A Survey of Tsume Shogi Programs Using Variable Depth Search

Reijer Grimbergen

Electrotechnical Laboratory,
1-1-4 Umezono, Tsukuba-shi, Ibaraki-ken, Japan 305
E-mail: grimberg@etl.go.jp

Abstract. Recently, a number of programs have been developed that successfully apply variable depth search to find solutions for mating problems in Japanese chess, called *tsume shogi*. Publications on this research have been mainly in Japanese. To present the findings of this research to a wider audience, we compare six different tsume programs. To find the solutions of difficult tsume shogi problems with solution sequences longer than 20 plies, we will see that variable depth search and hashing to deal with a combination of transposition, domination and simulation leads to strong tsume shogi programs that outperform human experts, both in speed and in the number of problems for which the solution can be found. The best program has been able to solve *Microcosmos*, a tsume shogi problem with a solution sequence of 1525 plies.
Keywords: Variable depth search, selective deepening, conspiracy number search, game playing, tsume shogi

1 Introduction

Most work on game-tree search has focused on algorithms that make the same decisions as full-width minimax search to a fixed depth [15]. Examples are alpha-beta pruning and $SSS^{\star}$ [30]. Human players do not use full-width fixed depth search, but a combination of shallow search and deep search [4]. A number of algorithms have been proposed that perform variable depth search. For example, *conspiracy numbers* [20, 27], *singular extensions* [3], *proof-number search* [1, 2] and *best-first minimax* [15] are all algorithms for searching game trees without explicit bounds on the search depth.

One of the domains where search to variable depths has been very successful but where this success has been almost unnoticed by the international AI community is *tsume shogi*. Tsume shogi are mating problems in Japanese chess. Since the early 90s, several tsume shogi programs have been developed that can quickly find the solution of problems with solution sequences of more than 50 plies. Both on the development of strong tsume shogi programs and the characteristics of the programs, there are a number of publications in Japanese ([24, 9, 17, 7, 10, 25, 6, 8, 29]). There have been a few English publications on tsume shogi, but only Seo [28] and Kawano [13] give a description of a program for finding

the solutions to difficult tsume shogi problems [1]. Some features of a tsume shogi program can be found in [22], but the description is very brief.

In this paper, we would like to present the results of tsume shogi research to a wider audience. In Section 2 a short explanation of the rules of tsume shogi are given, along with its history and the relevance for a shogi playing system. In Section 3 the computational features of a program to find solutions for tsume shogi problems are given. In Sections 4 to 9, a number of tsume shogi programs are described. Also, their results on different test sets of tsume shogi problems are summarized. We end with some conclusions and thoughts on future research on a general shogi playing system using methods discussed in this paper.

2 Tsume Shogi

2.1 Rules of Tsume Shogi

Tsume shogi are mating problems in Japanese chess. As far as the rules of Japanese chess are concerned, to understand the contents of this paper it is enough to know that shogi is similar to chess. The aim of the game is the same as in chess, namely the mating of the opponent's king. The shogi board is slightly bigger than the chess board, 9×9 instead of 8×8. Some pieces in shogi are the same as in chess, like the rook and the bishop, but some pieces are different. There is no queen in shogi, but instead there are golden generals, silver generals and lances. Promotion is also a little different. Most pieces can promote and can do so on any of the top three ranks of the board. The most important difference between shogi and chess is that in shogi captured pieces can be re-used. A piece captured becomes a *piece in hand* for the side that captured it. When it is a player's turn, he can either play a move with a piece on the board or put one of the pieces previously captured back on a vacant square on the board (this is called *dropping* a piece). It then becomes his own piece. Most drop moves are allowed, even dropping with check or mate is legal. Finally, it should be noted that in shogi the player to move first in the starting position is black and the other player is white (in chess this is the other way around). For a more detailed comparison of chess and shogi, see [19].

The rules of tsume shogi are simple. The goal of the attacking side (black) is to mate the king by consecutive checks; the goal of the defending side (white) is to reach a position where the attacking side has no checks. Therefore, the attacking side has to give check at every move and the defending side has to defend against these checks and prevent mate as long as possible.

2.2 History

Tsume shogi has a long history. The first tsume shogi problems date back to the 17th century [12] and the collection of tsume shogi problems that is still considered to be the most brilliant ever, has been published in 1755(!) and were

[1] Seo's work is described in a Master's thesis, so not easily available.

composed by Ito Kanju [11]. Also, there are many books with collections of tsume shogi problems and all shogi magazines have tsume shogi problem corners. There is even a monthly magazine called *Tsume Shogi Paradise* dedicated to tsume shogi. Of course, in tsume shogi problems that are published in shogi magazines the artistic element is very important, just as in chess problems. Recently, there have been attempts at automatically composing tsume shogi problems with artistically appealing features [5].

