Admissible Moves in Two-player Games

Tristan Cazenave

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

Abstract. Some games have abstract properties that can be used to design admissible heuristics on moves. These admissible heuristics are useful to speed up search. They work well with depth-bounded search algorithms such as Gradual Abstract Proof Search that select moves based on the distance to the goal. We analyze the benefits of these admissible heuristics on moves for rules generation and search. We give experimental results that support our claim for the game of AtariGo.

1 Introduction

In some games, abstract properties can be used to design admissible heuristics on the minimal number of moves required to win the game. It is possible to relax the rules of a game and play admissible moves in the game with relaxed rules. The number of moves to win is always lower in the relaxed game than in the real game. The interest of relaxation is that the minimal number of moves can be computed faster than in the original game. The abstract knowledge on the moves can be used to select a subset of relevant threat moves when using a threat based search algorithm. It can also be used to stop a depth-bounded search when the number of admissible moves required to win is greater than twice the depth of the search.

In the paper, the attacker is the player who tries to win a game, and the defender is the player who tries to prevent the attacker from winning. For example, if the game is to make a group live in the game of Go, the attacker is the player who plays moves to live, and the defender is the player who tries to kill the group. When the game is to connect two strings in the game of Go or in the game of Hex, the attacker tries to connect the strings and the defender tries to prevent the attacker from doing so.

We present examples of admissible heuristics on moves, as well as experimental results and some methods relevant to the generation and the use of admissible moves in two-players games. Where experimental results are available, we mention them. Whenever it is possible, we also outline ideas that are currently under investigation in the hope to stimulate research on admissible heuristics in games. In the second section, the game of AtariGo is described. In the third section, we explain how admissible heuristics can be designed in two-player games. In the fourth section, we relate the admissible heuristics on the number of moves that have to be played in order to win a game to threat based search algorithms. We also describe Gradual Abstract Proof Search and we give experimental results quantifying the usefulness of abstract knowledge for the game of AtariGo solved by the Abstract Gradual Proof Search algorithm. In the fifth section, we outline the interest of admissible heuristics for the automatic generation of rules. The sixth section details the automatic generation of admissible heuristics on moves by transforming a logic program of the rules of the game. The last section outlines future work and concludes.

2 AtariGo

AtariGo is used to teach beginners to play the game of Go. The board used to play AtariGo is a square grid. It can be played on any board size. It is usually played on a small board so that games do not take too much time. Teachers also often choose to start with a crosscut in the centre of the board in order to have an unstable position. We have tested the usefulness of different abstraction levels on the game starting with a crosscut in the centre of a 6x6 board.

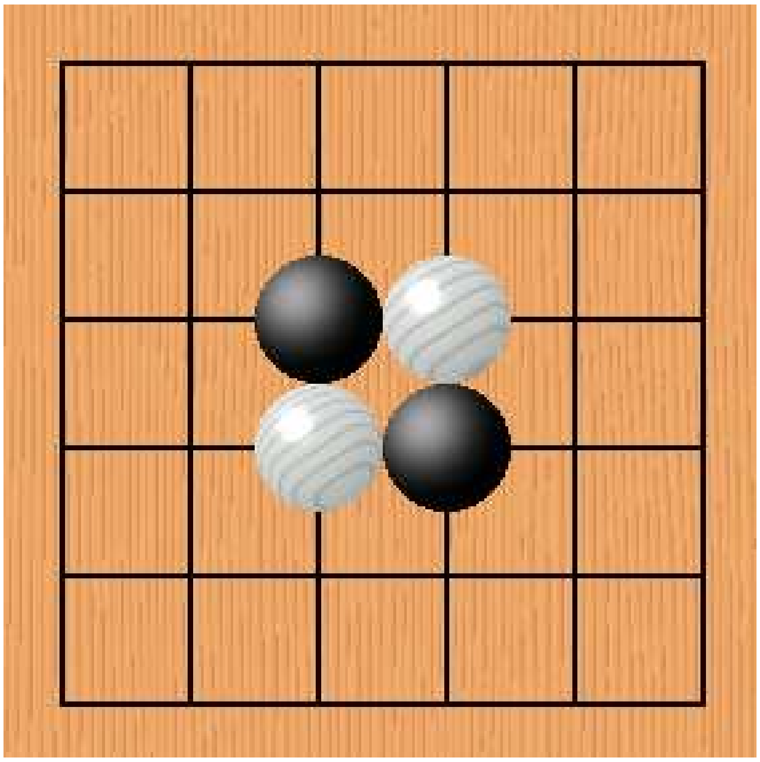

Fig. 1. The initial board for 6x6 AtariGo.

