Bayesian Pattern Ranking for Move Prediction in the Game of Go

David Stern
Cambridge University
Cambridge, UK
dhs26@cam.ac.uk

Ralf Herbrich
Microsoft Research Ltd.
Cambridge, UK
rherb@microsoft.com

Thore Graepel
Microsoft Research Ltd.
Cambridge, UK
thoreg@microsoft.com

Abstract

We investigate the problem of learning to predict moves in the board game of Go from game records of expert players. In particular, we obtain a probability distribution for professional play over legal moves in a given position. This distribution has numerous applications in computer Go, among them serving as a) an efficient stand-alone Go player, b) a move selector/sorter for game tree search and c) a training tool for Go players.

Our method comprises two major components: a) a pattern extraction scheme for efficiently harvesting patterns of given size and shape from expert game records and b) a Bayesian learning algorithm that learns a distribution over the values of a move given a board position based on the local pattern context. The system is trained on 20,000 expert games and shows excellent prediction performance as indicated by its ability to predict the moves made by professional Go players in 26% of test positions.

1 Introduction

Go is an ancient oriental board game of two players, 'Black' and 'White'.[1] The players take turns to place *stones* on the intersections of a grid with the aim of making territory by surrounding areas of the board. All the stones of each player are identical. Once placed, a stone is not moved but may be captured (by being surrounded with opponent stones). The resulting game is very complex and challenging. See Figure 1 for a typical Go position.

Many legal moves are usually available and it is difficult to statically estimate the value of a position. The ensuing defeat of Minimax search forces the pursuit of alternative approaches. Go has emerged as a major challenge for AI with the best computer Go programs currently playing at the level of weak amateur human players (contrast with the state of computer chess). Global search is typically replaced with a hybrid of local (goal-based) search, pattern matching and territory estimation. The most successful attempts to date have been knowledge intensive and require the management of complex board representations [2].

[1] A great deal of information about Go can be found at http://www.gobase.org.

The complexity of Go results in uncertainty about the future course and outcome of the game. Our research aims at modelling and managing this uncertainty using probability in a Bayesian sense (see also our earlier work [9]). Here we focus on the task of predicting moves made by expert Go players. In particular we wish to obtain a probability distribution over legal moves from a given board configuration. Such a distribution is useful for a) providing a stand-alone Go player that plays the moves of maximum probability, b) for move selection/sorting before performing more expensive analysis, c) as a study tool for Go. Go players frequently make moves which create known local *shapes* or satisfy other local criteria. We take advantage of this locality by matching patterns of stones centered on potential moves.

Existing Go programs use pattern matching on local configurations of stones for various purposes ranging from opening books (similar to chess) to the analysis of connectivity, life&death and territory [2]. Often, patterns may contain "don't care" points (GnuGo) or carry context-information such as the number of liberties of constituent chains [5]. Typically, the patterns are handcrafted (e.g., GnuGo) or constructed using search techniques [6]. Some attempts have been made at learning patterns from expert game records (e.g. from 2000 games in [3]), or learning to predict expert moves from various features using a neural network trained on expert games (e.g., on 25,000 moves $\sim$100 games in [10]).

Inspired by Frank de Groot's pattern system Moyogo Studio [7] we take these earlier approaches to a new level by focusing on exact local patterns for move prediction. This restriction allows us to match the patterns very efficiently, thus enabling us to train our system on tenths of thousands of games and generating moves for play very quickly. We define a pattern as an exact arrangement of stones within a sub-region of the board, centered on an empty location where a move is to be made. We choose our pattern templates as a nested sequence of increasing size so as to be able to use large patterns with greater predictive power when possible, but to be able to match smaller patterns when necessary. We automatically generate and label the patterns in two distinct processes, *harvesting* sufficiently frequent patterns from game records and *learning* a ranking of the patterns. Our probabilistic model is based on the idea that an expert in a given board configuration chooses the move-pattern that maximises a latent score. Each board configuration contains a subset of the harvested move-patterns of which the expert chooses one and thus indicates that its latent score is greater than that of the other move-patterns present. It is this information together with the fact that typical move-patterns occur in more than one position that allows the system to learn a global ranking among move-patterns—and thus to generalise across specific positions—assisted by the power of Bayesian inference.

