Computing Elo Ratings of Move Patterns in the Game of Go

Rémi Coulom

Computing Elo Ratings of Move Patterns in the Game of Go

Rémi Coulom

Université Charles de Gaulle, INRIA SEQUEL, CNRS GRAPPA, Lille, France

Abstract. Move patterns are an essential method to incorporate domain knowledge into Go-playing programs. This paper presents a new Bayesian technique for supervised learning of such patterns from game records, based on a generalization of Elo ratings. Each sample move in the training data is considered as a victory of a team of pattern features. Elo ratings of individual pattern features are computed from these victories, and can be used in previously unseen positions to compute a probability distribution over legal moves. In this approach, several pattern features may be combined, without an exponential cost in the number of features. Despite a very small number of training games (652), this algorithm outperforms most previous pattern-learning algorithms, both in terms of mean log-evidence (-2.69), and prediction rate (34.9%). A 19x19 Monte-Carlo program improved with these patterns reached the level of the strongest classical programs.

1 Introduction

Many Go-playing programs use domain knowledge encoded into patterns. The kinds of patterns considered in this paper are heuristic move patterns. These are general rules, such as "it is bad to play in the corner of the board", "it is good to prevent connection of two opponent strings", "don't fill-up your own eyes", or "when in atari, extend". Such knowledge may be used to prune a search tree, order moves, or improve random simulations in Monte-Carlo programs [2, 8].

Move patterns may be built by hand, or generated automatically. A popular approach to automatically generate patterns is supervised learning [1, 4, 6, 7, 9, 12–14]: frequent patterns are extracted and evaluated from game records of strong players. In this approach, expert knowledge is used to produce a relevant encoding of patterns and pattern features, and a machine-learning algorithm evaluates them. The advantage of automatic pattern learning over hand-made patterns is that thousands of patterns may be generated and evaluated with little effort, and little domain expertise.

This paper presents a new supervised pattern-learning algorithm, based on the Bradley-Terry model. The Bradley-Terry model is the theoretical basis of the Elo rating system. The principle of Elo ratings, as applied to chess, is that each player gets a numerical strength estimation, computed from the observation of past game results. From the ratings of players, it is possible to estimate a probability distribution over the outcome of future games. The same principle

Accepted at the ICGA Computer Games Workshop, Amsterdam, The Netherlands, June 2007

can be applied to move patterns: each sample move in the training database can be considered as a victory of one pattern over the others, and can be used to compute pattern ratings. When faced with a new position, the Elo ratings of patterns can be used to compute a probability distribution over all legal moves.

1.1 Related Work

This algorithm based on the Bradley-Terry model is very similar in spirit to some recent related works, but provides significant differences and improvements.

The simplest approach to pattern learning consists in measuring the frequency of play of each pattern [4, 9]. The number of times a pattern is played is divided by the number of times it is present. This way, the strongest patterns get a higher rating because they do not stay long without being played. A major weakness of this approach is that, when a move is played, the strengths of competing patterns are not taken into consideration. In the Elo-rating analogy, this would mean estimating the strength of a player with its winning rate, regardless of the strength of opponents. By taking the strength of opponents into account, methods based on the Elo rating system can compute more accurate pattern strengths.

Stern, Herbrich, and Graepel [12] address the problem of taking the strength of opponents into account by using a model extremely similar to Elo ratings. With this model, they can compute high-quality probability distributions over legal moves. A weakness of their approach, however, is that they are restricted to using only a few move features, because the number of patterns to evaluate would grow exponentially with the number of features.

In order to solve the problem of combining move features, Araki, Yoshida, Tsuruoka, and Tsujii [1] propose a method based on maximum-entropy classification. A major drawback of their approach is its very high computational cost, which forced them to learn on a restricted subset of moves, while still taking 8.75 days of computation to learn. Also, it is not clear whether their method would be able to provide a good probability distribution over moves, because, like the frequency-based approach, it doesn't take the strength of opponent patterns into account.

A generalized Bradley-Terry model, when combined with the minorization-maximization algorithm to compute its maximum likelihood, addresses all the shortcomings of previous approaches, by providing the algorithmic simplicity and efficiency of frequency-based pattern evaluation, with the power and theoretical soundness of methods based on Bayesian inference and maximum entropy.

