Using Patterns in Computer Go

Leszek Stanisław Śliwa(✉)

The Institute of Computer Science, Faculty of Electronics and Information Technology, Warsaw University of Technology, Warsaw, Poland
l.s.sliwa@stud.elka.pw.edu.pl
http://www.ii.pw.edu.pl

Abstract. Building a good heuristics for a computer program for Go is difficult. Game tree is highly branched and there is a threat that the heuristics would eliminate strong moves. Human players often use patterns to decide where to put stones. Therefore, one of the ideas is to develop the heuristics based on the database of "good" moves denoted by patterns. A pattern is a small segment of the board. Each pattern's point can be vacant, occupied by black or white stone or can be an off-board point. A potential move is executed in the center of the pattern. Patterns can be acquired from a human expert or through machine learning. This paper presents a technique for: (1) retrieving patterns from a collection of records of games played between human expert players, (2) storing patterns, (3) implementing patterns in a computer program for Go.

Keywords: Artificial intelligence · Computer Go · MCTS · Heuristics · Patterns

1 Introduction and Overview

Construction of computer Go programs has been considered a grand challenge for Artificial Intelligence. Unlike other two-players games, computer Go programs does not play at human-master level. Ongoing research aims to develop new algorithms to replace traditional methods of game tree search. Mini-max and alpha-beta algorithms, which proved to be effective in many two-players games, does not work successfully in computer Go programs because these algorithms require a good evaluation function in order to give satisfactory results. However, a good quality positional evaluation function, to predict the game state value, has not been developed and therefore these algorithms are not able to deal with an enormous size of game tree (which is a natural consequence of the combinatorial complexity of Go game)[9].

This paper describes the automatic extraction of 3×3 patterns from professional games and is organized as follows. Section 1.1 presents the rules of Go game. Next, Subsect. 1.2 introduces the algorithm MCTS. Then, in Sect. 1.3, a short explanation of UCT selection strategy is given. Section 2 is dedicated to related works. Section 3 presents the method of automatic patterns acquisition.

M. Kryszkiewicz et al. (Eds.): PReMI 2015, LNCS 9124, pp. 565–574, 2015.
DOI: 10.1007/978-3-319-19941-2_54

Section 4 is focused on some development issues. The results of experiments are presented and discussed in Sect. 5. The recommendations for future investigations are given in Sect. 6. The research conclusions are gathered in Sect. 7.

1.1 The Game of Go

Go is one of the oldest games in the world, originated in ancient China more than 2,500 years ago. Rules of Go are quite simple. The two players alternately place black and white stones, on the vacant intersections of a board with a 19×19 grid of lines. (Computer Go is often played on smaller 9×9 and 13×13 boards). Black makes the first move, alternating with White. A player may pass his turn at any time. A stone or orthogonally connected set of stones of one color are captured and removed from the board when all points directly adjacent to it are occupied by the opponent's stones. A player may pass his turn at any time. Stones cannot be placed to repeat a prior board position. The game ends when both players consecutively pass a turn. A player who controls more territory – wins. Generally, a player's territory consists of all the board points occupied or surrounded. Several rule sets exist for the game of Go but the Japanese and the Chinese are most popular. The primary difference between rule sets is the scoring method. Chinese rules are more often implemented in computer Go due to less implementation effort.

1.2 Monte-Carlo Tree Search

In recent years, a new algorithm MCTS (Monte-Carlo Tree Search) was developed [4,10]. MCTS uses Monte-Carlo simulations as an alternative for positional evaluation function. This approach allows to achieve good results, even in the absence of expert domain knowledge. MCTS was developed initially for computer Go game, but it is a general algorithm and can be applied to solve other problems.