The rules are similar to Go: Black begins, Black and White alternate playing stones on the intersections of the board, strings of stones are stones of the same color that are linked by a line on the board. The number of empty intersections adjacent to the string is the number of liberties of the string. A string is captured if it has no liberty. For example in the Figure 1, all the strings have two liberties. A string that has only one liberty left can be captured by the other color in one move, it is in Atari, this is where the name of the game comes from. The goal of the game is to be the first player to capture a string of the opponent.

3 Admissible number of moves

In this section we start with explaining how it is possible to design admissible heuristics on the number of moves to win by relaxing the rules of the game. Then we give some example of admissible moves, and of the related admissible heuristics. The last subsection outlines possible optimizations of the computation of the designed heuristics.

3.1 Relaxation of the rules of the game

The admissible rules of a game are modified rules of the game. Usually, the admissible rules are more simple than the original ones. The rules of a game are admissible if the number of moves to win under these rules is always lower than the number of moves to win under the real rules, in any legal position.

An example of relaxation in the game of Go is to remove the forbidden moves. For example, relaxed rules where it is always legal to play on an empty intersection can be designed. Relaxations of the rules of a game can be automatically discovered by removing some conditions of the rules of the game represented in a logic language. However, with this method there are a large number of possible relaxations. It is not always easy to automatically find the useful relaxations out of all the possible ones. It can be easier in some games than in other. For example in Go, the number of liberties is easily deduced as an admissible heuristic on the number of moves to capture a string. For the 15-puzzle, the Manhattan distance can also be found with a relaxation of the rules of the game, just by removing the condition that the target tile has to be empty.

3.2 Admissible moves

An admissible move is a move in a game with relaxed rules. The number of admissible moves required to win is always lower than the number of moves required to win in the real game.

For example in the game of Go, putting a stone on the liberty of a string is an admissible move. In the real game of Go, it is not always possible to play on the liberty of a string. For example it is not possible to remove a liberty if it is an eye, and if it is not the last liberty of the string. But if we relax the rules of Go, stating that it is always possible to put a stone on an empty intersection, we have admissible moves.

3.3 Admissible number of moves

The admissible number of moves is a lower bound on the number of moves needed to win the game. It is also the number of admissible moves needed to win the game. For example in the game of Go, the number of liberties of a string is an admissible heuristic of the number of moves in a row needed to capture the string.

In Philosopher's Football (Phutball) [1] threat search algorithms work very well. It is also possible to define simple admissible heuristics on the number of moves. A move in Phutball consists either in putting a white stone on an empty intersection, or jumping the ball (a black stone) over white stones. The game is over when the ball is on or behind the opponent goal line. A simple admissible heuristic on the number of moves is the length of the shortest line of empty intersections touching the goal line. The minimal number of moves in a row needed to win is half this length plus one, as there has to be at least one white stone every two intersections to move the ball to the goal line, and that moving the ball is the last move.

For the game of connection in Go, an admissible heuristic on the number of moves required to connect two strings is the length of the shortest path between the two strings. The length of the path being the number of empty intersections on the path plus the number of liberties that are not already on the path of the opponent strings that are on the path.

A refinement of these simple heuristics is to perform a tree search to find the minimum number of moves the attacker has to play when the defender is allowed to play one move, to play two moves separated by one or more moves by the attacker, etc... It is equivalent to a search algorithm, but it is faster than an Alpha-Beta as the Alpha-Beta is the limit of this sequence of trees: In Alpha-Beta the defender is allowed to play as many moves as the attacker. As the complexity of the search is exponential with the depth, these trees are computed much faster than the usual Alpha-Beta and they can stop search earlier and memorize useful information for move ordering as in the iterative deepening algorithm. We will come back to this refinement in the section on search.

3.4 Optimisation of the computation of the admissible number of moves

In Hex or in the connection game of Go, the admissible number of moves is the length of the shortest path between the two strings to connect. An optimization to the computation of the shortest path is to start searching the shortest path from the two strings to connect and to iteratively expand the perimeter around each string. It is equivalent to a bidirectionnal search, and it is faster than to expand from the first string until the second string is touched. It is also much faster that a brute force algorithm that would try all the possible moves.

