Fu-Tsume. For example, the promoting move of Rook (called Hisha in Shogi), Bishop (Kaku), or Pawn (Fu), which seems to dominate the non-promoting move of the same piece

to the same square in general, does not always dominate. And also there is a case such that if the Black side has a Pawn or Lance instead of having a Rook, the current unsolvable position would becomes solved, one of these cases is called "Hisha-Saki-Hisha-Fu" problem.

The typical example of the cases that the dominance relations are utilized to prune nodes effectively is as follows. The interposed dropping move of White side on the square where there are no White's pieces' ataks against the Black's check (Oute) by Rook (also promoted Rook) or bishop (also promoted Bishop) or Lance is called Chu-Ai. Chu-Ai often becomes a useless or meaningless move (called Muda-Ai). Even if Chu-Ai moves are neglected and excluded from the possible moves of White side, the correct solution can be obtained in many problems. But in few problems, Chu-Ai is not Muda-Ai, and it becomes the best move of White side. Chu-Ai often leads to the positions among which many dominance relations hold. The most effective successor ordering at AND nodes in such cases is as follows. The moves except Chu-Ai are expanded first, and then Chu-Ai moves are expanded later. Among Chu-Ai, they are ordered in the increasing order of the distance from the White King to the dropping square. By using such ordering, many positions can be pruned by utilizing dominance relations.

4.5.2 Killer Heuristic

There are often the cases that the decomposed subproblems (positions) are not dominated each other, but these positions are similar, and solved by almost the same move-sequence. It does not always hold, but when the probability is high, the moves used to solve one of similar positions would be better to order first in solving similar positions. This is a kind of killer heuristic or history heuristic in the research of computer chess [48]. In the case of AND/OR tree search, the killer heuristic is used only at OR nodes.

Our Tsume-Shogi program uses the killer heuristic against the interposed moves of White side called Ai-Koma, and also against the non-promoting moves of White side when having the right of promotion (called Narazu). The maximum possibility of pieces used as Ai-Koma is seven, from Pawn (Fu) to Rook (Hisha), so if the seven positions are solved independently, it takes much more time than that taken by human experts. We utilize the result of similar positions already solved, which

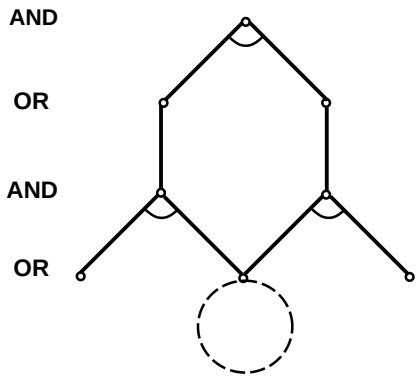


Figure 4.5: An example of a potential solution graph.

is different from the current position with only the kind of one piece dropped as White side's Ai-Koma or promotion move at several moves before by using the hash table. If a position n reached by Ai-Koma of White side is solved, then other similar positions are solved by ordering first the best move at n.

4.5.3 Pathological Cases When a Solution is a Graph

The above cases can be effectively solved by using a hash table. But when the solution is a graph, not a tree, the C* takes more computation time inferred from the degree of the difficulty for human experts to solve the problem. The reason is that when the solution is a cycle and two paths leaved from an AND node meet again at an OR node n later, C* dynamically counts the conspiracy number of n twice within the same potential solution tree, so the number of iterations needed to find the solution tree becomes larger than the solution's true conspiracy number as a graph (See Figure 4.5).

Of the unsolved 5 problems in the book "Zuku-Tsumuya-Tsumazaruya" experimented in Chapter 5, No.48 and No.49 are the problems of this kind, the solution of each problem is a graph and the conspiracy numbers counted doubly is considerably large. Especially the structure of the solution graph of No.49 is very complicated, the problem is called "A Big Labyrinth".

The inefficiency of C* can be detected among the nodes between which dominance relations or similarity hold in the same potential solution tree. Such dependent relations among nodes occur more frequently in Shogi or Chess. The improvement of C* against such games remains as a future work.

4.6 C*'s Admissibility and Optimality

The conspiracy number of an indefinitely large solution tree may be infinite. If an AND node has only one unsolved successor node, the conspiracy thresholds given to the AND node and its successor unsolved OR node are equal. So the algorithm may continue to search an infinite move-sequence with the same conspiracy threshold. It is possible to use modified conspiracy numbers, which is the same as the conspiracy numbers already defined except for adding 1 to the conspiracy number at each AND node [11]. By such a modification, the algorithm is guaranteed to terminate. But C* does not require such a modification. We detect cycles by hashing, and the node reached by a move-sequence containing cycles is determined to be unsolvable. And also the possible board positions of a Tsume-Shogi problem are finite, so a move-sequence containing no cycles never continue forever.

C* uses a multiple iterative deepening method, so at any nodes, it always finds an optimal partial solution tree in the sense that it is found by the minimum conspiracy threshold, on the condition that the ordering of successor nodes is fixed.

Only the nodes on the path from a root to the currently generated node are in memory. The space complexity of C* is $O(d)$, when d denotes the maximum depth of all generated nodes.

The general defect of iterative deepening method is the overhead of node reexpansions. In case the number of expanded nodes for the first time in each iteration is not adequately large, the rate of the number of reexpanded nodes to the total number expanded nodes becomes high, and the algorithm takes much more time than other algorithms such as best-first search. For applications which requires too much computation time due to the excessive reexpansions of nodes, a modification of depth-first iterative deepening method is presented [47]. But in our algorithm, as the following experimental results in Chapter 5 show, the rate of reexpansions is about 20%, adequately small for Tsume-Shogi problems.

Chapter 5

Experimental Results and Some Analysis

5.1 Zoku-Tsumuya-Tsumazaruya

We selected the set of Tsume-Shogi problems in the book "Zoku-Tsumuya-Tsumazaruya" [18] as benchmark problems. It carries about 200 problems from the Edo era to the Showa era, created by 41 persons. The shortest problem of them is 9-move and the longest is 873-move. There are various types of problems, so it will be a good benchmark to measure the performance of a Tsume-Shogi solving program.

The experimental results of Tsume-Shogi solving programs T2 and Ito to this book are as follows. T2 solved about 70 problems, and Ito solved about 135 problems [15] [23].

In our experiment, of course we did not teach the length of the solution-sequence of each problem to our program in advance before the experiment, also not limit the search depth to the length of the solution-sequence. Also we did not restrict the number of the moves to be expanded at each position like such a method as forward pruning used by game playing programs.

The result of experiments are shown in the form of tables. The notation of the tables is as follows. "NO" denotes the problem number. "LE" is the length of the solution-sequence, and the length of the Yo-Tsume sequence, if any, is in parenthesis. "SO" is the estimate of the solution-sequence our program answered. The mark "!}" shows that the solution-sequence answered is the same as the correct solution-sequence intended by the creator. "!{" denotes that our program answered the another solution-sequence than the creator intended, called "Yo-Tsume". "!!$"

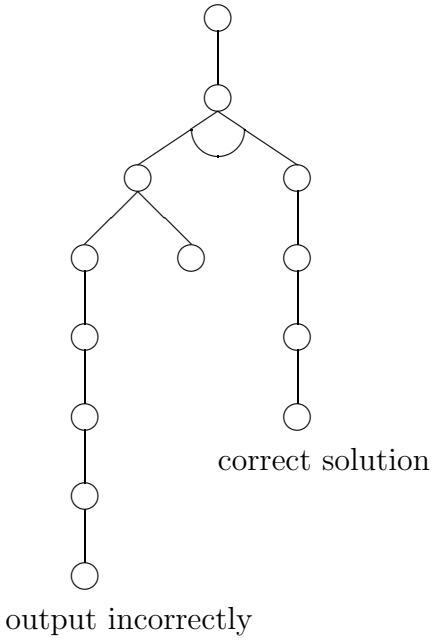


Figure 5.1: An example of answering "Henka-Betsu-Tsume", marked "!#$".