1.2 Paper Outline

This paper is organized as follows: Section 2 explains the details of the theory of minorization-maximization and generalized Bradley-Terry models, Section 3 presents experimental results of pattern learning, and Section 4 describes how these patterns were applied to improve a Monte-Carlo program.

2 Minorization-Maximization and Generalized Bradley-Terry Models

This section briefly explains, independently of the problem of learning patterns in the game of Go, the theory of minorization-maximization and generalized Bradley-Terry models. It is based on Hunter's paper [11], where interested readers will find more generalizations of this model, with all the convergence proofs, references, and mathematical details.

2.1 Elo Ratings and the Bradley-Terry Model

The Bradley-Terry model allows to make predictions about the outcome of competitions between individuals. Its principle consists in evaluating the strength of each individual i by a positive numerical value γ_i. The stronger i, the higher γ_i. Predictions are made according to a formula that estimates the probability that i beats j:

$$P(i \text{ beats } j) = \frac{\gamma_i}{\gamma_i + \gamma_j} \ .$$

The Elo rating of individual i is defined by $r_i = 400 \log_{10}(\gamma_i)$.

2.2 Some Generalizations of the Bradley-Terry Model

The Bradley-Terry model may be generalized to handle competitions involving more than two individuals. For n players:

$$\forall i \in \{1, \ldots, n\}, \ \ P(i \text{ wins}) = \frac{\gamma_i}{\gamma_1 + \gamma_2 + \ldots + \gamma_n} \ .$$

Another interesting generalization consists in considering not only individuals, but teams. In this generalization, the γ of a team is estimated as the product of the γ's of its members. For instance:

$$P(\text{1-2-3 wins against 4-2 and 1-5-6-7}) = \frac{\gamma_1\gamma_2\gamma_3}{\gamma_1\gamma_2\gamma_3 + \gamma_4\gamma_2 + \gamma_1\gamma_5\gamma_6\gamma_7} \ .$$

Note that the same γ may appear in more than one team. But it may not appear more than once in a team.

2.3 Relevance of Bradley-Terry Models

The choice of a Bradley-Terry model makes strong assumptions about what is being modeled, and may not be appropriate in every situation. First, a Bradley-Terry model cannot take into consideration situations where individual 1 beats individual 2 consistently, individual 2 beats individual 3 consistently, and individual 3 beats individual 1 consistently. The strengths are on a one-dimensional scale, which does not allow such cycles. Also, the generalization to teams assumes that the strength of a team is the sum (in terms of Elo ratings) of the strengths of its members. This is also a very strong assumption that may not be correct all the time.

2.4 Bayesian Inference

Bradley-Terry models, as described in the previous sections, provide a probability distribution over the outcomes of future competitions, given the strength of individuals that participate. Most of the time the exact value of parameters γ_i are unknown, and have to be estimated from the outcome of past competitions. This estimation can be done with Bayesian inference.

With $\boldsymbol{\gamma}$, the vector of parameters, and $\boldsymbol{R}$, past results, Bayes formula is:

$$P(\boldsymbol{\gamma}|\boldsymbol{R}) = \frac{P(\boldsymbol{R}|\boldsymbol{\gamma})P(\boldsymbol{\gamma})}{P(\boldsymbol{R})} \ .$$

It gives a likelihood distribution over $\boldsymbol{\gamma}$, from $P(\boldsymbol{R}|\boldsymbol{\gamma})$, that is to say the Bradley-Terry model described in the previous sections, $P(\boldsymbol{\gamma})$, a prior distribution over parameters, and $P(\boldsymbol{R})$, a normalizing constant. Parameters $\boldsymbol{\gamma}$ may be estimated by finding $\boldsymbol{\gamma}^*$ that maximizes $P(\boldsymbol{\gamma}|\boldsymbol{R})$.