An important speed-up come from the incremental updating of the heuristic. For the capture game of Go, it is not so simple to maintain the liberties incrementally from move to move. The algorithm used to incrementally update the liberties is an union find algorithm. In our experiments, all the liberties of all strings are maintained incrementally each time a move is made and each time it is taken back. The admissible heuristic which is the minimum number of liberties over all the strings for each player is also maintained incrementally. The incrementality gives a substantial speed-up.

In our experiments, it is faster to maintain the liberties incrementally for the capture game than to recompute them when needed. We did not test the efficacy of incrementality for the connection game. We are not aware of any theory that could tell us in which games incrementality speeds computations up. Such a theory would be very useful.

4 Optimization of Gradual Abstract Proof Search with abstract knowledge on the admissible number of moves

In games that have a large number of possible moves, and when playing few moves in a row often means a win, search algorithms based on threats can greatly outperform brute force Alpha-Beta. Go-Moku was solved by V. Allis using a search algorithm based on threats [2]. More recently, Abstract Proof Search [3] and Lambda Search [4] were designed and have outperformed basic Alpha-Beta search in the capture game of Go. The order of a position is the number of moves in a row that have to be played to win the game. A threat move leads to a position of order one. Abstract Proof Search uses abstract information to select the possible moves that can win a game at a given order.

4.1 Gradual Abstract Proof Search

Gradual Abstract Proof Search [5] is a refinement of Abstract Proof Search that solves the game of 6x6 AtariGo starting with a crosscut in the center. In AtariGo the first player to capture a string of stones wins. Abstract Proof Search selects MIN node moves using small depth bounded search. Lambda Search selects MIN node moves using order bounded search. Gradual Abstract Proof Search selects MIN node moves using both depth and order bounded search.

The gradual games that select moves at MIN nodes start with the letters 'ip'. Following the ip letters, a number gives the maximum number of attacker moves that can be played before winning the game. Then a sequence of numbers give the maximum order of each of these moves. For example the ip1 game finds moves that prevent the attacker from winning in one move. The ip2 game finds moves that prevent the attacker from winning in a depth two search tree. The ip2 game should be noted ip211, but as all the attacker moves in an ip2 game are always of order 1 it is simply noted ip2. The ip311 game finds moves that prevent a winning sequence of two consecutive direct threats followed by a winning move. All the attacker moves lead to positions of order one or less (winning position if the attacker moves and won positions). The ip4221 game finds moves that prevent the attacker from winning after two order 2 threats.

Figure 2 gives some examples of trees representing different gradual games. As in combinatorial game theory the two players are named Left and Right. The Left player is the attacker and the Right player is the defender. In these trees, a branch that goes on the left represents a Left move, and a branch that goes on the right represents some Right moves. Left branches are associated with winning moves for Left, and right branches are associated with the complete set of Right moves that can possibly prevent the win of the corresponding left branch (the left branch directly at the left of the right one, i.e. its sibling). All the leaves of the trees are positions won for Left. In order for the game to be verified, all Left moves have to be winning moves, and all Right moves have to be refuted by Left.

The tree labeled ip1 in the Figure 2 is the most simple one. The ip1 game is verified when the attacker can win in one move. A left branch that ends with a leaf node is always a winning move for the attacker. The ip2 tree represents a position where the attacker can win in at most two moves. The first left branch represents an attacker move that threatens to win. Below this first branch, there are two branches. The left one is the winning attacker move that executes the threat, and the right one represents all the defender moves that can potentially invalidate the threat. But all the defender moves are followed by a winning attacker move as symbolized by the last left branch. The other trees in the figure show some other games following the same graphical convention.

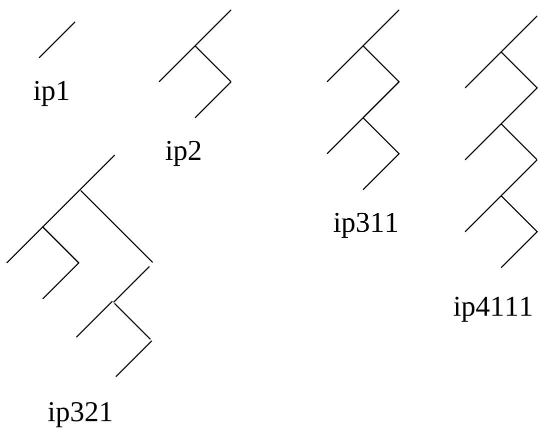

Fig. 2. Some gradual games.

