Metarules to Improve Tactical Go Knowledge

Tristan Cazenave

Labo IA, Université Paris 8
2 rue de la Liberté, 93526, St-Denis, France
fax: 33 1 49 40 64 10
e-mail: cazenave@ai.univ-paris8.fr

Abstract

Three main problems arise with automatically generated rules databases. They can be too large to fit in memory, they can take a lot of time to generate, and it can take time to match many generated 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 the overall search time of a tsumego problem solver.

Key words: Game of Go, Retrograde Analysis, Metaknowledge, Pattern Databases, Search

1 Introduction

I have written an automatic rule generator based on retrograde analysis of patterns with external conditions [3,4,2]. It generates rules about eyes and life in the game of Go. These generated rules can be used in a tsumego problem solver to find non trivial eye making moves and to reduce the search depth.

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 [16–20].

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 pattern 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). By convention, the player that tries to make an eye or to live is always Black. In this section, we start in the first subsection with giving properties of the external conditions associated to rules. Thes properties ensure that rules are always correct and this enables retrograde analysis to work. The second subsection deals with the retrograde analysis algorithm itself.

2.1 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 considers that the worst things can happen to Black, and the best things to White. This means that we assume that White will only need one move to reduce each of Black's external liberty, and will also only need one move to get as much liberties as he needs to for one of its strings. On the contrary, Black can only remove an external liberty of an opponent string if it is the last liberty of the string.

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

2.2 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 search on each rule to verify

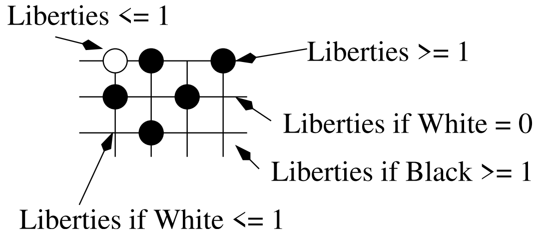

Fig. 1. A generated 4x3 rule on the side.

life. We use a much more efficient algorithm, albeit more complex. Previous versions and some optimizations were already described [4]. 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 and much faster: it consists of alternatively performing two passes. The first pass unmoves White moves on winning rules to find new won rules (verifying all the White moves on the unmoved rule lead to a winning rule). The second pass unmoves Black moves on won rules to find new winning rules. The algorithm stops when no new winning rule is found in a second pass.

```
RuleGeneration(PatternSize) {
 FindRulesWithTwoPhysicalEyes();
 while(NewRulesFound) {
  For all new won rules {
   Undo Black moves to find
          new winning rules;
   Memorize new winning rules;
  }
  For all new winning rules {
   Undo White moves to find
              new won rules;
   Memorize new won rules;
  }
 }
}
```

3 Metarules to reduce the number of rules

One major problem with generated rules databases is their size. Many 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 database, it is currently hard to compute databases for 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 metarule is described in the first subsection, it consists in removing all the rules that are special cases of another rule with an equal or 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. The fourth set of metarules is designed to detect and eliminate redundant rules. In the next subsection, we deal with metarules in charge of removing useless conditions. Then we describe some metarules used to detect illegal and irrelevant rules. Other metarules can be designed to eliminate low utility rules, this will also be the theme of a subsection. In the last subsection, we address the possible restrictions on the rules to generate.

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 searches 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 Black has at least two external liberties if he plays on an empty intersection on the border of the pattern and 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: Black only has now at least 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

An optimization that reduces a lot the sizes of the databases is to remove the winning rules (i.e. rules where making a move makes life), only keeping the won rules (i.e. rules where the strings live even if the other player plays frist). In the Tsume Go solver, the winning state can be found trying all the relevant moves to see if they lead to a state where a won rule is matched. Forced moves can be found by playing White moves and only keeping the the moves that do not lead to the matching of a won rule after a Black move following the first White move. This optimization has

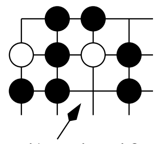

Fig. 2. Two independent eyes in a rule.

some similarity with Abstract Proof Search [8] which also dynamically finds such states.