This optimization can be made more convenient by choosing a prior that has the same form as the Bradley-Terry model itself. That is to say, virtual results $\boldsymbol{R}'$ will serve as a prior: $P(\boldsymbol{\gamma}) = P(\boldsymbol{R}'|\boldsymbol{\gamma})$. This way, the estimation of parameters of the model will consist in maximizing $P(\boldsymbol{R}, \boldsymbol{R}'|\boldsymbol{\gamma})$.

2.5 A Minorization-Maximization Algorithm

Notations. $\gamma_1, \ldots, \gamma_n$ are the strength parameters of n individuals. N results $R_1, \ldots, R_N$ of independent competitions between these individuals are known. These competitions are of the most general type, as described in Section 2.2. The probability of one competition result may be written as

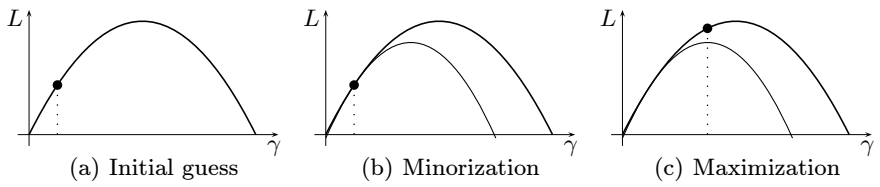
$$P(R_j) = \frac{A_{ij}\gamma_i + B_{ij}}{C_{ij}\gamma_i + D_{ij}} \ ,$$

where A_{ij}, B_{ij}, C_{ij}, and D_{ij} are factors that do not depend on γ_i. With this notation, each $P(R_j)$ can be written in n different ways, each time as a function of one particular γ_i. E_j is defined as $E_j = C_{ij}\gamma_i + D_{ij}$, and $W_i = |\{j|A_{ij} \neq 0\}|$ is the number of wins of individual i. The objective is to maximize:

$$L = \prod_{j=1}^{N} P(R_j)$$

Derivation of the Minorization-Maximization Formula. (Readers who do not wish to understand all the details may safely skip to the formula)

Minorization-maximization is an iterative algorithm to maximize L. Its principle is illustrated on Figure 1. Starting from an initial guess $\boldsymbol{\gamma}^0$ for $\boldsymbol{\gamma}$, a function m is built, that *minorizes* L at $\boldsymbol{\gamma}^0$. That is to say, $m(\boldsymbol{\gamma}^0) = L(\boldsymbol{\gamma}^0)$, and, for all $\boldsymbol{\gamma}$, $m(\boldsymbol{\gamma}) \leq L(\boldsymbol{\gamma})$. The maximum $\boldsymbol{\gamma}^1$ of m is then computed. Thanks to the minorization property, $\boldsymbol{\gamma}^1$ is an improvement over $\boldsymbol{\gamma}^0$. The trick is to build m so



(a) Initial guess (b) Minorization (c) Maximization

Fig. 1. Minorization-maximization

that its maximum can be computed in closed form. This optimization algorithm is often much more efficient than traditional gradient-ascent methods.

$$L = \prod_{j=1}^{N} \frac{A_{ij}\gamma_i + B_{ij}}{C_{ij}\gamma_i + D_{ij}}$$

is the function to be maximized. L can be considered as a function of γ_i, and its logarithm is:

$$\log L(\gamma_i) = \sum_{j=1}^{N} \log(A_{ij}\gamma_i + B_{ij}) - \sum_{j=1}^{N} \log(C_{ij}\gamma_i + D_{ij}) \ .$$

Terms that do not depend on γ_i can be removed, and, since either $B_{ij} = 0$ or $A_{ij} = 0$, the function to be maximized becomes:

$$f(x) = W_i \log x - \sum_{j=1}^{N} \log(C_{ij}x + D_{ij}) \ .$$

The logarithms in the right-hand part may be minorized by their tangent at $x = \gamma_i$, as shown on Figure 2. After removing the terms that do not depend on

