

TREE ANALYSIS TECHNIQUES IN TSUMEGO

David B. Benson, Bruce R. Hilditch, J. Denbigh Starkey
Computer Science Department
Washington State University
Pullman, Washington 99164 USA

The tsumego program performs extensive node analysis at the upper tree levels instead of attempting maximum node visitation. The success rate of the program is tested against a published set of problems of known difficulty.

1. INTRODUCTION

We use a tree searching technique of performing a great deal of analysis per node in order to choose superior lines for subsequent play. This idea is thought to be especially valuable for the chosen problem area, tsumego problems. Tsumego problems are restricted-size situations in Go that determine a local, tactical result. The most interesting feature for game-playing is the peculiar shape of the game trees. The number of level 1 branches of the tree is approximately equal to the depth, and the branching factor declines fairly regularly with the search depth (n at level 1, $n-1$ at level 2, etc. where n is the initial number of possible plays). This implies that high-level pruning is especially effective. Doing considerable analysis per node, as shown by exercising the program on a problem set, provides enough information quickly enough to make good decisions on choosing the line of play.

2. THE GAME

2.1 Go.

The strategy and rules of Go have been concisely and accurately described by Iwamoto [4]. Some aspects of programming the game have been treated by Benson [1] [2], Reitman & Wilcox [5] [6], Ryder [7], Thorp & Walden [9] [10], Zobrist [11], and others but the fundamental strategic and tactical heuristics used in human-quality play are still essentially undiscovered.

2.2 Tsumego.

The main threat used in deterring intrusions into one's rightful territory is the threat to capture the invading stones. It is thus frequently necessary to evaluate small sections of the board to determine whether a group of stones will be safe from capture. This can be accomplished by connecting to another safe group or by gaining 'life' on one's own merit. Since the connection-to-another-group problem usually depends on global game information concerning threats, book

problems usually deal with local, tactically-oriented life-or-death situations. These problems generally resolve themselves into two subclasses: first, a 'race-to-capture' where it is necessary to capture the opponent's stones before he succeeds in capturing your own stones. Second, sequences of play that attempt to form the two protected empty intersections (eyes) necessary to be safe from attack. Strategic concerns normally depend on the outcome of the tsumego battle but do not greatly influence its tactics. Thus we have an important but independent segment of the game which can be treated separately.

3. THE PROGRAM

3.1 The program's purpose

The tsumego program is designed to be one of several service routines callable by a driver. The functions of each of these subprograms (tsumego, semeai, fuseki, chuban, yose) will be closely defined and specific. The driver program has the responsibility of developing and maintaining a coherent strategy for the various phases of the game (opening, midgame, endgame) and of calling the service routines, as necessary, for answers to the queries which it develops. For the tsumego routine, input consists of the board configuration containing the groups in question, a set of control information (who plays first, limits on computation resources, etc.) and a list of desired results i.e., the specified group shall live, die, connect out, or live in seki. Life-in-ko is not considered to be a distinct end-point condition. Output consists of a binary result (YES implies that a desired result can be achieved or NO, it cannot) and, if YES, the answer includes a forcing play-tree leading to the result. Other statistics are gathered to allow us to quantitatively measure the suitability of heuristics for different classes of problems.

3.2 The program's method

The heart of the tsumego program is a heuris-

tic algorithm which generates only the 'good' plays. These plays of course are the heuristic's ideas of good plays and human players often take a different view.

Since constructing a 'live' group requires the formation of two simultaneously existing eyes that are both adjacent to the specified group, the program calculates all points which might become an eye. That is, it looks for unoccupied points that have all four of the adjacent points vacant or occupied by friendly stones and at least 3 of the 4 "shoulder" points vacant or occupied by friendly stones. The program then constructs a list of all pairs of these points which might form a pair of connected eyes. Note that if two possible eye points are adjacent then both points cannot simultaneously be eyes and thus would be excluded from the list of pairs of possible eyes. Construction of the list is basically derived from a transitive closure algorithm with additional code meant to discard the inviable pairs. For all pairs which make the list, the program then calculates the plays necessary to complete and connect the two eyes. After a slight, artificial increase in the frequency counts of certain threatening and capturing plays, the program orders the resulting list by the frequency of occurrence of the potential plays. The list thus represents an ordered set of plays that may produce a live group. The plays at the front of the list and therefore those that are considered first by the recursive game player are those which appear more frequently in board configurations featuring live groups. The list is constructed at each level in the game tree since the board configuration changes at each level as new plays are made.

If the program is on 'defense', its play generation task is done. If it is on 'offense', we rely on the well-known Go heuristic: "my opponent's best play is often my own best play." This tactic leads to occasional wildly aggressive offense and often to a rather passive defense (typical beginner habits) but it is remarkably effective for several types of tsumego problems. It is remarkably ineffective for several other types.

Another module recognizes 'unconditional life' as proposed by Benson [1]. Unconditional life is a condition where an attacker is allowed an unlimited number of consecutive plays and is still unable to destroy the unconditionally alive group(s). This concept has been extended to also recognize 'unconditional death' where the defender is allowed an unlimited number of plays but is never able to produce a live group. The seki condition is also treated but this is not a static condition in the sense of the previous two ending conditions since a seki group might be sacrificed in return for a threat elsewhere. Additionally, the seki condition (a sort of half-life, half-death) occurs much

less frequently than the other two, unconditional, ending conditions.

4. TESTING AND RESULTS

The tsumego problems used in testing our program were taken randomly from a collection of such problems by Davies [3]. The reasons for choosing this set of problems was simple: they come from published material and represent problems that humans solve. They were not composed especially for computer solution; their difficulty is easily rated; and we thought them to be within the capability of a specialized heuristic. The problems were modified slightly to restrict their size for economic reasons but we were careful to maintain all possibilities of attack, defense and counter-attack. As soon as the next module, a semeai-player, is incorporated into the program, these modifications can be removed since the program will be self-limiting.