The generation algorithm described in the section on rule generation can be modified to take this optimization into account:

```
RuleGeneration(PatternSize) {
 FindRulesWithTwoPhysicalEyes();
 while(NewRulesFound) {
  For all new won rules {
   Undo Black moves to find
          new winning rules;
   For these winning rules {
    Undo White moves to find
               new won rules;
    Memorize new won rules;
   }
  }
 }
}
```

In this retrograde analysis algorithm, the winning rules need not being stored even during the generation phase. This optimization saves a lot of memory. For example, for the 4x3 rules in the corner, there are 11404 won rules and 86938 winning rules. The number of winning rules is usually between 7 and 10 times the number of won rules. Moreover, the killing and living moves need no to be stored any more. This optimization divides by ten the size of the generated databases of rules.

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.

Fig. 3. Redundancy of a rule.

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 White 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.

Out of the 11,404 4x3 rules on life in the corner, 2053 contain two different patterns on eyes. This gives an idea of the potential savings of using this metarule. However, it is possible that the larger the pattern the larger the savings. For example, there are much more possible ways of combining two eye patterns in a 5x3 pattern than in a 4x3 pattern.

3.4 Suppression of redundant rules

Some conditions can be deduced from other conditions, and some conditions cover more cases than others. The system can take this knowledge into account to remove rules. For example, the condition that states that the number of external liberties is greater than one if Black plays on the pointed intersection in the left rule of the figure 3 can be deduced from the condition that the string has more than one external liberty in the right rule of the figure 3.

The first rule will always apply when the second rule applies. However, in some cases, the first rule will apply and not the second. The first rule is more general than the second, and the second rule should be discarded.

The metarule that discards redundant rules starts with adding redundant conditions to the candidate rule to be removed. For example, when a condition on a minimum number of liberties of a given string is present, it adds to the rule the corresponding conditions on the minimum number of liberties when Black plays on the liberties

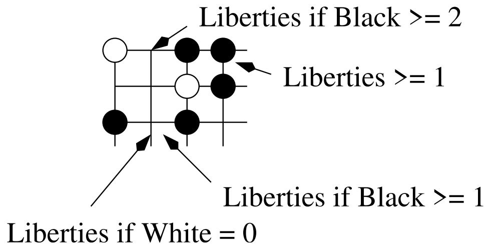

Fig. 4. A useless condition in a rule.

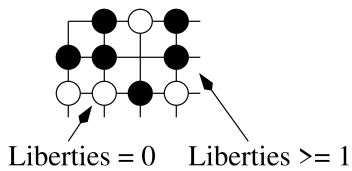

Fig. 5. Illegal rule.

of the string.

3.5 Removing useless conditions

Removing useless conditions is also a way to reduce the number of generated rules. Rules with less conditions will subsume more rules as they are more general. Removing useless conditions produces more general rules that enable to remove more redundant rules.

In the figure 4, the condition that the left black string has more than one external liberty is useless since there is also a condition that states that a Black move on an empty intersection, which is also a liberty of the string and only a liberty of this string, has at least two external liberties. As the empty intersection does not touch the frontier of the pattern, the only way for a Black move there to have two external liberties, is that the black string has at least two external liberties. Therefore the condition that the string has at least one external liberty can be removed.

3.6 Suppression of illegal and irrelevant rules

In the figure 5, the rule is illegal because the white string has no liberties in the pattern and is associated to the condition 'no liberties outside the pattern'. The white string has no liberty at all, it is never possible to match the rule and therefore the rule is removed. Rules that contain strings that do not touch the border of the pattern and that have no internal liberties are also removed.

In the figure 6, the rule is irrelevant because when playing on the empty intersection the resulting string has no liberties. However the white string neighbouring the

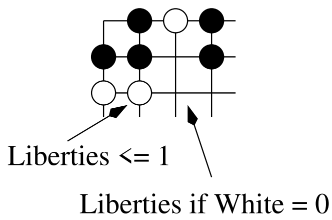

Fig. 6. Irrelevant rule.