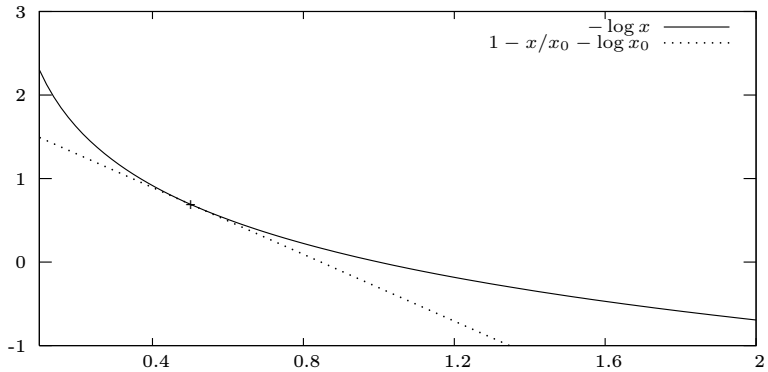


Fig. 2. Minorization of $-\log x$ at $x_0 = 0.5$ by its tangent.

x, the minorizing function to be maximized becomes

$$m(x) = W_i \log x - \sum_{j=1}^{N} \frac{C_{ij} x}{E_j} \quad .$$

The maximum of $m(x)$ is at

$$x = \frac{W_i}{\sum_{j=1}^{N} \frac{C_{ij}}{E_j}} \quad .$$

Minorization-Maximization Formula. So, minorization-maximization consists in iteratively updating one parameter γ_i according to this formula:

$$\gamma_i \leftarrow \frac{W_i}{\sum_{j=1}^{N} \frac{C_{ij}}{E_j}} \quad .$$

If all the parameters are initialized to 1, and the number of participants in each competition is the same, the first iteration of minorization-maximization computes the winning frequency of each individual. So, in some way, minorization-maximization provides a Bayesian justification of frequency-based pattern evaluation. But running more than one iteration improves parameters further.

When players have different strengths, C_{ij} indicates the strength of team mates of i during competition j, and E_j is the overall strength of participants. With the minorization-maximization formula, a win counts all the more as team mates are weak, and opposition is strong.

Batch Updates. The minorization-maximization formula describes how to update just one γ_i. It is possible to iteratively update all the γ_i one by one, but it may be inefficient. Another possibility is to perform batch updates. A set of mutually exclusive γ_i's may be updated in one single pass over the data. Mutually exclusive means that they cannot be members of the same team. The batch-update approach still has good convergence properties [11], and offers the opportunity to re-use computations. In particular, $1/E_j$ can be computed only once in a batch.

3 Pattern-Learning Experiments in the Game of Go

A generalized Bradley-Terry model can be applied to supervised learning of Go patterns, by considering that each sample move is a competition, whose winner is the move in question, and losers are the other legal moves. Each move can be considered as a "team" of features, thus allowing to combine a large number of such features without a very high cost.

3.1 Data

Learning was performed on game records played by strong players on KGS. These game records were downloaded from the web site of Kombilo [10]. The training set was made of the 652 games with no handicap of January, 2006 (131,939 moves). The test set was made of the 551 games with no handicap of February, 2006 (115,832 moves). The level of play in these games may not be as high as the professional records used in previous research on pattern learning, but they have the advantage of being publicly available for free, and their level is more than high enough for the current level of Go-playing programs.

3.2 Features

The learning algorithm used 8 tactical features: pass, capture, extension, self-atari, atari, distance to border, distance to the previous move, and distance to the move before the previous move. Some of these features may take more than one value, as explained in Table 1.

The 9th feature was Monte-Carlo owner. It was computed by running 63 random games from the current position. For each point of the board, the number of final positions owned by the player to move was counted.

The 10th feature was shape patterns. Nested circles of radius 3 to 10 according to the distance defined in Table 1 are considered, similarly to [12]. 16,780 shapes were harvested from the training set, by keeping those that appear at least 5,000 times.

Each value that these features can take is considered as a separate "individual", and is associated to one strength parameter γ_i. Since values within one feature are mutually exclusive, they were all updated together within one iteration of the minorization-maximization algorithm.

3.3 Prior

The prior was set by adding, for each γ_i, one virtual win, and one virtual loss, against a virtual opponent whose γ is 1. In the Elo-rating scale, this produces a symmetric probability distribution, with mean 0 and standard deviation 302.

3.4 Results

Table 1 lists the values of γ for all non-shape features.