4.2 Abstract knowledge for gradual games

Verifying complex gradual games such as ip4221 can take a relatively long time. Abstract knowledge can be used to detect early that a gradual game cannot be verified, for example when the number of admissible moves to win is greater than the order of the game. For example in AtariGo, it means that no order 2 game can be verified when the minimum number of liberties of all the defender strings is 3.

The abstract knowledge of order one is the knowledge to optimize the generation of possible moves when looking for a winning move. It consists in testing if there is an opponent string with only one liberty left. The attacker move generator returns the liberty if it is the case, and returns an empty set of moves when there is no such string. The defender move generator of order one first tries to capture an opponent string with only one liberty, and if there is none, it looks for friend strings with only one liberty. If there is one such string, it plays its liberty. If there is none, it returns an empty set of moves.

This order one abstract knowledge is very useful. An Abstract Gradual Proof Search with no abstract knowledge is impractical. We have stopped it after more than one hour of search. Whereas a simple order one abstract knowledge optimization makes it practical as can be seen in Table 1: 6x6 AtariGo with a cross cut in the centre is solved in less than 10 minutes.

The order two abstract knowledge for the attacker move generator consists in returning the liberty of a defender string if it has only one liberty, otherwise in returning an empty set if the minimum number of liberties of opponent strings is greater than two, else to return the liberty of a friend string with only one liberty, else to return the liberties of the opponent strings with two liberties, and if this last condition is not verified to return an empty set.

The order two abstract knowledge for the defender move generator is almost the symmetric of the attacker move generator, except when there is a defender string to save with two liberties. In this case the possible moves are the liberties of the defender string, the empty neighbors of the liberties of the defender string, the liberties and the empty neighbors of the liberties of the strings that have two liberties and which are also adjacent to the defender string, the liberties of the strings that have three liberties and which are also adjacent to the defender string and the liberties of the attacker strings that have two liberties.

The order three abstract knowledge for the attacker move generator is programmed in a similar way as the order two abstract knowledge, except that all the empty intersections that can be connected in two moves to the defender string are sent back when the defender string has only two liberties. Only the liberties are sent back when the defender string has three liberties.

There is no order three abstract knowledge for the defender move generator, it return all possible moves.

4.3 Experimental results quantifying the usefulness of abstraction

In order to estimate the usefulness of this abstract knowledge, we solved 6x6 AtariGo with Abstract Gradual Proof Search using different abstract knowledge orders. The results are given in the Table 1. The game is solved in 1 minute with the order three abstract knowledge, and in 10 minutes with the order one

Table 1. Search time with different abstractions orders.

	Time (s)		
Depth	*Order*1	*Order*2	*Order*3
1	0.06	0.02	0.00
2	17.86	4.23	2.41
3	18.06	4.04	2.98
4	54.86	11.56	5.56
5	52.95	9.59	5.43
6	114.71	21.66	10.19
7	127.42	23.25	12.85
8	174.98	38.28	19.09
9	21.07	8.15	3.89
10	3.16	0.68	0.25
Total	585.13	121.46	62.65

abstract knowledge. We did not report the results for the solution with no abstract knowledge as it took too much time. The maximum gradual game needed to solve 6x6 AtariGo is ip4221. At each MIN node, the ip1, ip2, ip311, ip4111, ip51111, ip4121, ip4211, ip4221 games are checked. If at least one game is verified, the intersection of all the sets of moves sent back by the verified games is performed and the moves in the intersection set are tried for the defender.

Table 2. Average time used to verify gradual games for different abstract orders.

	Average time (μs)		
Game	*Order*1	*Order*2	*Order*3
ip1	2	5	5
ip2	328	66	136
ip311	1165	137	137
ip4111	1751	253	220
ip51111	1763	187	148
ip321	36877	8476	4614
ip4121	63737	7843	3809
ip4211	40451	9477	4878
ip4221	196836	40368	20661

We have also output the average time used for the verification of the different gradual games. The results are given in Table 2. As can be expected the abstract knowledge of order three is quite useful for the more complex gradual games. For the ip4221 game, using abstractions of order 3 is 10 times faster than using abstractions of order 1, and twice as fast as abstractions of order 2.

The overall Alpha-Beta that calls the gradual games at MIN nodes, uses transposition tables, iterative deepening, two killer moves, the history heuristic and the difference between the minimum number of liberties of the attacker and the minimum number of liberties of the defender as an evaluation function.