shows that our program answered the longer solution-sequence than the intended solution-sequence, called "Henka-Betsu-Tsume" (See Figure 5.1). "!|" is the same as "!{", except that the problem is not incomplete, because the Yo-Tsume sequence is trivial, not essential, called "Shusoku-Yo-Tsume". "!_" shows that the problem could not be solved within the given time limit. "–" shows either of the following three cases. The first is that the problem has no solutions, because the creator of the problem failed to notice the good moves of White side leading to unsolvable positions. The problems are called "Fu-Tsume", and regarded as incomplete, of course. The problem numbers of this kind are 42, 113, 119, 133, and 175. The second is that the problem is intended to have no solutions. The creator demands that the solvers find the excellent move-sequence of White side to escape from mates, it is called "Nogare-Zushiki". The problem numbers of this kind are 26, and 31. The third is that the problem contains the piece called "Suizou" which is used in the game "Chu-Shogi". The problem number of this kind is 35.

"SE" shows the CPU time in seconds consumed in solving the problem. "GE" means the total number of nodes generated. "EX" means the total number of nodes expanded. "IT" is the number of iterations at the root node required to

solve the problem. "TE" denotes the number of terminal nodes in the solution tree of the problem. "RE" shows the ratio of the number of reexpanded nodes to the total number of expanded nodes. "BR" is the average branching factor of all nodes expanded in the search process.

A complete Tsume-Shogi problem has only one solution-sequence called "Sakui-Tejun" intended by the creator. But in general there are some solution trees even in complete problems. At least one move-sequence from the root to a terminal AND node is contained in all the solution trees, and the longest sequences of them is the solution-sequence intended by the creator, called "Sakui-Tejun". Some solution trees may contain the move-sequence longer than "Sakui-Tejun" which would not be generated if the program searched first the moves in the "Sakui-Tejun" at the OR nodes. And if the program searched one of these solution tree, it answers the different solution-sequence from the "Sakui-Tejun". The solution-sequence of this type is called "Henka-Betsu-Tsume", marked "**"$**". Figure 5.1 is an example of "Henka-Betsu-Tsume".

The No.30 program consists of 4 programs of relatively small size, so the total number of the programs printed in the "Zoku-Tsumuya-Tsumazaruya" is 203, and excluded the unsuitable 8 programs marked "–". We tried the 195 Tsume-Shogi problems in the "Zoku-Tsumuya-Tsumazaruya" by restricting the CPU time limit to 2 hours per a problem on Sun Sparc Station 20, and the numerical results are shown in the following 8 tables.

NO	LE	SO	SE	GE	EX	IT	TE	RE	BR
1	15	"$	3.7	38185	8924	11	59	.259	4.28
2	23	!}	0.4	3371	1189	5	8	.193	2.84
3	23	!}	0.7	8466	1895	10	792	.269	4.47
4	17	!}	0.3	3322	899	5	14	.197	3.70
5	31	!}	14.5	160711	32156	10	38	.216	5.00
6	19	!}	4.2	68988	9644	13	27	.274	7.15
7	33	"$	36.2	478487	86621	11	920	.212	5.52
8	11	!}	1.5	22874	3161	6	9	.099	7.24
9	21	"$	190.9	2483019	498470	48	488	.252	4.98
10	21	!}	12.9	159079	29752	9	56	.242	5.35
11	17	!}	16.0	186615	29937	8	92	.150	6.23
12	19	"$	509.6	5784812	775447	37	1364101	.138	7.46
13	27	!{	277.5	2804388	458552	17	349945	.131	6.12
14	35	"$	3637.9	39029934	8023985	47	60807	.181	4.86
15	37	"$	71.2	528216	121331	33	1206	.236	4.35
16	45	!{	112.5	1200285	189991	16	93975	.142	6.32
17	55	!}	0.4	2307	654	5	5	.185	3.53
18	23	!}	0.7	9095	1589	5	37	.154	5.72
19	25	!}	10.6	93199	17432	7	30	.192	5.35
20	25(21)	!{	19.0	145062	34458	8	555	.208	4.21
21	35	!}	4.9	47790	9062	7	45	.160	5.27
22	37	"$	17.8	154095	27891	13	3943	.233	5.52
23	39	!}	2.3	22331	5027	6	1243939	.192	4.44
24	57	!}	26.1	301244	45470	17	4473	.209	6.63
25	23	!}	138.6	1316635	214766	19	391	.129	6.13

Table 5.1: The results of the problems No.1 to 25 in "Zoku-Tsumuya-Tsumazaruya".

NO	LE	SO	SE	GE	EX	IT	TE	RE	BR
26	25	–	–	–	–	–	–	–	–
27	99(21)	!{	64.2	448731	89474	11	7217	.147	5.02
28	37	!|	13.6	177223	49222	12	25	.260	3.60
29	15	!}	0.2	3886	548	5	9	.135	7.09
30.1	19	"$	0.6	7728	1767	4	119	.094	4.37
30.2	15	"$	6.8	107230	21259	13	211	.237	5.04
30.3	15	!}	0.2	3078	526	4	6	.186	5.85
30.4	11	!}	0.1	1016	223	2	3	.166	4.56
31	10	–	–	–	–	–	–	–	–
32	67(23)	!{	0.1	758	162	1	5	.012	4.68
33	391	"$	682.2	5815667	1144191	23	232629416	.223	5.08
34	19	"$	210.6	3413691	431265	43	5599426	.188	7.92
35	45	–	–	–	–	–	–	–	–
36	15	!}	20.9	259352	41753	8	24	.198	6.21
37	19	!}	5.0	70010	12361	12	58	.239	5.66
38	19	!}	13.1	152313	28653	6	26	.155	5.32
39	33(19)	!}	5.9	80809	13765	8	198	.190	5.87
40	35	"$	1304.4	16314350	3063547	23	12847	.122	5.33
41	41	!}	20.2	178273	40942	11	891	.226	4.35
42	81	–	–	–	–	–	–	–	–
43	19	"$	42.1	497521	81381	13	25	.190	6.11
44	19	!}	0.1	126	39	2	2	.077	3.23
45	23	"$	1763.3	15688691	2746704	66	1152435	.167	5.71
46	45	!}	10.2	83935	16053	16	325	.208	5.23
47	79	!|	1235.8	9641329	1824098	47	2961135	.305	5.29
48	119	!_	–	–	–	–	–	–	–
49	163	!_	–	–	–	–	–	–	–
50	69	"$	2747.3	29206901	5390072	30	16286057	.223	5.42

Table 5.2: The results of the problems No.26 to 50 in "Zoku-Tsumuya-Tsumazaruya".