Figure 3 plots the mean log-evidence per stage of the game, against the data of Stern, Herbrich, and Graepel [12]. This mean log-evidence is the mean logarithm of the probability of selecting the target move according to the Bradley-Terry model, measured over the test set. The overall mean log-evidence is -2.69, which corresponds to an average probability of 1/14.7. Uniform probability gives a mean log-evidence of -5.49, which corresponds to an average probability of 1/243.

Figure 4 is a plot of the cumulative distribution of the probability of finding the target move at a given rank, measured over the test set, and compared with other authors.

Feature	Level	γ	Description
Pass	1	0.17	Previous move is not a pass
	2	24.37	Previous move is a pass
Capture	1	30.68	String contiguous to new string in atari
	2	0.53	Re-capture previous move
	3	2.88	Prevent connection to previous move
	4	3.43	String not in a ladder
	5	0.30	String in a ladder
Extension	1	11.37	New atari, not in a ladder
	2	0.70	New atari, in a ladder
Self-atari	1	0.06	
Atari	1	1.58	Ladder atari
	2	10.24	Atari when there is a ko
	3	1.70	Other atari
Distance to border	1	0.89	
	2	1.49	
	3	1.75	
	4	1.28	
Distance to previous move	2	4.32	$d(\delta x, \delta y) = \lvert\delta x\rvert + \lvert\delta y\rvert + \max(\lvert\delta x\rvert, \lvert\delta y\rvert)$
	3	2.84	
	4	2.22	
	5	1.58	
	...	...	
	16	0.33	
	≥ 17	0.21	
Distance to the move before the previous move	2	3.08	
	3	2.38	
	4	2.27	
	5	1.68	
	...	...	
	16	0.66	
	≥ 17	0.70	
MC Owner	1	0.04	$0 - 7$
	2	1.02	$8 - 15$
	3	2.41	$16 - 23$
	4	1.41	$24 - 31$
	5	0.72	$32 - 39$
	6	0.65	$40 - 47$
	7	0.68	$48 - 55$
	8	0.13	$56 - 63$

Table 1. Model parameters for non-shape features. Each feature describes a property of a candidate move in the current position. A feature my either be absent, or take one of the values indicated in the Level column.

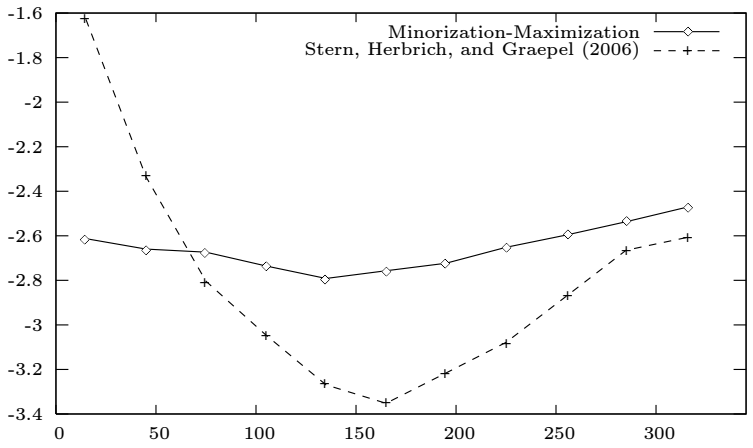


Fig. 3. Mean log-evidence per stage of the game (each point is an average over an interval of 30 moves).

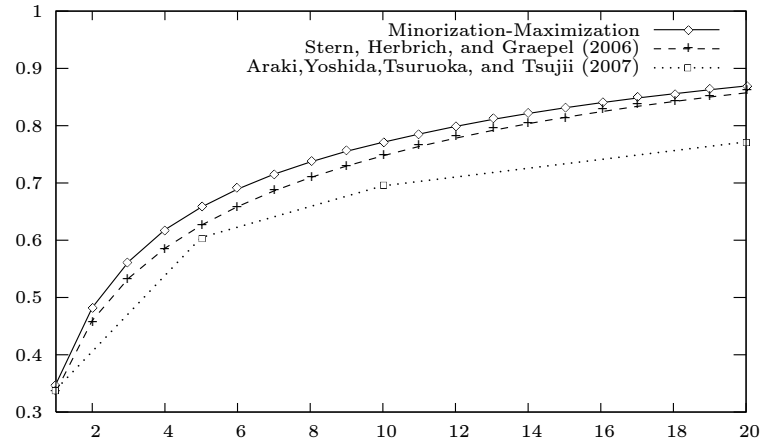


Fig. 4. Cumulative distribution: probability of finding the target move within the n best estimated moves.