MCTS algorithm consists of two strategies: (1) selection applied recursively until the leaf with the highest scoring move is reached and (2) simulation that selects moves in self-play. In the first strategy, selection of the node corresponding to the most urgent move is the most important decision. The selection attempts to balance between exploitation and exploration because on the one hand, the most promising moves should be favoured and on the other, less promising moves still should be evaluated because their low scores could be a result of unfortunate coincidence (unlucky simulations). Both strategies can be facilitated by applying knowledge [6]. The simulation strategy can be enhanced by transformation of pure random simulations into partly deterministic playouts based on the knowledge. The knowledge can be designed by human experts or learned automatically.

1.3 Upper Confidence Bounds Applied to Trees

The Upper Confidence bounds applied to Trees (UCT) is a variant of the MCTS, introduced by Kocsis and Szepesvári (2006), based on the Upper Confidence Bounds (UCB) algorithm [15]. This strategy is implemented in many

programs and it is used in mine as well[1]. The UCT controls the balance between exploitation and exploration. The strategy selects moves that leads to the best results (exploitation), as well as the less promising moves, due to the uncertainty of evaluation (exploration).

2 Related Works

The idea of applying patterns in computer Go is not new. There has been quite a big effort to implement patterns in a Go engine from the early beginnings – Zobrist A.L. (1970)[18]. At the end of the eighties the combination of search, heuristics, expert systems and pattern recognition was the winning methodology. Abramson B. (1990) described pattern acquisition from random generated game records [1]. Boon M. (1990) implemented pattern matching in the computer program GOLIATH [2]. Cazenave's work (2001) consisted in automatic acquisition of patterns generated in a special tactical context – connecting or making eyes and including liberties by using explanation-based learning [5]. Erik van der Werf (2002, 2003) described a neural network approach using professional recorded games to generate local moves [17]. Another approach using Bayesian generation and the K-nearest-neighbor representation can be found in Bouzy's and Chaslot's (2005) work [3]. Gelly et al. (2006) introduced the sequence-like simulation [11]. Wang and Gelly (2007) implemented sequence-like 3×3 patterns in their Go program MOGO [16]. Coulom (2007) searched for useful patterns in Go by computing Elo ratings for patterns, improving program CRAZY STONE [8]. Chaslot et al. (2008), described different methods of learning patterns [7]. Aduard et al. (2009) showed that opening books make a big improvement in play level. Hoock and Teytaud (2010) investigated the use of Bandit-based Genetic Programming to automatically find "better" patterns that should be more often simulated for their computer program MOGO [14]. Chaslot et al. (2010) provided a comprehensive description of common patterns, tactical and strategic rules, progressive widening or progressive unpruning [6].

3 Automatic Pattern Acquisition

In this section the method of automatic pattern acquisition is described and it is structured in the following way. First the training set is introduced. Next, the coding of patterns is described. Subsequently, the collection of raw patterns is characterized. Then, clusters of patterns are introduced. Finally elimination of "poor" patterns is explained.

3.1 The Training Set

A set of Go game records has been copied from the web page [13]. It has contained records of the games played on the K amateur Go Server (KGS, formerly known

[1] Development of computer Go program has been one of goals of author's Master's thesis.

Table 1. Number of games, moves and cumulative amount of patterns identified in the training set

Tournaments	Number of games	Number of moves	Cumulated amount of patterns
Jan 2013	1,287	251,812	945
Feb 2013	842	164,641	956
Mar 2013	1,085	211,299	963
Apr 2013	1,146	227,220	970
May 2013	1,053	199,673	974
Jun 2013	1,065	204,844	976
Jul 2013	975	237,782	978
Aug 2013	1,217	267,051	984
Sep 2013	1,384	249,183	985
Oct 2013	1,252	249,183	988
Nov 2013	1,230	248,235	990
Total	12,536	2,447,567	990

as Kiseido Go Server). Participated in the tournaments players represented a master level of playing strength (6–9 dan). For the analysis, tournaments played in 2013 have been chosen – 12,536 games and 2,447,567 moves. All games have been played on 19×19 boards. Table 1 summarizes the number of games, moves and the cumulative amount of discovered patterns.