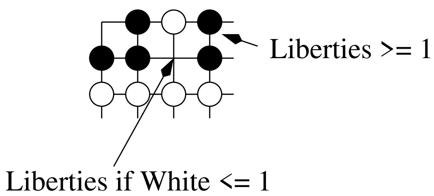

Fig. 7. Low utility rule.

intersection has possibly one liberty. This is not compatible with the condition that a move on one of its liberties has no external liberties. The only possible condition on the neighbouring string is that it has no external liberties. Irrelevant rules are not tested.

Rules that contain a Black string with no internal liberties, and no conditions on its minimum number of liberties are also considered irrelevant since the condition that the Black string has at least one external liberty has always to be verified in order for the rule to be valid. If we do not discard this rule, it may cause problems when used by larger rules and the string is associated to a minimum of one external liberty, when it should be associated to two so as not to be captured.

3.7 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 (such as capturing a white string). 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.

The table 1 gives the repartition of the conditions for maximum numbers of liberties

Table 1
Repartition of max conditions for won 4x3 rules.

	$C0$	$C1$	$I0$	$I1$	$S0$	$S1$
C0	1918	4421	2643	1121	375	282
C1	4421	5065	4389	2255	1286	776
I0	2643	4389	2223	1646	1102	490
I1	1121	2255	1646	417	371	319
S0	375	1286	1102	371	96	66
S1	282	776	490	319	66	60

for White in the 4x3 set of rules for won life in the corner. The different acronyms represent the number of max conditions (C0=no max conditions, C1=at least one max condition), and more specifically we also use I0: at least one condition that the number of liberties if White plays on an empty intersection is zero, I1: at least one condition that the number of liberties if White plays on an empty intersection is one, S0: at least one condition that a White string has no external liberties, S1: at least one condition that a White string has one or no external liberties.

A location in the table gives the number of rules that verify the property corresponding to an acronym, when the other property is also verified and that the condition associated to the verification of the property is removed. For example, the top left location corresponds to the couple (C0,C0) which is the number of rules that have no max conditions: 1918 rules. Another example is the (I0,C1) couple which means that a condition with no external liberties if White plays is in the counted rule, and that at least another max condition is also present. There are 4389 such rules.

Rules that have a low probability of being matched are for example rules that have two or more conditions on no external liberties. An approximation of the number of such rules is the sum of (S0,S0), (S0,I0) and (I0,I0), which is 96+1102+2223=4421 rules out of 11404 4x3 rules on won life in the corner.

3.8 Restrictions on the rules to generate

It is possible to have constraints on the conditions associated to a rule. We will consider two special cases: the generation of pessimistic rules and the generation of optimistic rules. In order to differentiate them from the rules generated by the usual algorithm, we will call the usual rules realistic rules.

Pessimistic rules have no associated conditions. Therefore a pessimistic rule takes less memory than a realistic rule. The conditions associated to the rule are pes-

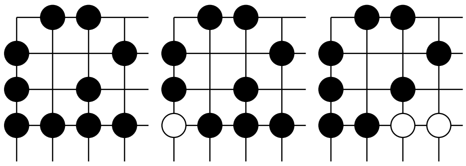

Fig. 8. Many pessimistic rules for one realistic one.

simistic for Black. All the black strings are supposed to have no external liberty, all the white strings are supposed to have an infinite number of liberties, every empty intersection on the border of the pattern is supposed to have no external liberties if Black plays on it, and an infinite number of external liberties if White plays on it. These limitations imply that the generated rules are correct without having to check any condition on the external liberties because all the restrictions are imposed on the player that tries to live or to make an eye. If he is alive given these extreme restrictions, he is alive independently of the environment exterior to the pattern. The pessimistic rules correspond to the realistic rules with no associated conditions.