The verification of the games definitions in Abstract Proof Search and Gradual Abstract Proof Search can be related to the progressive admissibility of the depth of the win. We have used so far as an admissible heuristic the number of moves in a row of the attacker color needed to win the game. If instead, we refine the heuristic by taking into account only one defender move. We still have an admissible heuristic on the depth of the win, but it is usually higher as the defender is allowed to play one move before all the attacker moves are played. If the admissible heuristic with one move for the defender gives a number greater than the maximal depth allowed, then we can cut. The number of allowed defender moves can be progressively increased until it is equal to the number of attacker moves or the depth of the search is too high. This progressive increase of the admissible heuristic costs one defender move at each step. The cost of the search is exponential in the number of moves. The final tree is the Alpha-Beta tree. This progressive refinement of the admissible heuristic toward the real Alpha-Beta is very close to what is performed when verifying the gradual games definitions as the reader can verify.

Currently, we do not use heuristics for move ordering when verifying the gradual games. It would make the search faster to use transposition tables and the killer move heuristic in the gradual games.

5 Automatic generation of rules

Retrograde analysis of patterns has been succesfully applied to the 15-puzzle [6] and to the Rubik's cube [7] to improve the accuracy of admissible heuristics. In our application to the game of Go, we rather use admissible heuristics to generate safe rules associated with external conditions by retrograde analysis [8]. A rule is a rectangular pattern associated with conditions related to the number of liberties of the objects in the pattern. The admissible number of moves is used to reduce the amount of learned rules, to generate more general rules and to ensure the validity of the generated rules. Hundreds of thousands of rules have been generated for making one eye, making two eyes and connecting strings. They give a large speed-up for our problem solver as they enable the solver to detect eyes and life many moves in advance.

In an automatically generated rule, the conditions associated with the attacker are always admissible conditions. For example, if the attacker has to be able to resist two defender moves for the capture of a string, the associated condition is that the minimum number of liberties of the string at the exterior of the pattern is three. On the contrary, the conditions associated with the defender are always upper bounds of the real value. For example, if a defender string has to be captured in one move when it has no more liberties at the interior of the pattern, the associated condition is that the string has one liberty or no liberty at the exterior of the pattern. There are never conditions on more than one liberty for the defender, as a string with two external liberties can have two eyes, and can possibly never be captured whatever the number of moves of the attacker.

The conditions for the defender strings are the maximal number of moves to remove all his liberties, and the conditions for the attacker strings are minimal number of moves to remove them. This ensures that the attacker will always be able to verify the conditions, whatever the real values are. On the contrary of admissible heuristics in one player games where there are only lower bounds on the number of moves, here we see that we need both lower and upper bounds on the number of moves.

It would be interesting to relate the time used to solve problems with the size of the generated databases and see if the same behavior is observed as in single agent search [9]. The condition that the number of external liberties of the defender is always less than one can sometimes prevent from learning useful rules. A refinement of the heuristic to take into account situations where the maximum number of moves to capture a defender string is greater than one would enable our system to generate more complex rules.

6 Automatic generation of heuristics

Introspect is a partial evaluator specialized on games [10]. It can automatically generate programs that verify the gradual game definitions described in the section on search. It uses the rules of the game written in first order logic to unfold the gradual games definitions and obtains efficient programs for gradual games. It is able to discover by partial evaluation the admissible heuristics on the number of moves. For example, in the generated programs for the capture game of Go, the number of liberties of the string is tested at first to see if the string can be captured a given number of moves ahead. The generated programs are also very selective on the moves to consider. They have knowledge very similar to the knowledge on relevant moves described in the section on search.

An important issue when using partial evaluation to unfold the rules of a game is the set of predicates used to represent the rules. Different rules representations can generate very different programs. For example, if some of the rules used to specialize the capture game contain the two commonly occuring conditions 'liberties(X, N), N $>$ 2' it is worse than containing the condition 'minliberties(X, 2)'. In the first case, the specializer will unfold the definition of liberties, and there are many rules to update liberties after a move. In the second case, the number of rules to compute the minimal number of liberties is much less. As the specializer will unfold the rules of the game as many times as there are moves in a gradual game, it will generate a very large number of rules in the first case, and only a few rules in the second case.

A related problem is the unfolding of recursive predicates. The number of liberties is a recursive function, and it is well known that unfolding recursive function has to be done very cautiously, and sometime is not to be done at all.

In the case of the number of liberties, the unfolding leads to an explosion of the number of generated rules and to worse generated programs. So the number of liberties should not be unfolded.