NO	LE	SO	SE	GE	EX	IT	TE	RE	BR
51	41	!_	–	–	–	–	–	–	–
52	69	!}	15.5	146148	34897	13	1774834	.250	4.19
53	85	!|	387.4	2652108	575489	16	2880563553	.234	4.61
54	117	!}	684.2	7838296	1340932	22	4057125849	.215	5.85
55	611	!|	5291.6	36059779	7408022	60	12527	.235	4.87
56	23	"$	24.5	261127	46400	12	431971	.157	5.63
57	17	!}	51.1	646476	74579	9	30	.114	8.67
58	23	!}	1.1	10274	2873	8	85	.179	3.58
59	27	!}	43.1	684329	112382	10	117	.230	6.09
60	29	"$	79.9	852435	131567	24	14557	.232	6.48
61	41	!}	53.4	478854	105046	14	540	.239	4.56
62	23	!}	2.0	18966	5190	5	49	.174	3.65
63	37	!}	2.5	21110	5406	6	698	.277	3.90
64	23	!}	5.8	64851	13959	6	14	.221	4.65
65	31	!}	1.4	12312	3162	5	12	.202	3.89
66	31	"$	18.6	227510	37231	16	4153	.255	6.11
67	31	!}	3.1	25249	6393	8	33	.235	3.95
68	35	!}	59.4	627554	92274	12	46	.211	6.80
69	19	!}	0.3	3434	861	4	6	.137	3.99
70	47(11)	!{	7.5	69704	13744	8	181	.205	5.07
71	79(35)	!}	186.1	1355873	259425	15	584	.235	5.23
72	39	"$	8.9	90955	13533	4	2621	.105	6.72
73	57	"$	12.4	154278	28086	13	21910	.204	5.49
74	29	!}	0.7	6836	1519	7	66	.178	4.50
75	49(45)	!{	12.9	127780	30844	7	62	.194	4.14

Table 5.3: The results of the problems No.51 to 75 in "Zoku-Tsumuya-Tsumazaruya".

NO	LE	SO	SE	GE	EX	IT	TE	RE	BR
76	51	"$	249.9	2204672	439438	35	15865	.239	5.02
77	37[45]	!}	37.5	320620	63630	15	76	.291	5.04
78	53(23)	!{	16.5	123772	26703	7	55607	.195	4.64
79	53(35)	!{	9.3	65515	12455	6	35	.224	5.26
80	81	!}	33.4	324424	48576	15	72868	.187	6.68
81	63	"$	356.4	2649161	609396	42	7146	.278	4.35
82	51	!}	38.2	254034	65122	15	110	.342	3.90
83	93(79)	"$	9.9	88771	14386	9	132	.154	6.17
84	123	!}	26.0	189734	39938	10	39256	.202	4.75
85	71(57)	"$	71.1	570080	121141	21	189677	.323	4.71
86	65(49)	!{	14.2	104112	24367	18	113	.397	4.27
87	59	!}	53.5	372497	84076	26	285	.307	4.43
88	73	!}	17.8	130699	29562	9	52	.246	4.42
89	77(43)	!{	51.9	294668	74439	15	920	.293	3.96
90	103(31)	!{	3.1	17457	4553	6	210	.178	3.83
91	39	"$	30.9	259651	53290	10	58694	.216	4.87
92	119	!}	88.3	712055	138016	18	4792	.207	5.16
93	115	!}	126.4	1002939	196618	20	522190	.246	5.10
94	15	!}	4.4	49511	11195	13	42	.270	4.42
95	39	!}	22.6	202022	40611	14	24334	.266	4.97
96	31	!}	9.2	123106	26956	17	53	.213	4.57
97	29	"$	6.6	52018	12855	11	50	.211	4.05
98	23	!}	193.6	1990178	320873	21	423	.195	6.20
99	27	!}	1.2	7990	1926	9	126	.233	4.15
100	43	!}	6.2	44865	9976	8	127	.235	4.50

Table 5.4: The results of the problems No.76 to 100 in "Zoku-Tsumuya-Tsumazaruya".

NO	LE	SO	SE	GE	EX	IT	TE	RE	BR
101	95	!}	44.5	363246	63023	10	3877332481	.189	5.76
102	69	!}	66.8	843112	120913	18	875382	.296	6.97
103	31	!}	142.4	1375329	271580	26	2176	.204	5.06
104	23(21)	!}	19.0	242148	39197	13	148	.281	6.18
105	25	"$	2471.7	26572081	4485130	32	3280	.154	5.92
106	29(25)	!}	50.4	487498	86750	10	9458	.200	5.62
107	33	"$	96.3	913367	177142	16	1517	.227	5.16
108	35	!}	972.0	9444377	1301550	58	10440	.285	7.26
109	41	"$	1183.8	9038188	1744043	21	1212	.196	5.18
110	59	"$	117.9	980858	173012	22	587	.260	5.67
111	65(61)	!{	18.4	173635	27844	14	362391	.212	6.24
112	321(51)	!{	1.4	6585	1575	5	14	.177	4.18
113	51	–	–	–	–	–	–	–	–
114	15	!}	2.1	25091	3961	5	751	.127	6.33
115	21	"$	10.0	104447	23401	10	269	.226	4.46
116	21	"$	3.2	26190	5745	9	81385	.239	4.56
117	27	"$	54.9	539725	120179	11	2361	.292	4.49
118	29	"$	1199.3	15391545	1724929	16	152793	.123	8.92
119	31	–	–	–	–	–	–	–	–
120	33	!|	225.5	2387939	402375	28	34919670	.178	5.93
121	35	!|	10.0	104082	18441	9	73	.206	5.64
122	41	!}	14.9	139004	22867	7	328	.203	6.08
123	43	"$	237.5	2442286	383117	17	1794593072	.146	6.37
124	43	!|	236.0	1699434	377517	18	1079	.259	4.50
125	43	"$	239.9	2013686	379531	36	154524	.189	5.31

Table 5.5: The results of the problems No.101 to 125 in "Zoku-Tsumuya-Tsumazaruya".

NO	LE	SO	SE	GE	EX	IT	TE	RE	BR
126	45	!}	504.1	4462634	778967	48	110667	.205	5.73
127	47	!}	77.0	918745	115043	14	765945	.128	7.99
128	23	"$	329.0	3192722	532172	29	10500420	.202	6.00
129	25	"$	3.8	46679	7921	9	2444	.156	5.89
130	31	!|	15.0	143095	28773	9	178	.191	4.97
131	41	!}	176.2	2023004	312194	14	6535	.183	6.48
132	43	!}	0.6	7488	1443	5	31	.235	5.19
133	53	–	–	–	–	–	–	–	–
134	67	!}	108.0	751016	165600	38	13107	.342	4.54
135	31	"$	4.0	33516	7402	18	872	.289	4.53
136	43	!|	23.8	206633	40719	17	1282	.256	5.07
137	53(27)	!{	15.2	104977	23122	19	435	.226	4.54
138	47	!}	148.8	1340380	227960	15	4942	.243	5.88
139	57	"$	4596.8	42947470	8109248	36	672873623	.278	5.30
140	265(133)	!{	14.5	100084	20712	6	346	.204	4.83
141	93	"$	180.4	1542838	323484	42	260902	.232	4.77
142	39	!}	0.7	6307	1082	4	61594	.164	5.83
143	59	!}	11.9	117947	17946	9	50	.127	6.57
144	15	!}	2.6	28634	6493	6	16	.206	4.41
145	15	!}	42.6	493362	75356	9	4755	.202	6.55
146	17	!}	7.7	97602	14765	10	69	.157	6.61
147	21	!}	3.5	29473	5591	5	221	.153	5.27
148	21	!}	9.4	98513	19021	9	36	.196	5.18
149	23	!}	9.0	108781	27272	8	16	.198	3.99
150	25	!}	6.9	51371	12894	9	105	.274	3.98

Table 5.6: The results of the problems No.126 to 150 in "Zoku-Tsumuya-Tsumazaruya".