In Section 2 we describe the process by which we automatically harvest over one million such patterns from records of expert games. In Section 3 we describe our Bayesian ranking model and the resulting method for training the move predictor from observed moves. Section 4 covers experimental results and Section 5 presents some further discussion.

2 Pattern representation, matching, and harvesting

Board and pattern representation We represent the Go board as a lattice $\mathcal{G} := \{1, \ldots, N\}^2$ where N is the board size and is usually 9 or 19. In order to represent patterns that extend across the edge of the board in a unified way, we expand the board lattice to include the off-board areas. The extended board lattice is[2] $\hat{\mathcal{G}} := \{\vec{v} + \vec{\Delta} : \vec{v} \in \mathcal{G}, \vec{\Delta} \in \mathcal{D}\}$ where the offset vectors are given by $\mathcal{D} := \{-(N-1), \ldots, (N-1)\}^2$. We define a set of "colours" $\mathcal{C} := \{\mathrm{b}, \mathrm{w}, \mathrm{e}, \mathrm{o}\}$ (black, white, empty, off). Then a board configuration is given by a colouring function $c : \hat{\mathcal{G}} \to \mathcal{C}$ and we fix the position for off-board vertices,

[2] We will use the notation $\vec{v} := (v_x, v_y)$ to represent 2-dimensional vertex vectors.

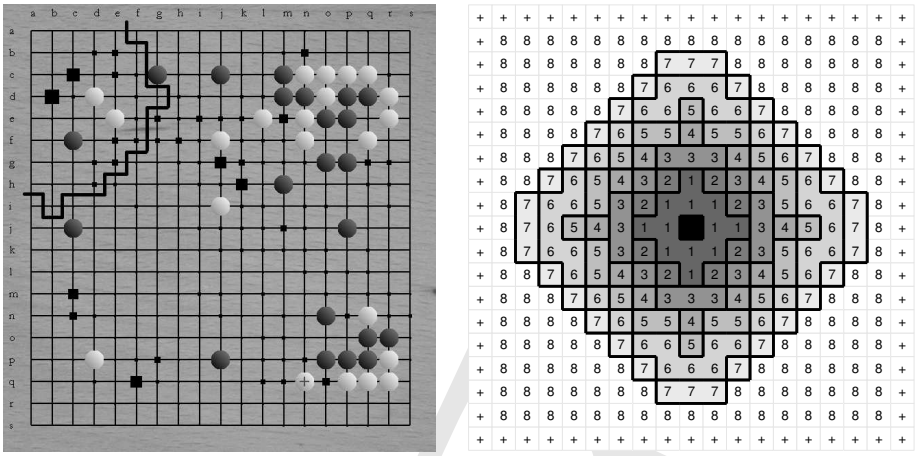

Figure 1: **Left:** Shown is a screenshot of the pattern system showing a board configuration from an expert game. The area of the black squares indicates for each vertex the probability of being the next black expert move under the model. In the top left corner pattern template T_5 is shown centred about the lower 2-4 point of that corner. **Right:** The sequence of nested pattern templates T_α with $\alpha \in \{1, \ldots, 8\}$. Note that T_8 extends beyond the plot as indicated by "+". These are the same pattern templates as used by de Groot in his Moyogo System [7].

$\forall \vec{v} \in \hat{\mathcal{G}} \setminus \mathcal{G} : c(\vec{v}) = \mathrm{o}$.