2.3 Relevance for Shogi Game Playing

Tsume shogi is not the same as perfect endgame play. It is possible that a game can end in fewer moves than by consecutive checks. However, tsume shogi is very important in the shogi endgame. In the endgame, the number of pieces in play is the same as on the first move. Furthermore, dropping pieces can have a major impact on the strength of attack or defense. Mate is the prime objective. Resignation because of material deficit is rare. Usually, a player resigns when he can no longer avoid mate, either because the opponent has started a tsume sequence (continuous checks leading to mate) or if there is no defense against such a tsume threat. The shogi endgame is a mating race, so finding mate and realizing that the opponent is threatening mate is vital for the endgame strength of a shogi player and also for a shogi playing program.

2.4 The First Tsume Shogi Program

The first tsume shogi program was built by Ochi in 1968 (described in [10]). Ochi's program has been reported to find the solution of tsume problems with a solution sequence of 9 to 11 plies as quickly and accurately as human players. Even though the program ran on one of the fastest computers of its time, this is quite an incredible result for a program that is 30 years old. We have been unable to find the original paper with a description of the program and the supporting data for this claim. In any case, for 25 years there was no breakthrough in the development of tsume shogi programs that made it possible to find the solution of problems with a solution of 15 plies or more. This changed with the introduction of algorithms for searching to variable depths in the early 90s.

3 Computational Features of Tsume Shogi

A correct tsume shogi problem should have only one solution. The search tree for a tsume shogi problem is an AND/OR tree, or a minimax tree where the evaluation of every node can have only two values, TRUE or FALSE. At each OR-node it is sufficient to find one check for which all the defenses lead to mate. If there is one check that leads to mate from the root, the problem is solved and the search can be stopped.

A tsume shogi problem to search an AND/OR tree has to deal with several problems:

– search deep for long mating sequences
– avoid redundant search
– recognize that positions have the same mating sequence
– decide which moves to search first

We will now describe each of these problems in detail.

3.1 Problem 1: Long Solution Sequences

Tsume shogi problems have a solution length ranging from 3 plies to hundreds of plies[2]. Seo [28] has experimentally found that the average branching factor of tsume shogi is only about 5. This is much smaller than the average branching factor of normal shogi play, which is about 80 [18]. However, even with this small branching factor, it is difficult to find the solution of tsume shogi problems with solutions of more than 15 plies by brute force methods. Currently the tsume shogi problem with the longest solution sequence is a problem called *Microcosmos*, which has a solution of 1525 plies. Finding long solutions of tsume shogi problems is the first challenge for tsume shogi programs.

3.2 Problem 2: Avoiding Redundant Search

To avoid searching the same position from different parts of the search tree, usually a standard transposition table is used. In shogi, not only transposition, but also *domination* can lead to redundant search. Domination is a concept that is used in every tsume shogi program, but it has not been properly defined in the literature. We define it as follows:

Definition 1 *A position P is* dominating *position Q if the board positions of P and Q are the same, and the pieces in black's (white's) hand of P are a proper superset of the pieces in black's (white's) hand of Q and it is black's (white's) turn to play in both P and Q.*

In tsume shogi, if P dominates Q regarding the attacker's pieces and a mating sequence has been found in Q then this mating sequence will also work in P. One way to check this, is to store extra information about the pieces in hand in the hash table.

3.3 Problem 3: Finding Simulating Positions

In chess, the number of checks and defenses against these checks is very limited. In tsume shogi the possibility of dropping pieces greatly increases the number of possible moves and makes the problem much harder. For example, in Diagram 1, the defending white king on 1a is checked by the black rook on 9a (top left

[2] A position where there is mate in one move is trivial and not considered a tsume shogi problem.

corner)[3]. The white pawn on 1b blocks the king's escape to 1b. A lance in shogi is a piece that moves like a rook but only in the forward direction. The black lance on 2i therefore blocks the escape of the king to 2b and also covers 2a. In a similar chess position, this position would be mate.

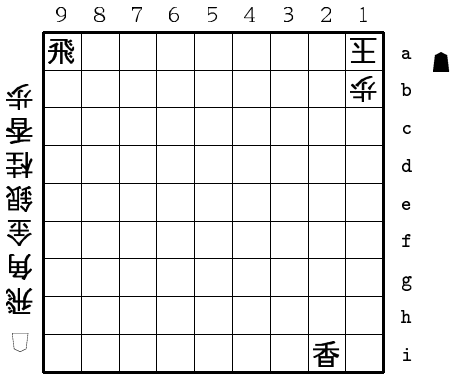

Diagram 1. An example of interpositions

However, in this shogi position the defense has all the pieces on the left side of the board in hand. The defending side can therefore drop any of these seven pieces between the checking rook and the king. Since there are seven vacant squares between rook and king and seven different pieces to drop, the number of interposing defenses is 49.

A rule of tsume shogi is that drops which do not change the solution sequence but only increase its length are not considered proper defenses. These useless interpositions are not counted as moves in the length of the solution sequence. Since all the interposing moves in Diagram 1 are useless (black can take any interposing piece immediately with the rook), this position is mate according to the rules of tsume shogi. For a tsume shogi program to recognize when drops are useless for check or defense with as little search as possible is a non-trivial task.