NO	LE	SO	SE	GE	EX	IT	TE	RE	BR
151	25	!}	15.6	200278	29681	13	45	.149	6.75
152	25	!}	8.3	91507	17594	8	35	.207	5.20
153	27	!}	5.6	63082	13365	16	57	.191	4.72
154	27	!}	16.6	185224	32445	10	33	.184	5.71
155	31	!}	1162.7	14819628	2626565	20	293	.160	5.64
156	35	!}	13.7	154251	28186	8	15571	.208	5.47
157	37	!}	36.5	276984	64102	15	990	.230	4.32
158	51	"$	102.4	840819	187760	18	764	.246	4.48
159	51	"$	691.3	5736568	1188230	40	94631210	.259	4.83
160	53	"$	116.0	1273762	206242	20	484	.216	6.18
161	65	!}	160.0	1226765	261298	18	14284	.222	4.69
162	83	!}	20.9	205739	39154	10	745163132	.190	5.25
163	21	!}	0.4	2595	759	3	2500	.146	3.42
164	21	!}	0.3	2297	493	8	84	.185	4.66
165	17	"$	802.1	8379959	1494049	17	27387786	.127	5.61
166	61	!}	77.0	547276	115821	13	4515331	.262	4.73
167	43	!}	2.7	17481	4345	13	1948	.247	4.02
168	39	!}	4.1	24778	7023	11	4291	.236	3.53
169	31	!}	0.6	3747	889	5	13	.281	4.21
170	33	"$	19.2	132028	29322	10	18305	.196	4.50
171	95	!}	272.7	2235169	427706	22	130341	.224	5.23
172	23	!}	13.9	146783	25726	18	4108	.134	5.71
173	35	!}	14.6	184217	45746	14	61	.282	4.03
174	87(19)	!{	9.0	59802	14956	9	3853	.259	4.00
175	47	–	–	–	–	–	–	–	–

Table 5.7: The results of the problems No.151 to 175 in "Zoku-Tsumuya-Tsumazaruya".

NO	LE	SO	SE	GE	EX	IT	TE	RE	BR
176	19	!}	0.3	3813	767	2	3	.141	4.97
177	15	!}	0.6	7171	1434	4	12	.209	5.00
178	27(27)	!{	19.1	288468	41197	14	185	.221	7.00
179	23	!}	2.2	29649	6598	8	223	.285	4.49
180	57	!}	7.7	50253	12693	8	169	.235	3.96
181	33	"$	3396.1	30188448	4390960	89	10312477	.227	6.88
182	43	!}	444.1	4718936	803064	24	16648	.202	5.88
183	29	"$	365.9	3622713	549175	22	10073772	.139	6.60
184	45	"$	2686.3	25391925	4643753	43	801370	.188	5.47
185	103	!}	1222.0	8663447	1734066	43	1314	.289	5.00
186	23	!}	0.5	5172	846	4	35	.136	6.11
187	47	!}	1078.1	13501938	1865991	23	8438	.130	7.24
188	57	!}	213.5	1595264	338073	29	837	.245	4.72
189	55(39)	!{	55.1	394935	82328	20	420	.203	4.80
190	19	!}	2.1	25869	4889	8	31	.227	5.29
191	17	"$	20.7	201923	41804	10	70	.259	4.83
192	31	!}	6.8	67951	14585	11	2283	.129	4.66
193	35	!_	–	–	–	–	–	–	–
194	47(25)	!{	140.6	1515452	226971	15	285869	.188	6.68
195	57	!}	470.7	3106936	577543	39	12596151	.229	5.38
196	105	!}	1272.2	10562696	2207003	60	1890501446	.241	4.79
197	75	!}	101.2	671439	145103	30	698	.231	4.63
198	103	"$	1455.8	18809178	4797823	46	473	.219	3.92
199	103	!_	–	–	–	–	–	–	–
200	873	"$	4606.3	33076119	6624029	26	185615722	.214	4.99

Table 5.8: The results of the problems No.176 to 200 in "Zoku-Tsumuya-Tsumazaruya".

NAME	SOLVED	RATIO (%)	YEAR
Ito	135	69	1992
T2	70	36	1992
SEO	190	97	1995

Table 5.9: The comparison of the number of solved programs in Zoku-Tsumuya-Tsumazaruya

solution length	problems	solved	ratio of solved (%)
11–19	29	29	100
21–29	44	44	100
31–39	38	37	97
41–49	24	23	96
51–99	44	44	100
101–873	16	13	81
Total	195	190	97

Table 5.10: The ratio of solved problems classified by the solution length.

The number of solved problems and its ratio classified by the length of the solution-sequence are shown in Table 5.10. The number of solved problems classified by the CPU time taken is in Table 5.11. The average of the ratio of reexpansions of the 190 problems solved is calculated as 0.208. The agerage of the branching factor of the 190 problems solved is calculated as 5.23.

The problem numbers and the length of the solution-sequences of all the unsolved problems are shown in the Table 5.13.

The length of the longest solution-sequence solved by our program is 873-move of No.200 program in Fugure 5.2. It is a new world record solved by computers! The previous record is 611-move (which is No.55 program in "Zoku-Tsumuya-Tsumazaruya") achieved by the programs Kawano, Ito, and T3 in 1994 [16].

We also found 2 imcomplete programs which have been regarded as complete so far. The No.13 program is Yo-Tsume (See Figure 5.3), and No.119 program is Fu-Tsume (See Figure 5.4).

CPU time (seconds)	the number of problems
0–1	21
1–10	44
10–100	68
100–1000	39
1000–7200	18
Total	190

Table 5.11: The number of solved problems classified by the CPU time taken on Sun Sparc Station 20.

reexpansions ratio	.208
branching factor	5.23

Table 5.12: The average of 190 problems solved.

the problem number (No.)	48	49	51	193	199
the length of the solution-sequence	119	163	41	35	103

Table 5.13: The problems unsolved within 2 hours.

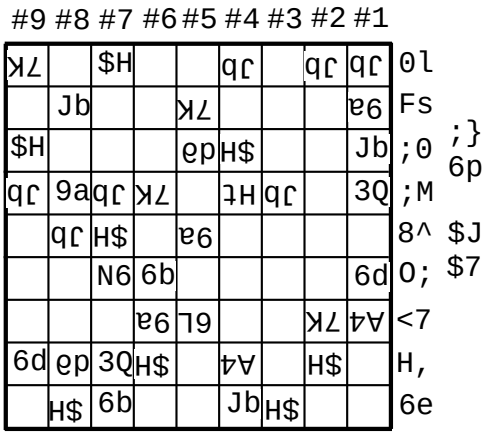


Figure 5.2: The No.200 which is a 873-move problem.

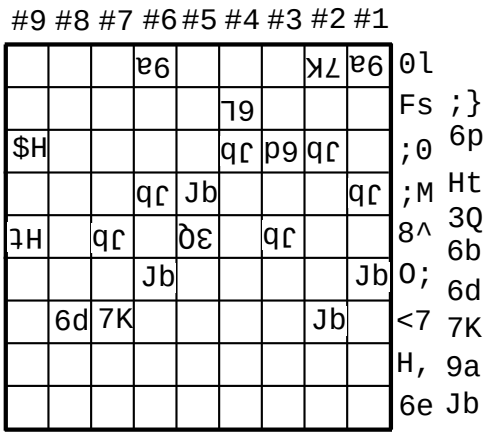


Figure 5.3: The No.13 program turned out to Yo-Tsume.