Given these warnings on the use of partial evaluation to generate game knowledge, it is possible to use partial evaluation to generate clever programs that select relevant moves and find admissible heuristics on moves. The unfolding on the rules of the game of the dumb algorithm that plays all the possible moves n times in a row to see a win can generate a very selective and efficient algorithm for generating moves. Another possible use of partial evaluation is to find fast way to compute the heuristics when they are given. For example, it could find a fast way to compute the shortest path between two strings given a simple and inefficient algorithm.

A more ambitious goal for Introspect is to discover more accurate heuristics. Given a simple heuristic such as the number of liberties of a string in the capture game, it could transform the definition of the heuristic to find another one which always gives a greater value. For example, a more accurate admissible heuristic for the capture game is to add the number of independent protected intersections minus one. An intersection is protected if the opponent is captured when he plays on it. It is possible to find this heuristic and some other by unfolding the definition of the heuristic with the MinMax algorithm and the rules of the games. Once this unfolding is performed, many rules are generated that give all the cases when the heuristic has underestimated the number of moves. Out of all this rules, the most simple and general can be kept. A similar method to generate admissible heuristics is to remove conditions inside the rules of the game [11, 12]. The two methods overlap, but a combination of the two might give better results than each of them.

7 Conclusion and Future Work

We have given experimental evidence that admissible heuristics on moves in two-player games account for a large speed-up for threat search algorithms. These algorithms are already much faster than basic Alpha-Beta for the games of AtariGo, Phutball and others games with frequent low order threats. The admissible heuristics we have used can also improve a depth-bounded Alpha-Beta to stop search earlier. We feel that some progress can still be made in the accuracy of these heuristics so as to improve the efficacy of current threat based search algorithms.

Admissible heuristics on the number of moves are also required in the generation of rules by retrograde analysis. Using them, our system reduces the amount of learned rules, generates more general rules and ensures the validity of the generated rules.

Some work is still needed to understand the reasons why incrementality of the computation of the admissible heuristics works well for some heuristics in some games but not in others. Improving the accuracy of our current admissible heuristics on the number of moves could speed-up our search algorithm by orders of magnitude. We are interested in experimenting threat search algorithms with different games and different admissible heuristics. Introspect, our partial evaluation system can be used to automatically generate accurate heuristics from simple ones. A special attention has to be given to the representation of the rules of the game, as different representations can lead to very different qualities of generated programs. Introspect can also be used to generate programs that can compute a given heuristic faster. In some games where it is not so easy to write admissible heuristics, it has been used as a program writing assistant that writes long, complex and efficient programs by unfolding the definition of the gradual games on the rules of the game.

References

1. Conway, J.H., Berlekamp, E., Guy, R.K.: Philosopher's football. In: Winning Ways. Academic Press (1982) 688–691
2. Allis, L.V., van den Herik, H.J., Huntjens, M.P.H.: Go-moku solved by new search techniques. Computational Intelligence **12** (1996) 7–23
3. Cazenave, T.: Abstract proof search. In Marsland, T.A., Frank, I., eds.: Computers and Games. Volume 2063 of Lecture Notes in Computer Science., Springer (2002) 39–54
4. Thomsen, T.: Lambda-search in game trees - with application to go. In Marsland, T.A., Frank, I., eds.: Computers and Games. Volume 2063 of Lecture Notes in Computer Science., Springer (2002) 19–38
5. Cazenave, T.: Gradual abstract proof search. In: Proceedings of RFIA, Angers, France (2002)
6. Culberson, J., Schaeffer, J.: Pattern databases. Computational Intelligence **14(4)** (1998) 318–334
7. Korf, R.E.: Recent progress in the design and analysis of admissible heuristic functions. In: AAAI/IAAI. (2000) 1165–1170
8. Cazenave, T.: Generation of patterns with external conditions for the game of go. In van den Herik, H., Monien, B., eds.: Advance in Computer Games 9. Universiteit Maastricht, Maastricht (2001) 275–293 ISBN 90 6216 566 4.
9. Holte, R., Hernadvolgyi, I.: Experiments with automatically created memory-based heuristics. In: SARA-2000, Lecture Notes in Artificial Intelligence N 1864 (2000) 281–290
10. Cazenave, T.: Synthesis of an efficient tactical theorem prover for the game of go. ACM Computing Surveys **30** (1998)
11. Knoblock, C.A.: Automatically generating abstractions for planning. Artificial Intelligence **68** (1994) 243–302
12. Prieditis, A., Davis, R.: Quantitatively relating abstractness to the accuracy of admissible heuristics. Artificial Intelligence **74** (1995) 165–175