Another complication of tsume shogi is that in most cases promotion of pieces is optional. For tsume shogi, this usually leads to the same move sequence, since the moves of the promoted piece are in general only slightly different from the moves of the unpromoted piece. However, even in the case where the moves of a promoted piece are a superset of the moves of the unpromoted piece, there are some special cases where promotion of a piece does not lead to mate, while the non-promotion of a piece does. As a result, a tsume shogi program has to search the moves where a piece promotes and also the moves where the piece does not promote. Since in shogi a piece can promote on any of the top three

[3] The possibility of dropping pieces makes the total number of pieces in any shogi position the same (40). In Diagram 1 only the relevant pieces are shown.

ranks, this considerably increases the number of moves to be searched. A tsume shogi program should avoid redundant search for optional promotion moves as much as possible.

To deal with the problems of useless interpositions and optional promotion, the concept of *simulation* has been introduced. The general definition of simulation is [13]:

Definition 2 *A position P* simulates *position Q if all move sequences that can be played from position P can also be played from position Q.*

For example, the moves of a promoted rook are a superset of the moves of an unpromoted rook. If P is a position with an unpromoted attacking rook and Q is the same position with a promoted attacking rook, then P simulates Q. For tsume shogi, it would seem logical that if there is a mate in position P, there is also a mate in Q. However, in the previous section it was already mentioned that there are some special cases where a position with a promoted rook does not lead to mate, while a position with an unpromoted rook does.

Also, moves may have the same meaning even though they are not exactly the same. For example, a mate in position P is found and next position Q is searched where the only difference between the two positions is that the starting square of the rook is shifted one square to the left or right. This is a case where P might simulate Q. In these cases, it is natural to try the mating sequence in P first. However, since the starting square of the rook move differs, these positions do not simulate each other according to the strict definition given above.

Therefore, usually a more general, but heuristic concept of simulation is used in tsume shogi programs. Instead of searching for simulating positions where the move sequences *must* be the same, some shogi dependent knowledge is used to look for positions where the move sequences are *likely* to be the same. If the current position P might simulate a position Q based on these heuristics and a mate from P has been found, then the mating sequence found in P is tried first in Q. In tsume shogi programs the following heuristics are used for simulation:

- promotion vs. non-promotion
- different starting squares for the same long range piece (rook, bishop and lance)

3.4 Problem 4: Most Promising Moves First

The final challenge for a tsume shogi program is to guide the search for a solution in the right direction. Expert human tsume shogi solvers are very good at selecting promising candidate moves from a position. Usually the first or the second move considered in a position is the move leading to mate. To avoid wasting time by searching moves that are not likely to lead to mate, move ordering is very important.

Now the problems a tsume shogi program has to deal with have been discussed, we will give a description of six tsume shogi programs:

– Noshita's T1, T2 and T3 tsume shogi programs
– Ito's tsume shogi program
– Kawano's tsume shogi program
– Seo's tsume shogi program

These are the strong tsume shogi programs for which a detailed description of the methods used have been published. There are other strong tsume shogi solvers, but these are part of commercial shogi software and there have been no publications on them. Especially famous in this category is a program called Morita Shogi. Results of this program [21] show that it might be just as good as the programs described here. It would be very interesting to know if this program uses different methods for finding solutions in tsume shogi problems.

4 T1: Iterative Alpha-beta with Selective Deepening

4.1 Method

In 1991, Noshita's T1 program [24] became the first program to use selective deepening in combination with iterative alpha-beta to successfully find solutions of tsume shogi problems with solutions longer than 11 plies. T1 uses alpha-beta iterative deepening with only limited selective deepening. The selective deepening is based on the heuristic of measuring the freedom of the king. After each move, for each of the eight squares adjacent to the king it is calculated if the king can move to this square or not. If the king is very limited in its movement, the position is assumed to be close to mate and the search is extended for a maximum of 4 plies to try and find a mate.

T1 puts a lot of emphasis on move ordering. There are no less than 60 criteria for move ordering. Examples are: ordering promotions higher than non-promotions, preferring king moves away from the attacking pieces and ordering drops higher the closer they are dropped to the king. Noshita also used parameter learning to tune these move ordering criteria.

T1 has some heuristics to deal with transposition and domination, but has no hash tables. To avoid useless interposing drops, T1 uses the definition of the shogi programmer Kakinoki [16]:

Definition 3 *(i) An interposed piece is useless if capturing the piece results in a mating position. (ii) Suppose the king is mated in N moves in a certain position P without any interposing drop between checking piece and king. Then an interposed piece in P is useless if it is immediately captured, and there is a mating sequence of length N after the capture and the captured piece is not used in the mating sequence.*

In T1 this is applied as follows. If a piece is dropped between the checking piece and the king on square S and can be captured immediately, this might be a useless interposing drop. If capturing the piece results in mate, the drop was useless according to (i) and all other pieces that can be dropped on S are useless interposing drops and search can be stopped. If taking the dropped piece does

not give a mating position, but a move sequence is found leading to mate after the capture without using the captured piece, then the dropped piece on S was a useless interposing drop according to (ii) and search can be stopped.