3.5 Discussion

The prediction rate obtained with minorization-maximization and the Bradley-Terry model is the best among those published in academic papers. De Groot[9] claims a 42% prediction rate, so his results are still significantly better.

Despite the similarity of the cumulative distributions, the mean log-evidence per stage of the game has a very different shape from that of Stern, Herbrich, and Graepel. Their algorithm provides much better predictions in the beginning of the game, and much worse in the middle. It is worth noting also that their learning experiments used many more games (181,000 instead of 652) and shape patterns (12,000,000 instead of 16,780). So they tend to learn standard opening sequences by rote, whereas our algorithm learns more general rules.

The learning process of our algorithm is not particularly optimized, and took about one hour of CPU time and 600 Mb of RAM to complete. So it is very likely that prediction performance could be improved very easily by using more games, and more shape patterns. Most of the computation time was taken by running the Monte-Carlo simulations. In order to learn over many more games, the slow features could be trained afterward, over a small set of games.

4 Usage of Patterns in a Monte-Carlo Program

Despite the clever features of this pattern-learning system, selecting the move with the highest probability still produces a terribly weak Go player. It plays some good-looking moves, but also makes huge blunders because it really does not "understand" the position. Nevertheless, the domain knowledge contained in patterns is very precious to improve a Monte-Carlo program, by providing a good probability distribution for random games, and by helping to shape the search tree. This section briefly describes how patterns are used in Crazy Stone [5].

4.1 Random Simulations

The pattern system described in this paper produces a probability distribution over legal moves, so it is a perfect candidate for random move selection in Monte-Carlo simulations. Monte-Carlo simulations have to be very fast, so the full set of features that was described before is much too slow. Only light-weight features are kept in the learning system: 3x3 shapes, extension (without ladder knowledge), capture (without ladder knowledge), self-atari, and contiguity to the previous move. Contiguity to the previous move is a very strong feature ($\gamma = 23$), and tends to produce sequences of contiguous moves like in Mogo [8].

4.2 Progressive Widening of the Monte-Carlo Search Tree

Crazy Stone also uses patterns to prune the search tree. This is performed at a much slower rate, so the full power of complex features can be used. When a node in the Monte-Carlo search tree is created, it is searched for a while without any

pruning, selecting the move according the policy of random simulations. As soon as a number of simulations is equal to the number of points of the board, this node is promoted to internal node, and pruning is applied. Pruning consists in restricting the search to the n best moves according to patterns, with n growing like the logarithm of the number of random simulations. More precisely, the nth ($n \geq 2$) move is added when $40 \times 1.4^{n-2}$ simulations have been run. On 19x19, because of the strength of the distance-to-the-previous-move feature, progressive widening tends to produce a local search, again like in Mogo [8].

4.3 Performance against GNU Go

Table 2 summarizes Crazy Stone's performance against GNU Go 3.6. Tests were run on an AMD Opteron at 2.2 GHz, on one CPU. Crazy Stone ran, per second, from the empty position, 15,500 simulations on 9x9, and 3,700 on 19x19.

Pat.	P.W.	Size	Min./game	GNU Level	Komi	Games	Win ratio
-	-	9x9	1.5	10	6.5	170	38.2%
x	-	9x9	1.5	10	6.5	170	68.2%
x	x	9x9	1.5	10	6.5	170	90.6%
-	-	19x19	32	8	6.5	192	0.0%
x	-	19x19	32	8	6.5	192	0.0%
x	x	19x19	32	8	6.5	192	37.5%
x	x	19x19	128	8	6.5	192	57.1%

Table 2. Match results. P.W. = progressive widening. Pat. = patterns in simulations.