3.2 Coding of Patterns

"Go Game" computer program has been used to extract patterns from the data set. Game records have been read from SGF[2] files. For every move, a 3×3 pattern has been extracted and coded. The move has always been made in a central point of the pattern. To encode 4 possible states of every point, 2 bits have been used. Bits 00 means a point is a black stone, 01 is a white stone, 10 is an empty point, 11 is a point located outside of the board (off-board). To encode all points Formula 1 has been used, where P_{id} is a pattern's identifier and C_p is the colour code in point p.

There are 3 reasons why 3×3 patterns have been chosen: (1) cardinality of patterns set is small (2) matching is fast and (3) patterns represent enough information about connections and cutting of adjacent chains of stones.

$$P_{id} = ((((((((C_1 * 4) + C_2) * 4 + C_3) * 4 + C_4) * 4 + C_5) * 4 + C_6) * 4 + C_7) * 4 + C_8 \quad (1)$$

[2] Smart Game Format – tree-based, portable, file format used for storing records of board games.

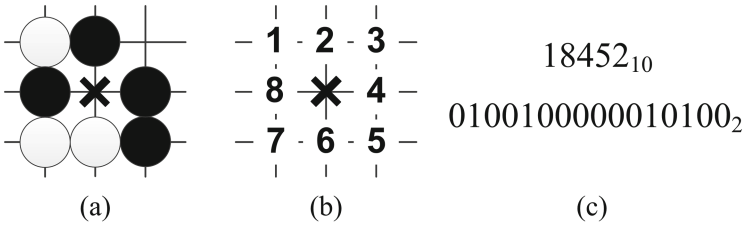

Fig. 1. Coding of 3×3 patterns

Figure 1 shows how the feature patterns have been represented: (a) 3×3 pattern, (b) the sequence of coded points, (c) a pattern's identifier in the decimal and binary code.

3.3 Collection of Raw Patterns

Recognized patterns have been saved in a disk file. Each pattern's record had 2 attributes: the pattern's identifier and the player's colour. Additionally, for every game, additional information has been registered: (1) size of the board, (2) the initials of players, (3) playing strength of opponents, (4) the date of the game, (5) *Komi* points, (6) game's rules, (7) the reason the game has been ended, and (8) the number of handicap stones[3].

3.4 Clusters of Patterns

Collected samples have been loaded from a disk file into the program memory. Then, patterns have been sequentially checked whether each item has already been registered in the database. If the pattern has not been found, it has been added to the database and its frequency counter has been set to 1. Otherwise, its frequency counter has been incremented. During the clustering, to avoid duplication of patterns, operations of: (1) rotation, (2) mirroring and (3) colour exchange have been applied. If the I is the identity, R_x is the rotation of the x degrees, H reflection horizontal and X swap colors, then the same pattern can be represented in 16 identical forms: I, R_{90}, R_{180}, R_{270}, H, $H{*}R_{90}$, $H{*}R_{180}$, $H{*}R_{270}$, $I{*}X$, $R_{90}{*}X$, $R_{180}{*}X$, $R_{270}{*}X$, $H{*}X$, $H{*}R_{90}{*}X$, $H{*}R_{180}{*}X$, $H{*}R_{270}{*}X$.

It has been assumed that the database would contain patterns only for moves made by white. If the registered pattern corresponded to a move made by black, then colour of stones has been inverted (black to white and opposite).

The majority of the patterns (945 out of a total number of 990) have been recognized during the analysis of the first tournament (Jan 2013). During further analysis the number of patterns has been growing logarithmically. On this basis, it has been concluded that: (1) training set does not have to be very large and (2) cardinality of the training set has been sufficient to build the complete database of patterns.

[3] Handicap stones – stones placed on a board to balance a game between players of significantly different playing strength.