4.2 Results

T1 was implemented on a 16 MHz PC98-RA21 (a Japanese PC) and on a SUN IPC workstation. Here we will summarize the results of three tests performed with the program [24]. One was a test set of 50 tsume shogi problems made by the professional shogi player Nakata Shodo. The length of the solution sequence was 7 to 15 plies. The performance of T1 was compared to that of human experts. T1 found the solutions of the entire problem set in about 27 minutes, while the best human expert could only solve 46 problems within a one hour time limit.

There are some unanswered questions with this test. One is that the original test set had 54 problems. It is unclear why only 50 of those problems were selected. Also, it is unclear how many human subjects there were and what the exact test conditions were.

The second test given to T1 was a set of 32 difficult tsume shogi problems from a shogi magazine. The solution length of the problems was 7 to 9 plies. T1 found the solution of all the problems with an average solution time per problem of 55 seconds. According to the scoring table attached to the tsume problems, this is the highest possible level: "professional strength". Of course, the solutions of these problems are not long enough to really test the program. After all, the old program by Ochi mentioned before already claimed the same performance.

The final test for T1 were eight 15 ply tsume shogi problems and eight 17 ply tsume shogi problems. The 15 ply tsume problems were solved in 130 seconds on average, while the 17 ply tsume problems were solved in about 10 minutes on average.

Noshita's own conclusion is that T1 works well for tsume shogi problems with a solution sequence up to 17 plies and is able to find the solution of some problems with solutions that are longer than 30 plies. However, for most of the problems with a solution longer than 17 plies, T1 is too slow to find the solution within a reasonable time limit.

5 T2: T1 Plus Hash Tables

5.1 Method

T2 is also made by Noshita and an improved version of T1, using similar ideas, which are described in [9, 25]. One improvement in T2 over T1 is that the heuristics for king safety are better. However, the big difference between the two programs is the introduction of hash tables for transposition and simulation. The search for useless interposing pieces is now done differently. If a possible useless interposition is detected, the piece is taken and given back to the defending side. If there is a mate in the resulting position, then the piece was indeed a useless

interposition and all moves with other pieces dropped in defense on the same square need not be searched. Here hashing is very useful if the position with the capturing piece closer to the king is searched first and stored in the hash table. For tsume shogi the introduction of hash tables can make a big difference. Noshita gave the position of Diagram 1 before the check with the rook on 9a (black rook on 9i instead of 9a) to both T1 and T2. T1 had to search 1,500,000 positions to search through all the interposing drops and find a mate, while T2 only needed to search 25,000 positions.

5.2 Results

T2 runs on a SUN SPARC IPX with 28 MB of memory. For T2 a different test set of 3 to 9 ply tsume problems was used. There were 112 problems in the test set. T2 found the solutions to the whole problem set in 1 minute. According to the author of these tsume shogi problems, solving all problems in less than 70 minutes could be considered to be a "professional" performance. Again, the solution lengths of these test problems are too short to really test the program.

T2 has also been given 25 problems with solutions of 19 to 25 plies. T2 could solve all these problems, but needs more than 15 minutes per problem for the 23 and 25 ply tsume problems.

Finally, T2 has been given the problems in the classic book *Zoku-tsumuya-tsumazaruya* ("Mate or no mate?") [12]. In this collection the tsume problem with the shortest solution has a solution sequence of 11 plies and the longest problem has a solution of 873 plies. Even though the collection officially has 200 tsume problems, the actual number of problems is 195. A few problems are not tsume problems and a few problems have no solution because a defense against which no mate can be found was overlooked by the composer.

T2 found the solution of 70 of these hard problems. It is unclear what the maximum time per problem was, but the graphs in [9] seem to suggest more than two and a half hours.

6 Ito's Tsume Program: Selective Deepening by Best-first Search

6.1 Method

Ito's program was the first to use selective deepening from the root of the search tree with best-first search [9, 6]. It assigns the following numbers to the nodes of the search tree:

- Mate: 0
- No possible checks: ∞
- Leaf node: KingFreedom
- AND-node: $\sum KingFreedom_{children(n)}$
- OR-node: $\min KingFreedom_{children(n)}$

The next node to expand is the node where the freedom of the king is minimal. If multiple nodes have the same minimal value, one node is chosen randomly. There are no depth limits to the search. Transposition and simulation is dealt with in the same way as in T1 and T2.

6.2 Results

Ito's program runs on a Sparc Station 10. It was given the same tests as T2. It took 2.5 minutes to find the solutions of the 112 easy problems. Of the 25 problems of 19 to 25 plies solutions depth, Ito's program could solve 19. It was much quicker than T2 in solving the 23 and 25 ply tsume problems, using only 100 seconds on average.