Our analysis is based on a fixed set $\mathcal{T}$ of pattern templates $T \subseteq \mathcal{T}$ on which we define a set Π of patterns $\pi : T \to \mathcal{C}$ that will be used to represent moves made in a given local context. The patterns have the following properties (see Figure 1) :

1. The pattern templates T are rotation and mirror symmetric with regard to their origin, i.e., we have that $\vec{v} \in T \Rightarrow (-v_x, v_y) \in T$ and $(v_y, -v_x) \in T$, thus displaying an 8-fold symmetry.
2. Any two pattern templates $T, T' \in \mathcal{T}$ satisfy that either $T \subset T'$ or $T' \subset T$. For convenience, we index the templates $T \in \mathcal{T}$ with the conventation that $\alpha < \beta$ implies $T_\alpha \subset T_\beta$, resulting in a nested sequence (see Figure 1 (right)).
3. We have $\pi(\vec{0}) = \mathrm{e}$ for all patterns because each pattern is to represent a legal move the centre point must be empty .
4. The set of patterns Π is closed under rotation, mirroring and colour reversal, i.e., if $\pi \in \Pi$ and π' is such that it can be generated from π by any of these transformations then $\pi' \in \Pi$. In this case, π and π' are considered equivalent, $\pi \sim \pi'$, and we define a set $\tilde{\Pi}$ of equivalence classes $\tilde{\pi}_n \subset \Pi$.[3]

We say that a *pattern* $\pi \in \Pi$ matches configuiration c at vertex $\vec{v}$ if for all $\vec{\Delta} \in T(\pi)$ we have $c(\vec{v} + \vec{\Delta}) = \pi(\vec{\Delta})$. Note that $T(\pi)$ is the template for the pattern π. We say that *pattern class* $\tilde{\pi} \in \tilde{\Pi}$ matches configuration c at vertex $\vec{v}$ if one of its constituent patterns $\pi \in \tilde{\pi}$ matches c at $\vec{v}$.

[3]Note that $\tilde{\Pi}$ is a partition of Π and thus mutually exclusive, $\bigcap_{\tilde{\pi} \in \tilde{\Pi}} \tilde{\pi}_n = \emptyset$, and exhaustive, $\bigcup_{\tilde{\pi} \in \tilde{\Pi}} \tilde{\pi}_n = \Pi$.

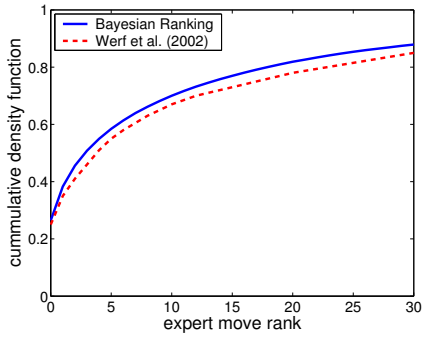

Figure 2: Cumulative distribution of the ranks the Bayesian ranking system assigns to the moves played by expert players. The Bayesian system was tested on 500 as yet unseen expert games. For comparison, we also show the corresponding curve from [10], which was obtained on 50 games from the same collection (on which we did not test our system because they were a subset of our training set).

Pattern matching and storing We do not use an explicit representation of the patterns but define a hash key for patterns and store their properties in a hash table. We use a variant of Zobrist hashing [11], which has the advantage that it can be updated incrementally. We generate four sets of 64 bit random numbers, $h_a : \hat{\mathcal{G}} \to \{0, \ldots, 2^{64} - 1\}$, $a \in \mathcal{C}$, four for each vertex in the extended Go lattice $\hat{\mathcal{G}}$. The hash-key of a given pattern π can be calculated by XORing together the corresponding random numbers,

$$k(\pi) := \bigoplus_{\vec{\Delta} \in T(\pi)} h_{\pi\left(\vec{\Delta}\right)} .$$

Both adding a stone and removing a stone of colour $a \in \{\mathrm{b}, \mathrm{w}\}$ at position $\vec{\Delta}$ correspond to the same operation $k \leftarrow k \oplus h_a$. From the commutativity of XOR it is clear that the hash-key can be calculated incrementally as stones are added or removed from a pattern. However, we would like to store the pattern classes $\tilde{\pi}$ instead of single patterns π to take account of the relevant symmetries. This is achieved by choosing $\tilde{k}(\tilde{\pi}) := \min_{\pi \in \tilde{\pi}} k(\pi)$, i.e., by calculating the hash-key for every symmetry variant of the pattern and choosing the minimum of those hash-keys. The resulting hash-table allows us to store and retrieve information associated with each pattern without an explicit representation of the pattern itself. This could be the game-record the move was found in or relevant statistics.