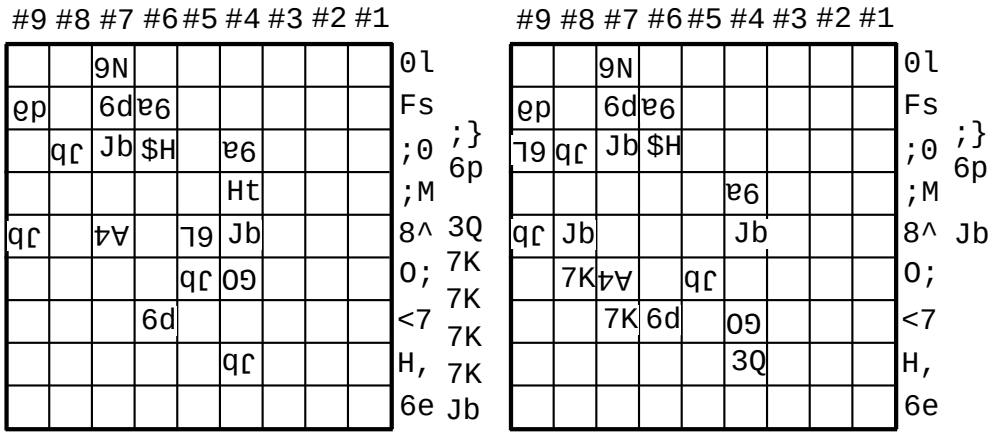


Figure 5.4: The No.119 program turned out to Fu-Tsume (left side) and the position of it after 15 moves (right side).

NAME	SOLVED	YEAR
Ito	62	1993
T3	68	1993
Kawano	92	1994
SEO	99	1995

Table 5.14: The comparison of the number of solved problems in Shogi-Zuko.

5.2 Shogi-Zuko

Though "Zoku-Tsumuya-Tsumazaruya" is a good benchmark set containig various types of problems, it has the following two defects.

1. It has relatively many incomplete problems, "Yo-Tsume" or "Fu-Tsume".

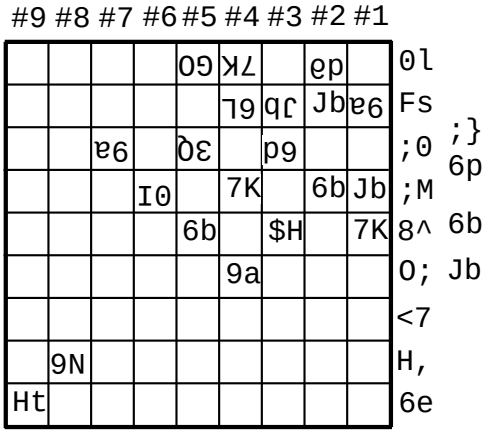
2. "Zoku-Tsumuya-Tsumazaruya" is not tried by other Tsume-shogi solving programs recently, so only the experimental results of other programs tried two years ago [in 1992 or 1993] exist. The results may be seemed to be somewhat old.

"Shogi-Zuko" is the set of Tsume-Shogi problems created by Kanju Ito in the Edo era [17]. The problems contained in it are not only hard to solve but also excellent as an art, and they often give artistic impressions to human solvers. Artistic impressions of Tsume-Shogi has been studied by Koyama and Kawano [25] [26] [27].

"Shogi-Zuko" is sophisticated and the number of imcomplete problems is relatively small. There are two Fu-Tsume problems in it, but revised version of those are created by other persons later, and the solution-sequences of these revised problems are the same as the original ones created by Kanju. So we use the revised problems for the two Fu-Tsume problems, No.73 and No.93.

"Shogi-Zuko" is tried to solve by three programs recently [in 1993 or 1994], and the results are showed in Table 5.14. The algorithm of all the three programs is best-first search.

Our results of "Shogi-Zuko" is as follows. Our program solved 99 problems within 2 hours per a problem on Sun Sparc Station 20, that is, it solved all problems



The No.8 problem [41-move] in "Shogi-Zuko".

except only one problem. The problem number of unsolved one is No.8. The solution length of it is 41-moves, not so long. So it may be seemed to be easy to solve the No.8. The No.8 problem is also No.51 in "Zoku-Tsumuya-Tsumazaruya".

The reason that No.8 could not be solved is seemed to be as follows. The solution tree of the No.8 is absolutely complicated, and the conspiracy number of the potential solution trees, those are the partial trees of the solution tree, is calculated to be large, so the algorithm selectively expands other potential solution trees unrelated to the solution tree.

Next we show the experimental results of "Shogi-Zuko", The notation of the following 4 tables is the same as "Zoku-Tsumuya-Tsumazaruya". We also did not teach the length of the solution-sequence of each problem to our program, and not restrict the search depth to the length of the solution-sequence.

NO	LE	SO	SE	GE	EX	IT	TE	RE	BR
1	69	"$	2747.3	29206901	5390072	30	16286057	.223	5.42
2	21	!}	6.8	75603	13724	9	18	.297	5.51
3	45	"$	6.9	65227	13031	6	310	.217	5.01
4	39	!}	38.2	447634	66636	20	622506	.211	6.72
5	21	"$	3.5	35450	6734	11	34	.260	5.26
6	81	!}	127.7	1418400	274976	17	111578	.234	5.16
7	13	!}	46.5	622248	94033	12	36	.177	6.62
8	41	!_	–	–	–	–	–	–	–
9	31	!}	156.5	1730731	308353	36	543	.209	5.61
10	73	!|	38.0	322314	70982	17	623	.249	4.54
11	49	!}	2.3	26467	5655	8	34	.257	4.68
12	21	!}	15.3	240468	36383	15	311	.273	6.61
13	17	!}	7.0	82656	13502	6	18	.163	6.12
14	25	!}	18.5	246260	39725	14	218	.198	6.20
15	29	!}	6.1	72193	13932	20	4885	.239	5.18
16	23	!}	6.8	73388	13887	6	850	.189	5.28
17	41	!}	928.6	9593607	1814748	64	780	.263	5.29
18	23	!}	0.6	6170	1581	7	222	.221	3.90
19	21	!}	8.6	130493	17968	7	34050	.216	7.26
20	35	"$	310.7	2673100	471383	27	3041	.255	5.67
21	21	"$	475.1	5715337	849142	17	441	.180	6.73
22	49	!}	5.6	46818	11084	7	152143	.188	4.22
23	23	"$	5148.6	64256721	10871666	99	194156	.217	5.91
24	43	!}	65.8	623000	96288	15	235	.157	6.47
25	35	"$	130.2	1122163	196871	20	83986	.196	5.70

Table 5.15: The results of the problems No.1 to 25 in "Shogi-Zuko".

NO	LE	SO	SE	GE	EX	IT	TE	RE	BR
26	25	"$	40.6	547321	90776	16	570	.202	6.03
27	43	"$	134.0	1617287	296264	11	646102	.239	5.46
28	39	!}	2.4	35197	5289	9	107	.132	6.65
29	15	!}	12.8	140144	20518	9	536	.118	6.83
30	17	!}	2.4	28578	5425	5	46	.171	5.27
31	23	"$	468.8	6056443	879077	27	32456246	.141	6.89
32	33	"$	70.7	783480	119402	19	279656	.182	6.56
33	15	!}	23.5	292809	50882	10	3458037	.221	5.75
34	17	"$	14.4	199185	31411	18	162	.176	6.34
35	17	!}	37.3	480155	62716	7	94	.142	7.66
36	13	!{	3.2	32661	7548	5	51	.173	4.33
37	23	!}	409.6	4874946	689301	19	93	.239	7.07
38	29	!}	8.6	83300	13804	8	88	.157	6.03
39	67	"$	45.2	524878	90265	11	3462	.236	5.81
40	21	"$	13.3	143469	23058	15	6336	.166	6.22
41	23	!}	27.7	249474	47143	8	32	.173	5.29
42	27	!}	0.5	4624	988	4	10	.170	4.68
43	85	!|	387.4	2652108	575489	16	2880563553	.234	4.61
44	15	!}	1772.9	14474454	3310013	15	33151	.175	4.37
45	45	!{	9.7	104531	18688	12	1860	.173	5.59
46	43	!}	17.9	202549	34891	9	179	.223	5.81
47	27	"$	570.0	7264729	1151953	46	281928	.163	6.31
48	25	"$	1481.2	19674194	3004847	24	164559	.168	6.55
49	47	!}	245.7	1918999	346784	14	227742	.245	5.53
50	9	"$	239.6	2684720	400977	13	865	.194	6.70

Table 5.16: The results of the problems No.51 to 75 in "Shogi-Zuko".