Ito's program was much better than T2 in solving the hard problems of Zoku-tsumuya-tsumazaruya. It could solve 120 problems.

The collection Zoku-tsumuya-tsumazaruya has a major drawback: no less than 25 problems have been shown to be incorrect, namely having more than one solution. In all but one case there was a shorter mate than the one intended by the composer of the problem. In one case there was a defense which had a solution that was 9 plies longer than the intended solution. The collection is therefore easier than the number of moves of the intended solutions suggest.

A better test set is *Tsume Zuko*, the masterpieces of the tsume composer Ito Kanju (1719-1761). These 100 tsume problems were published in 1755 and are still considered to be among the best problems ever made. In the set of 100 problems with a problem length of 9 plies to 611 plies (average length 42 plies) there are only two problems for which there is a shorter solution than the one intended. For these two problems there are repaired versions available with only small changes in the position and the solution as intended by Ito Kanju. Ito's program could find the solutions of 63 of these 100 problems [28].

Ito's program could not find the solutions of short tsume problems as well as T2, but it was a major step forward in deep search. However, like most selective deepening approaches, it suffered from memory problems when searching very deep, since too big a portion of the search tree had to be kept in memory. For example, Ito's program was not able to find the solution of the 611 ply *Kotobuki*[4] problem with its normal memory management. Only when the memory management was replaced by a scheme that freed large portions of the search tree that were unlikely to lead to mate, Ito's program could solve Kotobuki. However, it took 70 hours to find the solution, indicating that too many parts of the search tree had to be regenerated.

[4] Kotobuki means "Long life" and is the final problem in the Tsume Zuko problem set.

7 T3: AO* for Tsume Shogi

7.1 Method

T3 is the third tsume shogi program by Noshita. The methods used in the program are described in [10][5]. T3 is based on AO* [23] and therefore very different from T1 and T2. There is a priority queue of leaf nodes which are stored with their expectancy of mate. This value is again based on the freedom of the king, but in T3 also the mating expectancy of several ancestor nodes of the leaf nodes is used in the mate expectancy value of the leaf nodes [26]. The node that is at the top of the queue is the node to be expanded next.

The second part of the T3 algorithm is the serialization of AND-nodes. Since all AND-nodes need to be TRUE, only one of the children needs to be expanded. All other nodes are only expanded if the active node returns a TRUE value. OR-nodes are expanded in the normal way. An enhancement of this scheme used in T3 is to use a limited alpha-beta search to look for a quick mate after an AND-node is being expanded. This keeps the mating search in a local promising area as long as possible.

The third important concept in T3 is a different way of dealing with transposition, domination and simulation. T3 has a set of linked lists that connect nodes for which the mating value is related. Only the node on which all other nodes in the linked list depend is considered for expansion. All other nodes are *frozen* until this one node returns either a mating or a non-mating value. Then the other nodes in the list are revived and the mating sequence of the solved node is tried in each of the other nodes in the linked list. As a result, the search tree is no longer a proper tree, but an AND/OR graph.

The data structure (numerous sets of linked lists) necessary for this best-first search scheme is large. T3 keeps as much of it as possible in memory. However, discarding part of the search tree and the connected linked lists cannot be avoided when searching for tsume shogi problems with very long solutions. Regenerating too many nodes will seriously slow down the program, so the speed of the program depends very much on the quality of the garbage collection. Still, connecting nodes in this way uses less memory than normal hash tables. This is because in shogi the concepts of domination and simulation make it necessary to store extra information in the hash table about pieces in hand.

7.2 Results

T3 ran on the same machine as T2, a SUN SPARC IPX with 28 MB of memory. The only published test results of T3 are on the Tsume Zuko problems. T3 could find the solution of 68 of these. Kotobuki is solved in 65 hours.

[5] I have not found a separate publication on T3. Details about the program have been taken from Japanese papers describing different programs and co-authored by several tsume shogi programmers.

8 Kawano's Tsume Program: Priority and Simulation

8.1 Method

Kawano's tsume program [13] is based on best-first search. Each node is given a *priority*. At the leaf nodes, the priority is the same as the move ordering according to the criteria given for Noshita's programs. However, this move ordering is only done for moves by the defending side. This local ordering is generalized into a global ordering of the tree by giving the attacker's move the priority of its parent node and making the move with the highest local priority equal to the priority of the parent node. This gives an ordering of the nodes in the game tree and the leaf node to be expanded next.

Kawano deals with the problem of possible simulation by defining a choice function for the moves of the attacking side. This choice function defines when two moves are the same even though they might be textually different. Simulation is now defined as follows [13]:

Definition 4 *Position P simulates position Q if the move sequences* defined by the choice function *that can be played from P are the same as the move sequences* defined by the choice function *that can be played from Q.*

The choice function is only defined for the moves of the attacking side, so the definition of this function does not affect the outcome of the search. If two moves are defined by the choice function to be the same, but are actually different (i.e. the mating sequence of P does not work in Q), the different defense moves at the next move will show that the two positions are not simulating each other. The choice function is therefore a shogi heuristic used to guide the search in a more promising direction. The choice function defines moves with promoted pieces the same as moves with unpromoted pieces and moves with pieces from different starting squares as the same (see [13] for more details).

