Metarules to Improve Tactical Go Knowledge

Tristan Cazenave
Labo IA, Université Paris 8
2 rue de la Liberté, 93526, St-Denis, France
e-mail: cazenave@ai.univ-paris8.fr

Abstract

Three main problems arise with automatically generated rules databases. They are too large to fit in memory, they can take a lot of time to generate, and it takes time to match many rules on a board. I propose to use exceptions and metarules to reduce the size of the databases. The reduction in size enables larger rules to be used, at the cost of small overheads in generation and matching times. However, the reduction in search depth provided by the new larger rules decreases much the overall search time, stopping search at smaller depths.

1 Introduction

I have written an automatic rule generator based on retrograde analysis of patterns with external conditions [3],[5],[2]. It generates rules about eyes and life in the game of Go. Life and death in the game of go has been already studied, and some clever algorithms have been designed. Beginning with Benson's algorithm [1] that detects unconditional life with the opponent moving as many times he wants to and still being unable to kill. More heuristic approaches have followed such as in Star of Poland [14] or Go Intellect [9],[10]. The usual way to approach the problem is to write an elaborate static life and death evaluation function [11], and to use a search algorithm based on it such as in Gotools [16],[17],[18],[19],[20].

Related works on retrograde analysis and databases include D. Dyer's shapes database [12] that enumerates completely enclosed living shapes, and R. Gasser's work on finding exceptions in Nine Men's Morris endgame databases [13].

The second section describes the generations of rules for eyes and life and death. The third section explains different metarules that can be used to reduce the number of rules, including exceptions rules. The fourth section gives some experimental results on the generated patterns databases. The fifth section underlines future work.

2 Automatic generation of rules

A rule is composed of a rectangular pattern, and of some associated conditions on liberties external to the pattern. A rule can conclude on the life of strings, or on the eye potential of strings. There are two type of rules: rules that conclude on a won state (the goal can always be reached, whatever the opponent plays), and rules that conclude on winning states (the goal can be reached if the friend color plays first).

2.1 Generation of rules by retrograde analysis

A naive and very inefficient implementation of an automatic rule generator would generate all possible rules, and perform an alpha-beta on each rule to verify life. We use a much more efficient algorithm, albeit more complex. Previous versions and some optimizations were already described [5]. Our current retrograde analysis is an improvement. It uses an unmove function and directly finds all the rules that lead in one move to the rule at hand. Therefore, once this unmove function is written, the algorithm is quite simple: it consists of alternatively performing two passes. The first pass unmoves opponent moves on winning rules to find new won rules (verifying all the opponent moves on the unmoved rule lead to a winning rule). The second pass unmoves friendly moves on won rules to find new winning rules. The algorithm stops when no new winning rule is found in a second pass.

2.2 Conditions on external liberties

The conditions associated to the liberties outside of the pattern (external liberties) are restricted. The restrictions on the possible conditions ensures that the generated rules are always correct (provided the friend player is the komaster). The algorithm always consider that the worst things can happen to the friend player, and the best things to the opponent. This means that we assume the opponent will only need one move to reduce each of the player's external liberties, and will also only need one move to get as much liberties as he needs to for one of its strings. On the contrary, the friend player can only remove an external liberty of an opponent string if it is the last one of the string.

Friend strings in the pattern can only be associated to conditions on the minimum number of external liberties they can have. Opponent strings can only be associated to conditions on the maximum number of liberties they can have outside the pattern. Empty intersections can be associated to either a minimum number of external liberties if friend plays on the intersections, or a maximum number of external liberties if the opponent plays on the intersection.

3 Metarules to reduce the number of rules

One major problem with generated rule databases is their size. Many very useful life rules do not fit in a 4x3 pattern size, but as the number of rules grows exponentially with the size of the pattern associated to the rule, it is currently hard to have pattern sizes greater than 5x3 in the corner. In this section, we will give some methods to reduce the number of generated rules, without reducing the number of situations covered by the system. The methods are called metarule as they are rules on rules. The first one consists in removing all the rules that are special cases of another rules with a smaller pattern size. The second one is the suppression of the rules that can be easily and cheaply found at runtime. The third one is the use of generated rules on eyes to reduce the number of generated rules on life.

3.1 Metarules to suppress subsumed rules

Each time the system generates a rule, it verifies that it is not a special case of another rule. If the rule is original, it adds it to the rule database, and search the database for rules that are a special case of the new added rule. If such special cases are found, the system removes the subsumed rules.

A tricky part of this mechanism is when the system verifies if a rule with a smaller pattern subsumes a rule with a larger pattern. In this case, the system has to compare conditions on strings and intersections that are not the same, and that change according to the colors of the intersections which are present in the large pattern and absent of the small one.

For example, if a condition in the small pattern is that the friend player has at least two external liberties if he plays on an empty intersection on the border of the pattern. Then if in the larger pattern, this same intersection is now neighboring a new empty intersection of the enlarged pattern, the condition on the intersection has changed for the larger pattern: the friend player has now at least only one external liberty if he plays on the intersection, since the neighboring empty intersection is part of the larger pattern but is external to the smaller.