5 Conclusion

The research presented in this paper demonstrates that a generalized Bradley-Terry model is a very powerful technique for pattern learning in the game of Go. It is simple and efficient, can combine several features, and produces a probability distribution over legal moves. It is an ideal tool to incorporate domain knowledge into Monte-Carlo tree search.

Experiment results clearly indicate that significant progress can be made by learning shapes over a larger amount of training games, and improving features. In particular, the principle of Monte-Carlo features is very powerful, and could be exploited more, as Bouzy did with history and territory heuristics [3].

Also, the validity of the model could be tested and improved. First, using all the moves of one game as sample data breaks the hypothesis of independence between samples, since consecutive positions are very similar. Sampling one or two positions per game might be better. Also, the linearity hypothesis of the generalized Bradley-Terry model, according to which the strength of a team is the sum of the strengths of its members, is likely to be wrong. Estimating the strength of some frequent feature pairs separately might improve predictions.

Acknowledgments

I thank David Stern, Ralf Herbrich and Thore Graepel for kindly providing files with their performance data, so that I could plot mine against theirs. I am also grateful to the workshop reviewer and the readers of the computer-go mailing list for their comments that helped to improve the first version of this paper.

References

1. Nobuo Araki, Kazuhiro Yoshida, Yoshimasa Tsuruoka, and Jun'ichi Tsujii. Move prediction in Go with the maximum entropy method. In *Proceedings of the 2007 IEEE Symposium on Computational Intelligence and Games*, 2007.
2. Bruno Bouzy. Associating domain-dependent knowledge and Monte-Carlo approaches within a Go program. *Information Sciences, Heuristic Search and Computer Game Playing IV*, 175(4):247–257, November 2005.
3. Bruno Bouzy. History and territory heuristics for Monte-Carlo Go. *New Mathematics and Natural Computation*, 2(2):1–8, 2006.
4. Bruno Bouzy and Guillaume Chaslot. Bayesian generation and integration of K-nearest-neighbor patterns for 19x19 Go. In G. Kendall and Simon Lucas, editors, *IEEE Symposium on Computational Intelligence in Games*, pages 176–181, Colchester, UK, 2005.
5. Rémi Coulom. Efficient selectivity and backup operators in Monte-Carlo tree search. In P. Ciancarini and H. J. van den Herik, editors, *Proceedings of the 5th International Conference on Computer and Games*, Turin, Italy, 2006.
6. Fredrik A. Dahl. Honte, a Go-playing program using neural nets. In Johannes Fürnkranz and Miroslav Kubat, editors, *16th International Conference on Machine Learning, Workshop Notes: Machine Learning in Game Playing*, Bled, Slovenia, 1999.
7. Herbert Enderton. The Golem Go program. Technical Report CMU-CS-92-101, School of Computer Science, Carnegie-Mellon University, 1991.
8. Sylvain Gelly, Yizao Wang, Rémi Munos, and Olivier Teytaud. Modification of UCT with patterns in Monte-Carlo Go. Technical Report RR-6062, INRIA, 2006.
9. Franck de Groot. Moyo Go Studio. http://www.moyogo.com/, 2007.
10. Ulrich Görtz and William Shubert. Game records in SGF format. http://www.u-go.net/gamerecords/, 2007.
11. David R. Hunter. MM algorithms for generalized Bradley-Terry models. *The Annals of Statistics*, 32(1):384–406, 2004.
12. David Stern, Ralf Herbrich, and Thore Graepel. Bayesian pattern ranking for move prediction in the game of Go. In *Proceedings of the 23rd international conference on Machine learning*, pages 873–880, Pittsburgh, Pennsylvania, USA, 2006.
13. David Stoutamire. Machine learning, game play, and Go. Technical Report TR 91-128, Center for Automation and Intelligent Systems Research, Case Western Reserve University, 1991.
14. Erik van der Werf, Jos Uiterwijk, Eric Postma, and Jaap van den Herik. Local move prediction in Go. In Jonathan Schaeffer, Martin Müller, and Yngvi Björnsson, editors, *Computers and Games, Third International Conference, CG 2002*, pages 393–412. Springer Verlag, 2003.