NO	LE	SO	SE	GE	EX	IT	TE	RE	BR
51	31	!}	16.3	211189	39343	12	114	.263	5.37
52	25	!}	8.2	80020	14540	8	18901	.213	5.50
53	33	"$	311.2	3551406	519372	13	5516	.193	6.84
54	39	!}	26.7	216653	41259	10	273	.227	5.25
55	23	!}	118.4	1272125	188401	14	957	.163	6.75
56	25	"$	166.8	1615430	260937	25	302	.222	6.19
57	65	"$	1968.0	21553155	3829547	88	89384242	.268	5.63
58	29	!}	6.8	71700	15511	7	59	.277	4.62
59	29	"$	194.9	2139653	371220	13	190227	.246	5.76
60	29	!}	47.9	680830	93539	25	5000	.222	7.28
61	29	"$	206.2	2437601	458189	15	6195444	.222	5.32
62	23	!}	74.0	995806	128538	13	83071	.176	7.75
63	35	!}	12.6	152424	30009	14	8177	.210	5.08
64	69	!}	15.5	146148	34897	13	1774834	.250	4.19
65	31	"$	39.6	473073	79754	9	304875	.230	5.93
66	51	!{	25.2	220668	32185	9	208	.180	6.86
67	47	"$	128.8	1638671	310105	17	12805	.217	5.28
68	25	!}	40.1	498516	86925	13	291	.222	5.74
69	13	"$	148.7	2068128	290160	40	2183	.149	7.13
70	27	!}	12.0	149441	24530	10	43	.161	6.09
71	39	"$	142.7	1490747	243131	11	107679882	.194	6.13
72	33	!}	42.2	394393	85720	13	63	.224	4.60
73	35	!}	93.7	1031135	170452	22	16729	.170	6.05
74	29	!}	4.2	38450	7392	8	203	.209	5.20
75	11	"$	48.2	565904	89128	16	207	.160	6.35

Table 5.17: The results of the problems No.51 to 75 in "Shogi-Zuko".

NO	LE	SO	SE	GE	EX	IT	TE	RE	BR
76	25	!}	9.4	117050	17177	9	21	.220	6.81
77	19	"$	35.0	422610	62535	12	61	.185	6.76
78	21	"$	62.9	651131	107462	20	6903	.225	6.06
79	13	"$	146.1	1552797	241436	18	5228	.160	6.43
80	29	!}	2.6	27844	4987	7	562	.158	5.58
81	17	!}	14.8	172557	23077	9	56	.196	7.48
82	21	!{	6.7	70913	12781	11	32	.224	5.55
83	261	"$	152.1	1625529	288789	16	524986369	.207	5.63
84	31	!}	31.3	305398	54037	9	373	.192	5.65
85	15	!}	2.4	32991	5447	5	14	.190	6.06
86	29	"$	165.7	1743457	306402	20	32150	.225	5.69
87	43	!}	91.3	1045164	179403	25	219	.246	5.83
88	35	"$	15.2	171324	37791	22	751	.191	4.53
89	29	"$	321.9	3674968	680103	33	20500007	.216	5.40
90	67	!}	507.5	4688436	902319	19	622871	.332	5.20
91	27	!}	979.3	13333222	1819696	16	601564710	.222	7.33
92	13	!}	6.1	75292	13191	8	85	.286	5.71
93	45	!}	88.8	782244	127001	16	2494	.178	6.16
94	23	!}	37.5	389312	64807	19	11466	.166	6.01
95	39	!}	8.0	120878	20382	7	213	.134	5.93
96	79	!}	41.0	408308	95686	13	6806642	.336	4.27
97	77	!}	104.5	1000607	214733	20	2927	.243	4.66
98	31	"$	21.8	382971	50747	10	598904127	.211	7.55
99	117	!}	684.2	7838296	1340932	22	4057125849	.215	5.85
100	611	!|	5291.6	36059779	7408022	60	12527	.235	4.87

Table 5.18: The results of the problems No.76 to 100 in "Shogi-Zuko".

Chapter 6

The C* Algorithm for Minimax Tree Search

The C* algorithm for AND/OR tree search can be incorporated into minimax tree search. Allis [1] presents such an incorporation method. His method iterates the two-valued search until the minimax value of the root node is fixed. The range of plausible values becomes strictly smaller as iterating the binary search procedure. However, it is sometimes impossible to decide the exact value of a minimax tree in practical game tree search within a time limit, and the space complexity of the method is too high, and it takes much time because of repeated traverses from the root to terminal nodes. And it is inefficient when the minimax value of the current iteration obtained by the move examined first at the root turned out to be larger than obtained at the iteration just before, all other moves at the root never have been examined at all, and it may take a large number of iterations to decide the minimax value of the root.

We combined the conspiracy number search with the idea of B* algorithm [4] [43]. It does not need to calculate the minimax value of the root in practice. It needs to decide only the best move at the root, that is, to find the immediate successor MIN node of the root in a solution tree. The B*-like method takes less time than deciding the minimax value of the root.

Also here we use an iterative deepening depth-first search, and in each iteration, moves at the root position are successively examined. A minimax tree search can be regarded as a series of AND/OR tree search. If the root value of a minimax tree is turned out to be m, the solution tree is a union of two solution trees. One of them is the solution tree which proves that the root value is greater than or equal

to m. Another is one which proves that the root value is less than or equal to m. Let the minimax value of a node n be m. If m is less than or equal to a value v, then the following hold.

1. $\text{"-CN}(n, v) = 0, \quad \forall v \geq m$
2. $\text{",CN}(n, v+1) > 0, \quad \forall v \geq m$

On the other hand, if m is greater than a value v, then the following hold.

1. $\text{"-CN}(n, v) > 0, \quad \forall v < m$
2. $\text{",CN}(n, v+1) = 0, \quad \forall v < m$

We now pay attention to the value BS, the minimax value gained at the iteration just before when searching the subtree rooted at the first successor node of the root, or the best minimax value obtained so far in the current iteration when investigating the subtrees rooted at one of the successor nodes except the first successor. We calculate the two conspiracy numbers of the root node r, $\text{"-CN}(r, BS)$ and $\text{",CN}(r, BS+1)$. After expanding a MAX node, the algorithm counts the number of successor MIN nodes whose values are evaluated as greater than or equal to $BS+1$, and after expanding a MIN node, it counts the number of successor MAX nodes whose values are evaluated as less than or equal to BS. When either of the two conspiracy numbers exceeds the given threshold to the root, the minimax value is backed up to the root, the conspiracy numbers of expanded nodes are revised, the current iteration terminates, and depth-first search is resumed at the next iteration.