3.2 Suppression of rules that can be found dynamically

One optimization that divides the sizes of the databases by ten is to remove the conditions associated to winning rules, only keeping the patterns. In the Tsume Go solver, the winning state can be found by verifying the pattern, and then trying all the relevant moves to see if they lead to a won state. This optimization has some similarity with Abstract Proof Search [4] which also dynamically finds such states.

3.3 Suppression of some rules on life given rules on eyes

The system has generated rules for different eye sizes. A group is alive when it has two eyes. Therefore patterns on eyes can be used to detect life. However, the two eyes need to be independent for the string to be alive. For example when a string has two won eyes on different locations, if they share an empty intersection, their combination may not give life as an opponent move on the empty intersection can threaten both eyes, and it is possible that no friend move can save both eyes in one move.

Nevertheless, if a life rule contains two independent eyes, it is not necessary to keep it as it can be deduced from the rules on eyes, provided there is a method to detect the independence of the two won eyes. A simple metarule that ensures independence of the two won eyes most of the times is the following one: if the intersection of the two won eye patterns contains only empty and friend stones, and if all the empty intersections are protected intersections (i.e. if the opponent plays on one of them it has at most one external liberty and no internal liberty) then the string is alive. This metarule enables to remove many rules on life, however it is sometimes false. In order to preserve the correction of our algorithm, a good way to overcome the difficulty is to generate all the rules for a given pattern size that put the independence metarule above at default. Once these exception rules are generated, the rule database is safe again as all the exceptions to the potentially false metarule have been generated.

3.4 Suppression of low utility rules

Many of the generated rules have conditions of no external liberties for opponent strings or opponent moves. When a rule has many of these conditions, it has a low probability to apply, and usually, life can be detected by other means than matching the rule. Another parameter related to the utility of a rule is the depth of the rule (i.e. the minimum number of moves to make life under best defense). We evaluate each generated rule according to an utility function that gives large penalties for conditions that a rarely matched, and gives bonus for the depth of the rule. Then only rules that have an utility value below some given threshold are kept. A better but more difficult way to find the utility of the rules is to keep statistics on their use inside the problem solver, and then after having used it on a large number of problems, to remove the ones with a low number of matching.

4 Experimental results

The Table 1 gives the number of won rules on life generated for different pattern sizes in the corner of the board. The only metarule used is the metarule that removes rules that are special cases of smaller rules. Some generated rules can replace some quite deep search at a low cost of matching. The Table 2 gives the number of won rules generated for different pattern sizes on the border of the board. The Table 3 gives the number of won eye rules for thet 3x3 and 4x3 patterns in the center, and for the 3x2,

Table 1: Repartition of won life rules in the corner.

	Pattern size in the corner				
Depth	$4x2$	$3x3$	$5x2$	$4x3$	$6x2$
0	1	10	2	71	10
2	4	18	12	405	60
4	6	29	37	1237	176
6	2	8	78	2625	371
8	1	10	54	3026	615
10	0	3	27	2745	562
12	0	0	13	1202	446
14	0	0	2	272	202
16	0	0	0	39	45
18	0	0	0	5	23
20	0	0	0	1	4
Total	14	78	225	11628	2515

Table 2: Repartition of won life rules on the side.

	Pattern size on the side			
Depth	$5x2$	$3x4$	$6x2$	$4x3$
0	1	5	1	42
2	5	22	8	156
4	11	48	35	310
6	6	55	124	243
8	5	47	122	200
10	2	16	102	75
12	1	4	77	40
14	0	2	28	8
16	0	0	14	0
18	0	0	18	0
20	0	0	2	0
Total	31	197	531	1074

4x2 and 3x3 patterns on the side of the board. The Table 4 gives results for eyes in the corner.

5 Future Work

One promising improvement is the association of rules to gradual games as defined in [7] and [8]. The gradual game is a good indicator of the complexity of finding the rule using search only. Therefore, it is a good indicator of how much we ae willing to keep the rule. The rules that are the most difficult to find again should be preserved, whereas the simple rules that can be found by search easily can be dropped without too much concern.

Improvements due to the use of generated rules can be mixed with improvements due to other new search algorithms such as Abstract Proof Search [4], algorithms that learn to order moves [15], and search heuristics such as the killer move heuristic used in Gotools [20]. It is interesting to test some combinations of these improvements on a life and death test suite [20],[6].

Potential reductions in the number of rules can com from the use of the static life and death heuristics of top

Table 3: Repartition of won eye rules center and side.

	Center		Side		
Depth	$3x3$	$4x3$	$3x2$	$4x2$	$3x3$
0	9	0	1	0	9
2	34	25	3	1	37
4	76	333	4	12	120
6	52	2561	0	59	257
8	60	2831	1	38	154
10	26	4430	0	36	89
12	16	1492	0	6	36
14	6	1486	0	0	23
16	0	283	0	0	0
18	0	0	0	0	0
20	0	0	0	0	0
Total	279	13441	9	152	725

Table 4: Repartition of won eye rules in the corner.