8.2 Results

Kawano's tsume program runs on a Dec Alpha 400 MHz with 1GB of memory. Like for T3, there are only reported results for the Tsume Zuko problems. Kawano's program can find the solution of 88 of these problems [14]. The maximum time per problem was 2 hours. In Table 1 it can be seen that 81 of these problems are solved within a 100 seconds. The solution of the Kotobuki problem is found in 6 minutes and 46 seconds.

9 Seo's Tsume Program: Conspiracy Numbers

9.1 Method

Seo's tsume program [28, 29] uses an algorithms called C^*. This name is not explained but probably chosen because the algorithm is based on AO* but uses

Table 1. Results of Kawano's program on Tsume Zuko

CPU time (s)	Solved
0-1	39
1-10	30
10-100	12
100-1000	6
1000-7200	1
Total	88

conspiracy numbers to guide the search. In conspiracy number search a heuristic value is assigned to each node in the search tree. The conspiracy number of a node N is the minimal number of leaf nodes in the subtree of N that need to change their value in order to change the minimax value of N. Nodes in the tree are then expanded in such a way that they will narrow the range of possible root values. For the AND/OR trees of tsume shogi, there are only three values for every node: TRUE (mate), FALSE (no mate) and UNKNOWN. Therefore, in tsume shogi we have the following rules of assignment of conspiracy numbers (CN) to a node n:

- Mate node: $CN(n) = 0$
- Node with no possible checks: $CN(n) = \infty$
- Leaf node: $CN(n) = 1$
- AND-node: $CN(n) = \sum CN_{children(n)}$
- OR-node: $CN(n) = \min CN_{children(n)}$

To solve the problem of memory, Seo uses a depth-first iterative deepening approach. First, find a solution for a conspiracy number threshold of 1 for every node. Then, if no solution is found, set the threshold to 2 and so on until a solution is found for conspiracy number threshold n. In general, this iterative approach would result in too many regenerated nodes. To keep this to a minimum, Seo uses big hash tables to store as many positions as possible with their conspiracy number. If a position is searched again at iteration p, the conspiracy number of the node is initialized to the conspiracy number in the hash table, which is a value smaller or equal to $p - 1$ (Seo calls this *dynamic evaluation*). Also, to avoid regenerating nodes as much as possible, Seo's tsume program gives priority to the nodes that have not been expanded at previous iterations. Seo has been able to keep the average number of regenerated nodes at about 20%.

Seo also uses move ordering and looks for transposition, domination and simulation. Interposing drops closer to the king higher are ordered higher than interposing drops further from the king like in most of the other programs. Simulation is used by Seo to try the same mating sequences in positions that only differ in the interposed pieces and in cases where promotion is optional. He calls this a *killer heuristic*, similar to the concept used in chess programs.

9.2 Results

The first version of Seo's program ran on a Sun Sparc Station 20 workstation. Seo's results are very impressive. His program can find the solution of 190 of 195 problems in the Zoku-tsumuya-tsumazaruya test set and 99 out of the 100 problems in Tsume Zuko, also using a maximum of two hours per problem. Detailed results on solution speed can be found in Table 2. The Kotobuki problem was solved in 1 hour and 12 minutes. Seo's program is slower than that of Kawano, but it is more powerful in that it can find the solutions of more tsume shogi problems with long solutions.

The only problem in the Tsume Zuko test set that Seo's program could not find a solution for was a very complicated 41 ply tsume problem with an unusually high number of long side variations leading to mate. As a result, the solution subtree has a very high conspiracy number throughout the search so there are only few node expansions in that vital part of the tree.

Table 2. Results of Seo's program

CPU time (s)	ZokuTT	Tsume Zuko
0-1	21	2
1-10	44	23
10-100	68	39
100-1000	39	29
1000-7200	18	6
Total	190	99

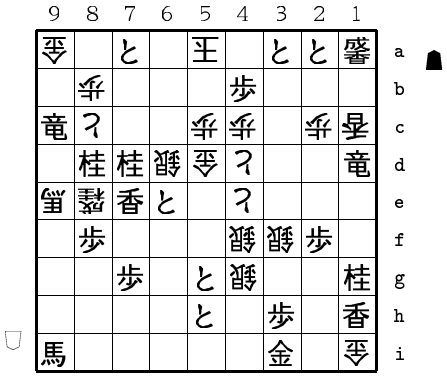

Diagram 2. Microcosmos

The holy grail for tsume shogi programs is the Microcosmos problem (Diagram 2). This problem was composed by Hashimoto in 1986 and has a solution length of 1525 plies. Since the publication of his master's thesis in 1995, Seo has improved his program and ported it to a 166 MHz Pentium with 256 MB memory. This new version was able to solve Microcosmos in about 30 hours in April 1997.