Next we show the procedure of the C* algorithm for minimax tree search in the form of ALGOL-like program list.

comment Iterative deepening at the root node;
procedure ITERATE:
  comment Generate all the successor nodes $n_i, 1 \leq i \leq k$ at the root node;
  GENERATE(r)
  comment Evaluate successor nodes of the root by a static evaluation function;
  while each n_i **do**
    EVALUATE(n_i)

comment Set BS to the highest evaluated value;

$BS \leftarrow \max_{1 \le i \le k}$ EVALUATE(n_i)

comment Initialize the conspiracy threshold at the root;

", $CN(r, BS+1) \leftarrow 1$

"- $CN(r, BS) \leftarrow 1$

comment Sort the successor nodes of the root in non-increasing order of each evaluated value, ties are resolved arbitrary;

SORT$(n_i, 1 \le i \le k)$

while within a time limit **do begin**

if EXPAND$(r,$ ", $CN(r, BS+1),$ "- $CN(r, BS)) = \infty$ **or** $-\infty$ **then**

return

", $CN(r, BS+1) \leftarrow$ ", $CN(r, BS+1) + 1$

"- $CN(r, BS) \leftarrow$ "- $CN(r, BS) + 1$

end

comment Expand the tip nodes of the selected optimal potential tree;

procedure EXPAND$(n,$ ", $CN(n, BS+1),$ "- $CN(n, BS))$:

1. **comment** Expand a node n to generate all its successor nodes;
   GENERATE(n)
   if n has no successors **then begin**
   if n is a MIN node **then**
   return ∞
   end
   if n is a MAX node **then**
   return $-\infty$
   end
   end

2. **comment** Count the conspiracy numbers of all successor MIN nodes;
   if n is a MAX node **then begin**
   for each n_i **do begin**
   if $(v_i \leftarrow$ EVALUATE$(n_i)) \ge BS + 1$ **then**
   "- $CN(n_i, BS) \leftarrow 1$
   else

" - $CN(n_i, BS) \leftarrow 0$

end

comment When the sum of the conspiracy numbers of successor nodes exceeds the threshold conspiracy number of n;

if $\sum_{i=1}^{k}$ " - $CN(n_i, BS) >$ " - $CN(n, BS)$ **then begin**

" - $CN(n, BS) \leftarrow \sum_{i=1}^{k}$ " - $CN(n_i, BS)$

return $\max_{1 \leq i \leq k} v_i$

end

else

" - $CN(n, BS) \leftarrow$ " - $CN(n, BS) - \sum_{i=1}^{k}$ " - $CN(n_i, BS)$

end

3. **comment** Count the conspiracy numbers of all successor MAX nodes;

if n is a MIN node **then begin**

for each n_i **do begin**

if $(v_i \leftarrow \text{EVALUATE}(n_i)) \leq BS$ **then**

", $CN(n_i, BS+1) \leftarrow 1$

else

", $CN(n_i, BS+1) \leftarrow 0$

end

comment When the sum of the conspiracy numbers of successor nodes exceeds the threshold conspiracy number of n;

if $\sum_{i=1}^{k}$ ", $CN(n_i, BS+1) >$ ", $CN(n, BS+1)$ **then begin**

", $CN(n, BS+1) \leftarrow \sum_{i=1}^{k}$ ", $CN(n_i, BS+1)$

return $\min_{1 \leq i \leq k} v_i$

end

else

", $CN(n, BS+1) \leftarrow$ ", $CN(n, BS+1) - \sum_{i=1}^{k}$ ", $CN(n_i, BS+1)$

end

4. **if** n is an MAX node **then begin**

comment The recursive call of successor MIN nodes;

for each n_i **do begin**

$\texttt{"},CN(n_i, BS+1) \leftarrow \texttt{"},CN(n, BS+1)$

$\texttt{"-}CN(n_i, BS) \leftarrow \texttt{"-}CN(n, BS) + \texttt{"-}CN(n_i, BS)$

$v_i \leftarrow \text{EXPAND}(n_i, \texttt{"},CN(n_i, BS+1), \texttt{"-}CN(n_i, BS))$

if $v_i \geq BS+1$ **then**

return v_i

end

comment $\texttt{"},CN(n, BS+1)$ is revised.

$\texttt{"},CN(n, BS+1) \leftarrow \min_{1 \leq i \leq k} \texttt{"},CN(n_i, BS+1)$

return $\max_{1 \leq i \leq k} v_i$

end

5. **if** n is an MIN node **then begin**

comment The recursive call of successor MAX nodes;

for each n_i **do begin**

$\texttt{"},CN(n_i, BS+1) \leftarrow \texttt{"},CN(n, BS+1) + \texttt{"},CN(n_i, BS+1)$

$\texttt{"-}CN(n_i, BS) \leftarrow \texttt{"-}CN(n, BS)$

$v_i \leftarrow \text{EXPAND}(n_i, \texttt{"},CN(n_i, BS+1), \texttt{"-}CN(n_i, BS))$

if $v_i \leq BS$ **then**

return v_i

end

comment $\texttt{"-}CN(n, BS)$ is revised.

$\texttt{"-}CN(n, BS) \leftarrow \min_{1 \leq i \leq k} \texttt{"-}CN(n_i, BS)$

return $\min_{1 \leq i \leq k} v_i$

end

The methods for searching AND/OR trees such as multiple iterative deepening or dynamic evaluation by hashing are not incorporated to the above program list yet. The sophisticated version of C* for minimax tree search is left as a future work.

We did not implemented the above algorithm in game playing programs such as Shogi (Sashi-Shogi), because the dependence among the positions in Shogi is very strong, and the branching factor of Shogi is much greater than Tsume-Shogi, and the search space of Shogi is a graph, not a tree, so the conspiracy number itself may be meaningless, the algorithm will take much time. It may be needed to combine with other methods such as null-move heuristic [3].

Chapter 7

Conclusion

We developed the C* algorithm, depth-first iterative deepening for AND/OR tree search, based on conspiracy numbers. C* can find the solution tree of an AND/OR tree surprisingly fast. Instead of depending on the heuristic of human experts, it is based on the probability of finding a solution tree. It is enhanced by such methods as multiple iterative deepening, dynamic evaluation by hashing, efficient successor ordering, and pruning by dominance relations. A hash table is indispensable for these enhancements.

We implemented the C* algorithm to a Tsume-Shogi solving program, and performed experiments to solve a set of Tsume-Shogi problems consisted of about 200 problems in "Zoku-Tsumuya-Tsumazaruya". The results shows that 190 problems can be solved within 2 hours per a problem on Sun Sparce Station 20. The time taken to solve these problems seems to be much shorter than taken for human experts. The experimental result is far superior to the result of other Tsume-Shogi programs, and it verifies the excellence of the C* algorithm. Our program solved the No.200 problem called "Shin-Ougi-Tsume", whose solution length is 873-mate. It is a new world record for the length of the solution-sequence of a Tsume-Shogi problem solved by computers. Our program also tried 100 problems in "Shogi-Zuko", and solved 99 problems.

Reference

[1] Allis, L.V., Meulen, M., and Herik, H.J., "Proof-number search," *Artificial Intelligence, Vol.66,* 1994, pp.91–124.

[2] Anantharaman, T., Campbell, M.S. and Hsu, F., "Singular Extensions: Adding Selectivity to Bruteforce Searching," *Artificial Intelligence, Vol.43,* 1990, pp.99–109.

[3] Beal, D.F., "A Generalised Quiescence Search Algorithm," *Artificial Intelligence, Vol.43,* 1990, pp.85–98.

[4] Berliner, H.J., "The B* tree search – a best-first proof procedure," *Artificial Intelligence, Vol.12,* 1979, pp.23–40.

[5] Berliner, H.J., and Ebeling, C., "Pattern Knowledge and Search: The SUPREM Architecture," *Artificial Intelligence, vol.38,* 1989, 161–198.

[6] Berliner, H.J., Goetsch, G., Campbell, M.S. and Ebeling, C., "Measuring the Performance Potential of Chess Programs," *Artificial Intelligence, Vol.43,* 1990, 7–20.

[7] Chakrabati, P.P., Ghose, S. and DeSarkar, S.C., "Admissibility of AO* When Heuristic Overestimate," *Artificial Intelligence, Vol.34,* 1988, 97–113.

[8] Chakrabati, P.P., "Algorithms for searching explicit AND/OR graphs and their applications to reduction search," *Artificial Intelligence, Vol.65,* 1994, pp.329–345.