	Pattern size in the corner			
Depth	$2x2$	$3x2$	$4x2$	$3x3$
0	1	1	0	7
2	2	4	11	31
4	1	13	26	111
6	0	34	93	342
8	0	7	133	387
10	0	0	93	247
12	0	0	25	74
14	0	0	0	1
16	0	0	0	0
18	0	0	0	0
20	0	0	0	0
Total	4	59	381	1200

programs [11]. The system could automatically find all the exceptions of the heuristic rules. Also, many rules could be removed as some special cases of the heuristics.

The shapes databases of D. Dyer [12] are a special case of our retrograde analysis algorithm. However, he uses isomers of shapes to reduce the number of shapes and we do not. We could use this heuristic to reduce the number of rules, detecting equivalent ones.

From a more general point of view, our methods can be used to help assessing the independence of subgames. For example, connections are often transitive, but sometimes a string A can be connected to a string B, the string B can be connected to a string C, but A is not connected to C. Non-transitivity occurs when the two connections are dependent [7]. Our rule generating system with exceptions could find all the non-transitivity cases automatically.

6 Conclusion

Many rules about life and eyes have been automatically generated, and are used to speed-up Tsume Go problem solving. The main limitation of automatically generating rules is the size of the databases. We have given metarules to reduce the number of generated rules, therefore enabling rules with larger pattern size to be generated. We also gave some experimental results concerning the depth and the number of generated rules.

References

[1] D.B. Benson, 'Life in the game of go', *Information Sciences*, **10**, 17–29, (1976).

[2] B. Bouzy and T. Cazenave, 'Computer go: An ai-oriented survey', *Artificial Intelligence*, **132**(1), 39–103, (October 2001).

[3] T. Cazenave, 'Système d'apprentissage par auto-observation. application au jeu de go', Phd thesis, Université Paris 6, (December 1996).

[4] T. Cazenave, 'Abstract proof search', in *Proceedings of Second International Conference on Computers and Games*, pp. 81–96, Hamamatsu, (2000).

[5] T. Cazenave, 'Generation of patterns with external conditions for the game of go', in *Advance in Computer Games 9*, eds., H.J. van den Herik and B. Monien, 275–293, Universiteit Maastricht, Maastricht, (2001). ISBN 90 6216 566 4.

[6] T. Cazenave, 'A problem library for computer go', in *IJCAI-01 Workshop on Empirical Methods in Artificial Intelligence*, eds., Holger H. Hoos and Thomas Sttzle, Seattle, USA, (2001).

[7] T. Cazenave, 'Theorem proving in the game of go', in *Proceedings of the first International Conference on Baduk*, pp. 275–292, Yong-In, Korea, (2001).

[8] T. Cazenave, 'Gradual abstract proof search', in *Proceedings of RFIA*, Angers, France, (2002).

[9] K. Chen, 'Group identification in computer go', in *Heuristic Programming in Artificial Intelligence: The First Computer Olympiad*, eds., D. Levy and D. Beal, 195–210, Ellis Horwood, Chichester, (1989).

[10] K. Chen, 'The move decision process of go intellect', *Computer Go*, **14**, 9–17, (1990).

[11] K. Chen and Z. Chen, 'Static analysis of life and death in the game of go', *Information Sciences*, **121**, 113–134, (1999).

[12] D. Dyer, 'An eye shape library for computer go', Technical report, (1987).

[13] R. Gasser, 'Endgame database compression for humans and machines', in *Heuristic Programming in Artificial Intelligence 3: the third computer olympiad*, eds., H.J. van den Herik and L.V. Allis, 180–191, Ellis Horwood, Chichester, England, (1991).

[14] J. Kraszek, 'Heuristics in the life and death algorithm of a go-playing program', *Computer Go*, **9**, 13–24, (1988).

[15] J. Ramon, T. Francis, and H. Blockeel, 'Learning a tsume-go heuristic with tilde', in *Proceedings of CG2000, the second international conference on computer and games*, eds., Ian Frank and Tony Marsland, Hamamatsu, Japan, (2000). To appear in LNCS.

[16] T. Wolf, 'Investigating tsumego problems with risiko', in *Heuristic Programming in Artificial Intelligence 2*, eds., D.N.L. Levy and D.F. Beal, Ellis Horwood, (1991).

[17] T. Wolf, 'The program gotools and its computer-generated tsume go database', in *Game Programming Workshop in Japan '94*, ed., H. Matsubara, pp. 84–96, Tokyo, Japan, (1994). Computer Shogi Association.

[18] T. Wolf, 'About problems in generalizing a tsumego program to open positions', in *Game Programming Workshop in Japan '96*, ed., H. Matsubara, pp. 20–26, Tokyo, Japan, (1996). Computer Shogi Association.

[19] T. Wolf, 'The diamond', *British Go Journal*, **108**, 34–36, (1997).

[20] T. Wolf, 'Forward pruning and other heuristic search techniques in tsume go', *Information Sciences*, **122**, 59–76, (2000).