The following two figures illustrate in a simple way the type of Go problem we are treating and the program's resulting search tree.

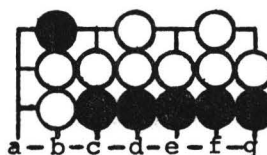


Figure 1: A simple tsumego problem

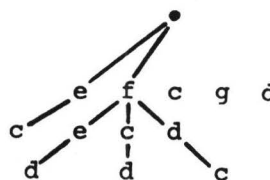


Figure 2: The corresponding computed game tree for Figure 1.

In Figure 2, note the decrease in branching factor as depth increases in the tree search for the sample problem. This could be expected since stones do not move, captures are infrequent and so each play reduces the number of possible continuations. If the pruning is heavy enough and early enough we are able to dispense with immense amounts of lookahead because of the top-heavy nature of

the full game tree. Table 1 gives tabular results for the complete set of problems tested.

TABLE 1

Prob. no.	To play	Result	Init. plays	Max. ply	#leaves visited
p15#1	white	yes	5	0	1
p15#1	black	yes	7	0	1
p19#2	black	yes	5	3	4
p19#4	white	no-0	13	9	11
p23#5	white	yes	14	3	7
p23#5	black	yes	11	6	8
p27#3	white	no-1	15	0	1
p27#3	black	no-1	15	0	1
p31#5	white	no-2	23	-	-
p31#5	black	no-2	21	-	-
p35#1	black	yes	13	22	263
p39#5	white	no-2	17	-	-
p39#5	black	yes	18	17	285
p43#2	white	no-1	15	0	1
p43#2	black	no-1	18	0	1
p47#2	white	yes	13	4	9
p47#2	black	no-0	14	9	38
p51#3	white	no-1	21	0	1
p55#1	black	yes	13	11	145
p58#1	white	no-2	19	-	-
p63#1	white	no-1	16	0	1
p67#4	white	no-2	22	-	-
p71#3	black	no-2	21	-	-

Notes:

- 0: heuristic failed... gave wrong binary result. We are counting this as a wrong answer. Both wrong answers were the result of a bug in the seki recognition routine.
 1: program returned a message saying the problem was unsuitable for the eye-stealing heuristic. We are not counting this result as a wrong answer since another local expert routine will presumably eventually be available to take over.
 2: program was making progress but exceeded allotted resources. This indicates that the

single heuristic is not suitable for this class of problem. Additional local-expert subroutines will have to be included to extend the range of the program's capabilities.

5. CONCLUSIONS

Bushy, top-heavy game trees where the number of future plays is inversely proportional to the tree depth suggest the need for heavy, early pruning. The strategy of performing a full-width search to a pre-set depth (Slate & Atkin [8]) would in fact search a considerable portion of the game tree. In addition, the complexity and diversity of patterns in Go make the development of effective evaluation functions difficult. Chess-type strategies which rely on a slow build-up of information are inappropriate for tsumego.

As an alternative, we provide enough heuristic analysis at high tree levels to provide information necessary for effective play ordering and then let a branch-and-bound technique severely prune the tree. This procedure has worked well for our tsumego program (9 correct vs. 2 incorrect). There is considerable pruning as evidenced by realized branching factors (<2 at upper levels) in the tested problems.

To date the program includes an eye-forming/eye-stealing heuristic as the play generator. The heuristic is effective for solving tsumego problems of the type found in the first five chapters of Davies [3]. The program is handicapped when solving other classes of problems, particularly problems involving race-to-capture (semeai) and play-under-the-stones (ishi-no-shida). Its proficiency for tsumego problems might generously be rated as the same as a human player in the 20-25 kyu range. Any increased capability of the program will be measured objectively by attempts to solve published problems.

REFERENCES

- [1] Benson, D., 'Life in the Game of Go', Information Sciences, 10:1 (1976) 17-29.
- [2] Benson, D.; Evers, P.; Miller, J.; Tackett, M.; Starkey, D.; 'Computerizing the Game of Go', In Proc. Northwest 76: ACM Pacific Regional Symposium, ACM, New York, 1976.
- [3] Davies, J. Life and Death, Tokyo: Ishi Press, 1975.
- [4] Iwamoto, K. Go for Beginners, Tokyo: Ishi Press, 1973.
- [5] Reitman, W.; Wilcox, B.; Nado, R., 'Goals and Plans in a Program for Playing Go.' In Proc. of the ACM Annual Conf., ACM, New York, 1974, pps 123-127.
- [6] Reitman, W.; Wilcox, B. 'Perception and Representation of Spatial Relations in a Program for Playing Go.' In Proc. of the ACM Annual Conf. ACM, New York, 1975, pps 37-41.
- [7] Ryder, J. 'Heuristic Analysis of Large Trees as Generated in the Game of Go.' PhD thesis, Computer Science Dept., Stanford University, 1971.
- [8] Slate, D.; Atkin, L. 'Chess 4.5: The Northwestern University Chess Program.' In Chess Skill in Man and Machine, P. Frey, ed., Springer-Verlag, New York, 1977.
- [9] Thorp, E.; Walden, W. 'A Computer Assisted Study of Go on MxN Boards', Information Sciences, 4:1 (1972), 1-33.
- [10] Thorp, E.; Walden, W. 'A Partial Analysis of Go', Computer Journal, 7:3 (1964), pps 203-207.
- [11] Zobrist, A. 'A Model of Visual Organisation for the Game of Go', PhD thesis, Computer Science Dept., University of Wisconsin, 1973.