10 Conclusions and Further Research

In this paper we have discussed the following features of a good tsume shogi program:

- an algorithm that can search deeply to variable depths
- hash tables to not only deal with transposition, but also with domination and simulation
- move ordering based on freedom of the king

Table 3. Results of all programs on the two hard test sets

Program	Author	ZokuTT	Tsume Zuko	Year
T1	Noshita	-	-	1991
T2	Noshita	70	-	1992
Ito	Ito	135	63	1992
T3	Noshita	-	68	1992
Kawano	Kawano	-	88	1994
Seo	Seo	190	99	1995

In Table 3 the results on the two major test sets for the programs discussed in this paper are summarized. It is not easy to compare the performance of the programs, since they are running on very different platforms. However, two things are clear from this table and from the data summarized in the previous sections. One is that most of the current tsume shogi programs perform better than human experts. The two tsume shogi collections are considered to be very hard and the general opinion among shogi players is that no human player will be able to solve more than 80% of these problems. The recognition of the performance of the tsume shogi programs is further supported by the fact that shogi magazines these days use tsume shogi programs to aid in the analysis of difficult endgames played by top professional players.

The second conclusion is that Seo's tsume program is clearly the best of the tsume programs discussed in this paper. Since different hardware is used, comparing Ito, T3 and Kawano's tsume program is almost impossible. Especially Kawano's tsume program is running on one of the fastest and biggest machines currently available. It would be interesting to test T3 again on such a machine.

However, even with its extra computing power, Kawano's program performs worse than Seo's program as far as the number of problems that can be solved is concerned, even though Seo's program is running on slower hardware. Seo's program is able to find the solution of almost any tsume problem and it will be hard to make a program to improve it. Still, even though Microcosmos has been solved, there remain a number of other long tsume shogi problems for which Seo's program cannot find the solution in a limited time.

One of the possible improvements of Seo's program might be to use proof-number search instead of conspiracy number search. Proof-number search is an improved version of conspiracy number search designed especially to solve AND/OR trees [2]. We have started to develop a tsume shogi program for our shogi playing system SPEAR based on proof-number search. Although work on this tsume shogi program has not been finished yet, preliminary results show that for a significant number of test problems, smaller search trees are built than in Seo's program. Improving the tsume shogi program in SPEAR is a future work.

It is interesting that complexity of a tsume shogi problem cannot be defined by the length of the solution sequence. For the 17 problems in the range from 9 plies to 19 plies, Seo's program on average needed 150 seconds per problem, while for the 18 problems from 31 plies to 39 plies, Seo's program took only 82 seconds per problem. Furthermore, there was a relatively short 23 ply tsume shogi problem in the Tsume Zuko test set that could not be solved by Kawano and for which Seo took almost 1.5 hours. Variable depth search is clearly having problems with other features of a tsume problem. For human players also, the difficulty of a tsume problem is not necessarily related to the length of the solution. It would be interesting to see if there is a correlation between the difficulty of tsume shogi problems for variable depth search and human experts.

Tsume shogi no longer seems to be a hard problem. In this paper we have discussed methods for building a strong tsume program. However, there is still a good number of problems for which the strongest programs cannot find a solution. Furthermore, the time limit of two hours for the problems in the test set is long, even though it has become the standard time limit for most tests with the programs discussed. Algorithmic improvements to get the same results with a stricter time limit are another challenge left for tsume shogi programs.

The important question is of course how these successful results can be used outside the domain of tsume shogi. After all, tsume shogi is only important in the final stages of a shogi game. Can the techniques discussed in this paper help in building a strong shogi playing program? For the time being, this remains an open question. As said, the branching factor of normal shogi (80) is much larger than the branching factor of tsume shogi (5). The size of the search tree for a 40 ply tsume shogi problem is therefore about the same as a 15 ply search in a normal shogi position. Such a deep search is not out of reach for tsume shogi programs, so the methods discussed in this paper might be interesting for a normal shogi playing program as well. Conspiracy number search has been shown to be applicable to normal minimax game trees. One big problem is of course to set the correct search target which is trivial in tsume shogi. This

problem is illustrated by Seo, who has made a shogi program based on his work in tsume shogi. His program thus far cannot compete with the strongest programs, despite its obvious strength in tsume shogi. We believe that variable depth search algorithms are worth further investigating for shogi and we intend to develop a shogi program based on these ideas in the future.

Acknowledgements

I would like to thank Hitoshi Matsubara for helping me understanding the Japanese papers on tsume shogi and Kohei Noshita for his patient explanations of his tsume shogi programs.