[9] Chang, C.L., and Slagle, J.R., "An Admissible and Optimal Algorithm for Searching AND/OR Graphs," *Artifitial Intelligence, Vol.2,* 1971, pp.117–128.

[10] Dasgupta, P., Chakrabati, P.P., and DeSarkar, S.C., "Agent Searching in a tree and the optimality of iterative deepening," *Attificial Intelligence, Vol.71,* 1994, pp.195–208.

[11] Elkan, C., "Conspiracy Numbers and Caching Searching And/Or Trees and Theorem-Proving," *IJCAI-89,* 1989, pp.341–346.

[12] Horacek, H., "Reasoning with Uncertainty in Computer Chess," *Artificial Intelligence, vol.43,* 1990, pp37–56.

[13] Ibaraki, T., "The Power of Dominance Relations in Branch-and-Bound Algorithm," *J. ACM,* 1977, pp.264–279.

[14] Iida, H., "A Strategy of Game Tree Search Modelling Experts' Thinking Process (in Japanese)," *J. Information Processing Society of Japan, Vol.33, No.11,* 1992, pp.1296–1305.

[15] Ito, T. and Noshita, K., "Two Fast Programs for Solving Tsume-Shogi and Their Evaluation (in Japanese)," *J. Information Processing Society of Japan, Vol.35, No.8,* 1994, pp.1531–1539.

[16] Ito, T., Kawano, Y., and Noshita, K., "Challenges for Solving Tsume-shogi with Extremely Long Solution-Steps (in Japanese)," *Programming Symposium, Information Processing Society of Japan,* 1994.

[17] Kadowaki, Y. "Tsumuya-Tsumazaruya, Shogi-Muso, Shogi-Zuko (in Japanese)," *Heibon-Sha, Toyo-Bunko,* 1975.

[18] Kadowaki, Y. "Zoku-Tsumuya-Tsumazaruya (in Japanese)," *Heibon-Sha, Toyo-Bunko,* 1978.

[19] Kaindl, H., "Quiescene search in computer chess," *Reprint in Computer Game-Playing: Theory and Practice,* 1983.

[20] Kaindl, H., "Minimaxing: Theory and practice," *AI Magazine, Vol.9(3),* 1988, pp.69–76,

[21] Knuth, D., and Moore, R., "An analysis of alpha-beta pruning," *Artificial Intelligence, Vol.6,* 1975, pp.293–326.

[22] Korf, R.E., "Depth-first iterative deepening: an optimal admissible tree search," *Artificial Intelligence, Vol.27,* 1985, pp.97–109.

[23] Kotani, Y., "Computer Shogi Association Material, Vol.6 (in Japanese)," *Computer Shogi Association,* 1992.

[24] Kotani, Y., Yoshikawa, T., Kakinoki, Y., and Morita, K., "Computer Shogi (in Japanese), " *Science-Sha,* Tokyo, 1990.

[25] Koyama, K., and Kawano, Y., "Quantitative Analysis of Impression on Tsume-Shogi (Mating Problems of Japanese Chess)," *SIGAI of Information Processing Society of Japan, 88-4* 1993, pp.29–42.

[26] Koyama, K., and Kawano, Y., "The Database of Tsume-Shogi Problems and Its Evaluations," *SIGAI of Information Processing Society of Japan, 88-5* 1993, pp.43–55.

[27] Koyama, K., and Kawano, Y., "Numerical Analysis of Favorable Impressions on Masterpieces of Tsume-shogi (Mating Problems of Japanese Chess)," *J. Information Processing Society of Japan, Vol.35, No.11,* 1994, pp.2338–2346.

[28] Kumar, V., and Kanal, L., N., "A General Branch and Bound Formulation for Understanding and Synthesizing AND/OR Tree Search Procedures," *Artifitial Intelligence, Vol.21,* 1983, pp.179–198.

[29] Levi, G., and Sirovich, F., "Generalized AND/OR Graphs," *Artificial Intelligence, Vol.7,* 1976, pp.243–259.

[30] Levy, D., and Newborn, M., "How Computers Play Chess," (in Japanese), *Science-Sha,* Tokyo, 1994.

[31] Martelli, A., and Montanari, U., "Optimizing Decision Trees Through Heuristically Guided Search," *Comm. ACM,* 1978, pp.1025–1039.

[32] Matsubara, H., "Shogi (Japanese Chess) as the AI research target next to chess," *ETL technical report ETL-TR-93-23,* Electrotechnical Laboratory, 1993.

[33] Matsubara, H., "Some properties of Shogi (Japanese Chess) as a game (in Japanese)," *SIGAI of Information Processing Society of Japan, 96-3,* 1994, pp.21–30.

[34] McAllester, D.A., "Conspiracy numbers for minmax search," *Artificial Intelligence, Vol.35,* 1988, pp.287–310.

[35] Nau, D.S, "An Investigation of the Causes of Pathology in Games," *Artificial Intelligence, Vol.19,* 1982, pp.257–278.

[36] Nau, D.S, "Pathology on Game Trees Revised, and an Alternative to Minimaxing," *Artificial Intelligence, Vol.21,* 1983, pp.221–244.

[37] Nau, D.S., "Decision Quality As a Fuction of Search Depth on Game Trees," *J. ACM, Vol.30, No.4,* 1983, pp.687–708.

[38] Nau, D.S., and Kumar, V., and Kanal, L., "General Branch and Bound, and its Relation to A* and AO*," *Artificial Intelligence, Vol.23,* 1984, pp.29–58.

[39] Newborn,M., "Unsynchronized iteratively deepening parallel alpha-beta search," *IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol.10, No.5,* pp.687–694, 1988.

[40] Nilsson, N.J., "Problem Solving Methods in Artificial Intelligence," *McGraw-Hill, New York,* 1972.

[41] Nilsson, N.J., "Principles of Artificial Intelligence," *Tioga, Palo Alto, CA,* 1980.

[42] Noshita, K., "How to make a quick and accurate Tsume-shogi solver (in Japanese)," *Shingaku-Kenkyukai, COMP91–56,* 1991, pp.29–37.

[43] Palay, A.J., "The B* Tree Search Argorithm – New Result," *Artificial Intelligence, Vol.19,* 1982, pp.145–163.

[44] Pearl. J, "On the Nature of Pathology in Game Searching," *Artificial Intelligence, Vol.20,* 1983, pp.427–453.

[45] Pearl, J., "Heuristics: Intelligent search strategies for computer problem solving," *Addison Wesley,* 1984.

[46] Reinefeld, A. and Marsland, T.A., "Enhanced iterative-deepmning search," *IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol.16, No.7,* 1994, pp.701–710.

[47] Sarkar, U.K., Chakrabarti, P.P., Ghose, S., and De Sarkar, S.C., "Reducing reexpansions in iterative-deepening search by controlling cutoff bounds," *Artificial Intelligence, Vol.50,* 1991, pp.207–221.

[48] Schaeffer, J., "The history heuristic and alpha-beta search enhancement in practice," *IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol.11, No.11,* 1989, pp.1203–1212.

[49] Schaeffer, J., "Conspiracy numbers," *Artificial Intelligence, Vol.43,* 1992, pp.67–84.

[50] Seo, M., "The C* Algorithm for AND/OR Tree Search," *SIGAI of Information Processing Society of Japan, 99-14,* 1995, pp.103–110.

[51] Slagle, J.R., and Bursky, P., "Experiments With a Multipurpose, Theorem-Proving Heuristic Program," *J. ACM, Vol.15, No.1,* 1968, pp.85–99.

[52] Slagle, J.R., and Dixon, J.K., "Experiments With Some Programs That Search Game Trees," *J. ACM, Vol.16, No.2,* 1969, pp.189–207.