Pessimistic rules are interesting because there are less pessimistic rules than realistic rules for a given pattern size, and because each rule takes much less memory. However there is an important drawback of pessimistic rules: if all the pattern sizes are generated using pessimistic rules, there are much more pessimistic rules than realistic rules and they cover less cases on a real board because of the restrictions on the external conditions. This is a priori contradictory to the observation that pessimistic rules correspond to the realistic rules with no associated conditions. In fact, this assertion is true only when the rules used for the smaller patterns are realistic rules. If the rules used for the smaller pattern sizes are also pessimistic rules, many pessimistic rules can be generated to replace only one realistic rule with a smaller pattern size. An example of this phenomena is given in the figure 8. It gives three pessimistic rules generated for the 4x4 pattern in the corner of the board. These rules can be compared with the first rule of the figure 9. They are all special cases of the first rule of the figure 9. Moreover the first rule of the figure 9 has only one external condition and is only one pattern size down of the pessimistic rules. There are already 24 corresponding pessimistic rules. This is even worse when there are more external conditions involved and a greater difference in pattern size.

On the contrary of pessimistic rules, optimistic rules are associated to conditions optimistic for Black. This means that all the black strings have an infinite number of external liberties if they touch the border of the pattern, that white strings have no external liberties, that Black moves touching the border have an infinite number of liberties, and White moves no external liberties. The interest of optimistic rules is that they take less memory than realistic rules, that one optimistic rule can cover multiple realistic rules as shown in the figure 9 where the same pattern is associated to four different realistic rules. The same pattern corresponds to only one optimistic rule.

Any pattern of a realistic rule is also a pattern of an optimistic rule. Therefore, it is

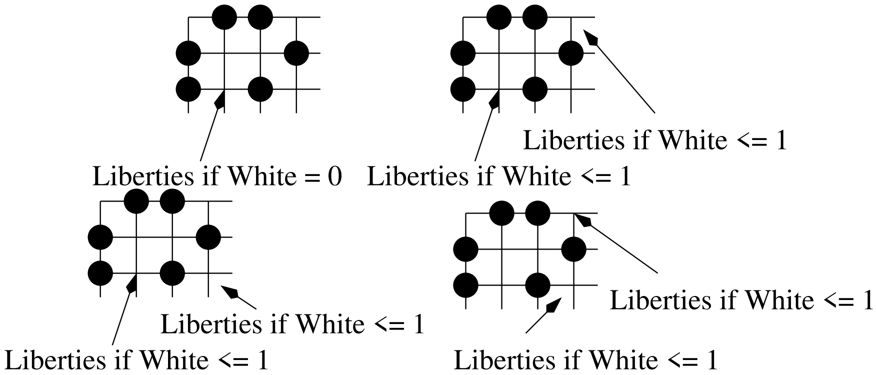

Fig. 9. Multiple rules with the same pattern.

useful to generate the optimistic rules in a first pass as they are faster to compute and need much less memory than the realistic rules. They can later be used to speed-up the computation of the realistic rules by focusing the search on the possible patterns, and preventing to search useless patterns that are not in any optimistic rule.

When the number of realistic rules is too high to fit in memory, computing the optimistic rules can be useful for the tsumego problem solver. They can of course be useful to stop search when they are verified (optimistic rules can be generated with a high value like 2 or 3 for the number of external liberties instead of infinity, making them more realistic and not changing the set of generated optimistic rules). But they can also be useful when the pattern is verified but not the optimistic conditions. In this case it is an indication for the solver that the matching of a won rule might not be too far, and that it can be worth searching the position.

4 Experimental results

The table 2 gives the number of won rules on life generated for different pattern sizes in the corner of the board. The metarules used are the metarule that removes rules that are special cases of smaller rules, the metarules on redundancy, the metarules on useless conditions and the metarules that remove illegal and irrelevant rules. Some generated rules can replace some quite deep search at a low cost of matching. The table 3 gives the number of won rules generated for different pattern sizes on the border of the board. The table 4 gives the number of won eye rules for the 3x3 and 4x3 patterns in the center, and for the 3x2, 4x2 and 3x3 patterns on the side of the board. The table 5 gives results for eyes in the corner.