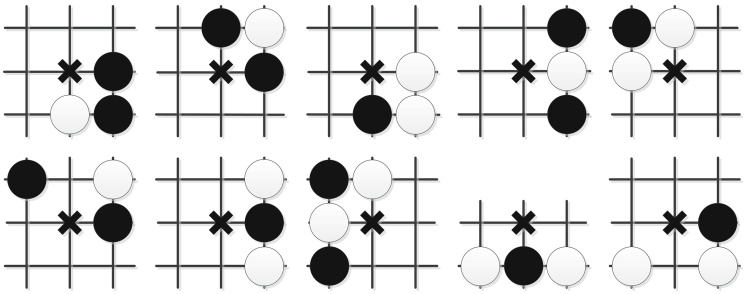

Fig. 2. Top ten high-frequency patterns for White

3.5 Refinement of the Pattern Database

There have been several reasons why collected patterns had to be corrected.

Analysed games have been played by different rules: Japanese, American, Chinese etc. According to Chinese rules a move is illegal, if one or more stones of that player's colour could be captured (prohibition of so called *suicide* move). But in analysed games such moves have been often played. Due to the fact that only Chinese rules have been implemented in "Go Game" computer program, the patterns including *suicide* would be considered as incompatible with the game rules and therefore had to be removed from the pattern database.

Some of the patterns have been found less valuable, eg. the pattern with all empty points. (Interestingly, this pattern has been played most often.) It has been assumed that such moves represented change in "the theatre of war" and attempted to "open another front". In MCTS algorithm, such a strategy is already implemented by random selection from unoccupied points. It has also been assumed that the pattern is interesting if contains at least 3 stones or off-board points.

The first trials of using patterns have shown that the developed set has contained many weak, rarely played moves that couldn't be recommended in any way as "strong". Therefore, the patterns have been sorted in descending order by the frequency counter. Patterns having the highest frequency that cover 80 % of all use cases have been selected. Other patterns have been deleted. After the last operation 231 patterns have left (from the initial set of 990). The result is close to the Pareto rule (80/20). Figure 2 shows the top ten patterns for White with highest frequency of occurrence from the final pattern database.

4 Use of Patterns in Monte-Carlo Simulations

In this section, some development issues are discussed: the algorithm, data structures, and crucial technical details having impact on the program performance.

4.1 The Algorithm and Data Structures

The set of 231 patterns has been stored in CSV file and loaded into the program's memory before start of tournaments. Patterns have been kept in memory in associative container with key-value elements. During the game, after an opponent's

move, the list of empty points adjacent to two last moves played on the board has been created. For every adjacent empty point, the corresponding pattern's id (Formula 1) has been calculated and searched in the set of "good" patterns. If several patterns have been found, the pattern having the highest frequency has been selected. The point in the center of the chosen pattern has been a recommended move.

The implemented strategy looks for an interesting move in the proximity of the last two played moves. It is based on insight that in Go game, moves are often played next to each other creating sequence of adjacent moves (mostly at a Manhattan distance of 1).

4.2 Development Details

During thinking time, the program executes tens of thousands of Monte-Carlo simulation. During tests, program with a thinking time of 3 s per move, has been performing from 50,000 simulations at the beginning of a game, up to 100,000 (and more) in the final phase (for games played on 9×9 board). Therefore time of code execution in playouts is critical because reducing the number of simulations may have a negative impact on the playing strength.

Measure of execution time of component operations executed in playouts has showed that the rotation and mirroring have been a bottleneck. Interestingly, searching patterns in the associative container has been the one of fastest operations. The problem has been solved in the following way. The database has contained only 231 patterns, each pattern could appear in maximum 8 variations (during experiments, the computer program has always been playing white), therefore the total number of patterns variations (including rotation and mirroring) has expanded to 1848. To eliminate the bottleneck operations, for fast matching, all possible combinations has been placed in the container. The tests have shown that increase in the number of items in a container from 231 to 1848 didn't affect the number of simulations.