References

1. L.V. Allis. *Searching for Solutions in Games and Artificial Intelligence.* PhD thesis, The Netherlands: University of Limburg, 1994. ISBN 90-9007488-0.
2. L.V. Allis, M. van der Meulen, and H.J. van den Herik. Proof-number search. *Artificial Intelligence*, 66:91–124, 1994.
3. T. Anantharaman, M.S. Campbell, and F. Hsu. Singular extensions: Adding selectivity to brute-force searching. *Artificial Intelligence*, 43:99–109, 1990.
4. A.D. de Groot. *Thought and Choice in Chess.* The Hague, The Netherlands: Mouton & Co, 1965.
5. M. Hirose, H. Matsubara, and T. Ito. The composition of tsume-shogi problems. In *Advances in Computer Chess 8*, pages 299–319, Maastricht, Holland, 1996. ISBN 9062162347.
6. K. Ito. Best-first search to solve tsume shogi problems. In H. Matsubara, editor, *Computer Shogi Progress*, pages 71–89. Tokyo: Kyoritsu Shuppan Co, 1996. ISBN 4-320-02799-X. (In Japanese).
7. K. Ito, Y. Kawano, and K. Noshita. On the algorithms for solving tsume-shogi with extremely long solution-steps. *Journal of the Information Processing Society of Japan*, 36(12):2793–2799, 1995. (In Japanese).
8. K. Ito, Y. Kawano, M. Seo, and K. Noshita. Tsume shogi. In H. Matsubara and I. Takeuchi, editors, *BIT special issue: Game Programming*, pages 130–138. Kyoritsu Shuppan Co., Tokyo, Japan, 1997. ISBN 00110-2-57035. (In Japanese).
9. K. Ito and K. Noshita. Two fast programs for solving tsume-shogi and their evaluation. *Journal of the Information Processing Society of Japan*, 35(8):1531–1539, 1994. (In Japanese).
10. T. Ito, Y. Kawano, M. Seo, and K. Noshita. Recent progress in solving tsume-shogi by computers. *Journal of the Japanese Society of Artificial Intelligence*, 10(6):853–859, 1995. (In Japanese).
11. Y. Kadowaki. *Tsumuya-tsumazaruya, Shogi-Muso, Shogi-Zuko.* Tokyo: Heibon-sha, 1975. ISBN 4-582-80282-0. (in Japanese).
12. Y. Kadowaki. *Zoku-tsumuya-tsumazaruya.* Tokyo: Heibon-sha, 1978. ISBN 4-582-80335-0. (in Japanese).
13. Y. Kawano. Using similar positions to search game trees. In R.J. Nowakowski, editor, *Games of no chance (Combinatorial games at MSRI, Berkeley 1994)*, pages 193–202. Cambridge: University Press, 1996.

14. Y. Kawano. Personal communication, 1998.
15. R.E. Korf and D.M. Chickering. Best-first minimax search. *Artificial Intelligence*, 84:299–337, 1996.
16. Y. Kotani, T. Yoshikawa, Y. Kakinoki, and K. Morita. *Computer Shogi.* Tokyo: Saiensu-sha, 1990. (in Japanese.).
17. H. Matsubara. *Shogi to computer (Shogi and computers).* Kyoritsu Shuppan Co., Tokyo, Japan, 1994. ISBN 4-320-02681-0. (In Japanese).
18. H. Matsubara and K. Handa. Some properties of shogi as a game. *Proceedings of Artificial Intelligence*, 96(3):21–30, 1994. (In Japanese).
19. H. Matsubara, H. Iida, and R. Grimbergen. Natural developments in game research. *ICCA Journal*, 19(2):103–112, June 1996.
20. D.A. McAllester. Conspiracy numbers for min-max search. *Artificial Intelligence*, 35:287–310, 1988.
21. K. Morita. Personal communication, 1996.
22. Y. Nakayama, T. Akazawa, and K. Noshita. A parallel algorithm for solving hard tsume shogi problems. *ICCA Journal*, 19(2):94–99, June 1996.
23. N. Nilsson. *Problem Solving Methods in Artificial Intelligence.* New York: McGraw-Hill, 1971.
24. K. Noshita. How to make a quick and accurate tsume-shogi solver. *Shingaku-Kenkyukai, COMP91*, 56:29–37, 1991. (In Japanese).
25. K. Noshita. The tsume shogi solver T2. In H. Matsubara, editor, *Computer Shogi Progress*, pages 50–70. Tokyo: Kyoritsu Shuppan Co, 1996. ISBN 4-320-02799-X. (In Japanese).
26. K. Noshita. Personal communication, 1998.
27. J. Schaeffer. Conspiracy numbers. *Artificial Intelligence*, 43:67–84, 1990.
28. M. Seo. The C* algorithm for and/or tree search and its application to a tsume-shogi program. Master's thesis, Faculty of Science, University of Tokyo, 1995.
29. M. Seo. A tsume shogi solver using conspiracy numbers. In H. Matsubara, editor, *Computer Shogi Progress 2*, pages 1–21. Tokyo: Kyoritsu Shuppan Co, 1998. ISBN 4-320-02799-X. (In Japanese).
30. G. Stockman. A minimax algorithm better than alpha-beta? *Artificial Intelligence*, 12:179–196, 1979.