Table 2
Repartition of won life rules in the corner.

	Pattern size in the corner				
Depth	$4x2$	$3x3$	$5x2$	$4x3$	$6x2$
0	1	10	2	71	4
2	4	18	12	411	36
4	6	29	37	1262	120
6	2	8	78	2737	359
8	1	10	54	3041	682
10	0	3	27	2601	680
12	0	0	13	1050	520
14	0	0	2	203	272
16	0	0	0	22	98
18	0	0	0	6	35
20	0	0	0	0	12
22	0	0	0	0	3
Total	14	78	225	11404	2835

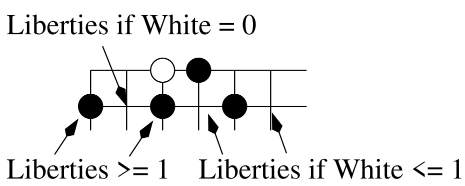

Fig. 10. A depth 16 6x2 won rule.

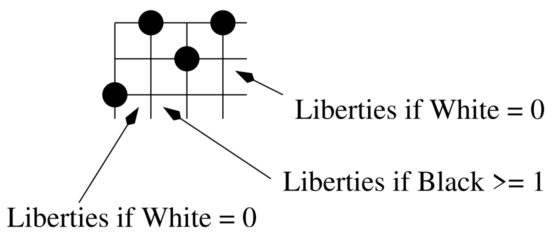

Fig. 11. Example of a generated depth 12 won rule.

5 Future Work

One promising improvement is the association of rules to gradual games as defined in [6] and [7]. The gradual game is a good indicator of the complexity of finding

Table 3
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

Table 4
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
Total	279	13441	9	152	725

Table 5
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
Total	4	59	381	1200

the rule using search only. Therefore, it is a good indicator of how much we are 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 [8], 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 [20,5].

Potential reductions in the number of rules can come from the use of the static life and death heuristics of top 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 [6]. 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. Generation of patterns with external conditions for the game of go. In H.J. van den Herik and B. Monien, editors, *Advance in Computer Games 9*, pages 275–293. Universiteit Maastricht, Maastricht, 2001. ISBN 90 6216 566 4.

[5] T. Cazenave. A problem library for computer go. In Holger H. Hoos and Thomas Stuetzle, editors, *IJCAI-01 Workshop on Empirical Methods in Artificial Intelligence*, Seattle, USA, 2001.

[6] T. Cazenave. Theorem proving in the game of go. In *Proceedings of the first International Conference on Baduk*, pages 275–292, Yong-In, Korea, 2001.

[7] T. Cazenave. La recherche abstraite graduelle de preuves. In *Proceedings of RFIA-02*, pages 615–623, Angers, France, 2002.

[8] Tristan Cazenave. Abstract proof search. In T. Anthony Marsland and Ian Frank, editors, *Computers and Games*, volume 2063 of *Lecture Notes in Computer Science*, pages 39–54. Springer, 2001.

[9] K. Chen. Group identification in computer go. In D. Levy and D. Beal, editors, *Heuristic Programming in Artificial Intelligence: The First Computer Olympiad*, pages 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 H.J. van den Herik and L.V. Allis, editors, *Heuristic Programming in Artificial Intelligence 3: the third computer olympiad*, pages 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 T. Anthony Marsland and Ian Frank, editors, *Computers and Games*, volume 2063 of *Lecture Notes in Computer Science*, pages 151–169. Springer, 2001.

[16] T. Wolf. Investigating tsumego problems with risiko. In D.N.L. Levy and D.F. Beal, editors, *Heuristic Programming in Artificial Intelligence 2*. Ellis Horwood, 1991.

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

[18] T. Wolf. About problems in generalizing a tsumego program to open positions. In H. Matsubara, editor, *Game Programming Workshop in Japan '96*, pages 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.