5 Experiments

This section is focused on experiments. In Subsect. 5.1 tournaments without patterns are shortly explained. In Subsect. 5.2, results of games where proposed patterns have been used are discussed in details.

5.1 Reference Tournaments

In order to obtain reference data, tournaments on boards of size 9×9, 13×13, 15×15 and 19×19 have been played against *GnuGo* – another computer program chosen as the opponent [12]. In every tournament 100 games have been played. The final move played in the actual game has been defined by the leaf (in the search tree) with the highest visit count ("robust" leaf). UCT (Upper Confidence Bound applied to trees) strategy has been used to select leaves worth to be estimated by Monte-Carlo simulations. Thinking time has been limited to 3 seconds. The pattern database has not been used.

Table 2. Scores of tournaments verifying effectiveness of "good" patterns heuristics

Parametr		9×9		13×13		15×15		19×19	
		B[a]	W[b]	B	W	B	W	B	W
Number of won games	R[c]	2	98	21	79	35	65	62	38
	H[d]	5	95	8	92	21	79	55	45
Cumulated amount of ahead points	R	16	819	267	1121	456	1282	1700	520
	H	32	1078	150	1363	371	1355	1906	665

[a]B–black (opponent–*GnuGo* program).
[b]W–white (player–*Go Game* program).
[c]R–reference games.
[d]H–games, where"good" pattern heuristics has been used.

5.2 The Assessment of the Pattern Database Heuristics

For tournaments verifying effectiveness of patterns heuristics, computer Go program parameters have been set as for the reference tournaments (number of contents and games, board's size, final move selection, UCT strategy, thinking time), but this time "good" patterns heuristics has been used.

During simulations, the heuristics of "good" patterns could not have the highest priority. Generally, 3×3 patterns are too small to contain information about the wide context of the move. Therefore, capture and *atari* defence moves being more urgent and valuable have been considered as the first option and then followed by patterns adjacent to the last two moves played on the board. Finally, if no move has been selected so far, a point from the list of available and legal moves has been randomly selected. The scores in the tournament games have been shown in Table 2.

The final score for games played on 9×9 board has been slightly worse than the result in the reference tournament. The program has lost 5 games (in the reference tournament only 2), but the difference in the number of ahead points has increased from 819 to 1078. Thus, if the program has won, it has scored more points.

The score of games on 13×13 board has been significantly better. The program has won 13 more games than in the reference tournament. The difference in ahead points has been increased from 1121 to 1363. In games on the 15×15 board the player advantage over the opponent has been maintained. The program has won 14 more games and the difference in the number of ahead points has increased from 1282 to 1355.

Computer programs for Go based on MCTS "do not scale well". They are the most effective at small and medium boards. On large boards they give way to "traditional" programs (based on expert knowledge). In the reference tournament on 19×19 board "Go Game" has won 38 games and has lost 62 games. After the application of "good" pattern heuristics database the playing strength of both programs have been aligned. "Go Game" program has won 45 games and has lost 55.

6 Future Research

I would consider patterns of bigger size – 5×5. Some work has already been done, but first attempts based on the method described in this paper and applied to 5×5 patterns were not satisfactory. One of ideas is to use n-gram statistical pattern acquisition, similar to NLP methods.

7 Summary

This paper presents the technique for: (1) machine retrieving patterns from a collection of game records played between human expert players, (2) storing patterns and (3) implementing patterns in a computer Go program. Each pattern has been represented by a small 3×3 points piece of board with an empty point in its center, where a potential move has been considered. Every pattern identified by an integer number (pattern ID) has been given a weight depending on its frequency. Patterns have been extracted from a training data set containing 2,447,567 moves in 12,536 games (all played on 19×19 board). During the clustering, to avoid duplication caused by patterns variations, operations of: (1) rotation, (2) mirroring and (3) color exchange have been applied. For final set, patterns with highest weight, due to the Pareto rule (80/20), have been selected.

To verify how the patterns are effective and improves the playing strength of the computer program, the heuristics based on patterns has been tested in 4 contests and compared to reference tournaments. The results of experiments shows that the heuristics of "good" patterns used in MCTS, in simulations strategy, improves the playing strength on medium and large boards.

References

1. Abramson, B.: Expected-outcome: a general model of static evaluation. IEEE Trans. PAMI **12**, 182–193 (1990)
2. Boon, M.: A pattern matcher for Goliath. Comput. Go **13**, 13–23 (1990)
3. Bouzy, B. and Chaslot, G.: Bayesian generation and integration of K-nearest-neighbor patterns for 19x19 Go. In: Kendall, G., Lucas, S.(eds.) IEEE 2005 Symposium on Computational Intelligence in Games, pp. 176–181, Colchester (2005)
4. Browne, C., et al.: A survey of Monte Carlo Tree Search methods. IEEE Trans. Comput. Intell. AI Games **4**(1), 1–43 (2009)
5. Cazenave, T.: Generation of Patterns with External Conditions for the Game of Go. In: van den Herik, H.J., Monien, B.(eds.) Advances in Computer Games 9, pp. 275–293. Universiteit Maastricht, Maastricht (2001)
6. Chaslot, G., Fiter, C., Hoock, J.-B., Rimmel, A., Teytaud, O.: Adding expert knowledge and exploration in Monte-Carlo Tree Search. In: van den Herik, H.J., Spronck, P. (eds.) ACG 2009. LNCS, vol. 6048, pp. 1–13. Springer, Heidelberg (2010)
7. Chaslot, G., et al.: Progressive strategies for Monte-Carlo Tree Search. New Math. Nat. Comput. **4**(3), 343–357 (2008)

8. Coulom, R.: Computing Elo ratings of move patterns in the game of go. Int. Comp. Games Assoc. J. **30**(4), 198–208 (2007)
9. Drake, P., and Uurtamo, S.: Heuristics in Monte Carlo Go. In: International Conference on Artificial Intelligence, pp. 171–175, Las Vegas (2007)
10. Gelly, S., et al.: The grand challenge of computer go: Monte Carlo Tree Search and extensions. Commun. ACM **55**(3), 106–113 (2012)
11. Gelly, S., et al.: Modifications of UCT with Patterns in Monte-Carlo Go. Technical report 6062, INRIA (2006)
12. GNU Go 3.8. http://www.gnu.org/software/gnugo
13. Görtz, U., Game records in SGF format. http://www.u-go.net/gamerecords
14. Hoock, J.-B., Teytaud, O.: Bandit-based genetic programming. In: Esparcia-Alcázar, A.I., Ekárt, A., Silva, S., Dignum, S., Uyar, A.Ş. (eds.) EuroGP 2010. LNCS, vol. 6021, pp. 268–277. Springer, Heidelberg (2010)
15. Kocsis, L., Szepesvári, C.: Bandit based Monte-Carlo planning. In: Fürnkranz, J., Scheffer, T., Spiliopoulou, M. (eds.) ECML 2006. LNCS (LNAI), vol. 4212, pp. 282–293. Springer, Heidelberg (2006)
16. Wang, Y., Gelly, S.: Modifications of UCT and sequence-like simulations for Monte-Carlo Go. In Proceedings of IEEE Symposium on Computational Intelligence and Games, pp. 175–182, Honolulu (2007)
17. van der Werf, E., Uiterwijk, J.W.H.M., Postma, E.O., van den Herik, H.J.: Local move prediction in go. In: Schaeffer, J., Müller, M., Björnsson, Y. (eds.) CG 2002. LNCS, vol. 2883, pp. 393–412. Springer, Heidelberg (2003)
18. Zobrist, A.L.: Feature Extraction and Representation for Pattern Recognition and the Game of Go. Ph.D. thesis, University of Wisconsin, Madison (1970)