Pattern harvesting From a database of Go game records we harvest pattern classes $\tilde{\pi}$ corresponding to moves made by expert players. We let the computer play through each of the games in the collection and maintain a $|\mathcal{T}| \times |\hat{\mathcal{G}}|$-table $\mathbf{H}$ of hash-keys corresponding to each of the pattern templates T at each of the vertices $\vec{v} \in \hat{\mathcal{G}}$. The update after each move makes sure that if pattern class $\tilde{\pi}$ matches the resulting configuration c at vertex $\vec{v}$ then $\mathbf{H}_{\alpha,\vec{v}} = \tilde{k}(\tilde{\pi})$. Whenever an entry in $\mathbf{H}$ changes, the new hash-key can be used to mark that pattern as being present in the collection.

A rough estimate shows that for 20,000 game records with an average length of 250 moves and $|\mathcal{T}| = 8$ different pattern templates we have at most 40 million patterns at our disposal. To limit storage requirements and to ensure generalisation to as yet unseen positions we only want to include in Π those patterns that appear as a move twice in the collection. We use a Bloom filter [1] B to mark off patterns that have been seen at least once. For every pattern we observe we use B to check if it is new; if not, it is added to B. If B indicates that the pattern has been seen before we increment the count in our pattern hash-table $D_{\tilde{\Pi}}$ that represents $\tilde{\Pi}$.

3 Bayesian Pattern Ranking

Model We now present our model of the probability $P(\vec{v}|c)$ of an expert Go player making a move (at vertex) $\vec{v} \in \mathcal{G}$ in board configuration c. We only consider legal moves $\vec{v} \in \mathcal{L}(c)$, where $\mathcal{L}(c) \subseteq \mathcal{G}$ is the set of legal moves in configuration c.

A move at $\vec{v}$ in configuration c is represented by the largest pattern class $\tilde{\pi}_{\max}(\vec{v}, c) \in \Pi$ that matches c at $\vec{v}$. In our Bayesian model, we use a Gaussian belief $p(\mathbf{s}) = \mathcal{N}(\mathbf{s}; \boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma}^2))$ over scores $s(\tilde{\pi})$ of pattern classes $\tilde{\pi}$. Then the predictive distribution is given by $P(\vec{v}|c) = \int P(\vec{v}|c, \mathbf{s}) p(\mathbf{s})\, d\mathbf{s}$ (see Figure 1a) for an illustration). Our likelihood model, $P(\vec{v}|c, \mathbf{s})$, is defined via the notion of a latent, unobserved score $x(\tilde{\pi})$ for each pattern class, where $p(x|s) = \mathcal{N}(x; s, \beta^2)$ is also assumed to be Gaussian with mean s and a fixed variance β^2; the value of β expresses the variability of the score depending on specific position and player characteristics. In this sense, β can also be related to the consistency of play and could be chosen smaller for stronger players. We assume that the expert makes the move with the highest latent score value, hence,

$$\begin{aligned} P(\vec{v}|c, \mathbf{s}) &:= P\left(\forall \vec{v}' \in \mathcal{L}(c) \setminus v : x(\tilde{\pi}_{\max}(\vec{v}, c)) > x\left(\tilde{\pi}_{\max}\left(\vec{v}', c\right)\right)\right) \\ &= P\left(\mathbf{A}_{\vec{v},c}^{\mathrm{T}} \mathbf{x} \geq \mathbf{0} \,\middle|\, \mathbf{s}\right), \end{aligned} \tag{1}$$

where the matrix $\mathbf{A}_{\vec{v},c}$ has $|\mathcal{L}(c)| - 1$ many columns each of which contains $+1$ for the row indexed by pattern $\pi_{\max}(\vec{v}, c)$ and -1 for the row indexed by pattern $\pi_{\max}(\vec{v}', c)$. As an example, consider $|\mathcal{L}(c)| = 5$ legal moves, 5 patterns and the winning pattern be the third pattern, in which case we would have

$$\mathbf{A}_{\vec{v},c} := \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}.$$

The probability in (1) is the mass in the positive orthant of a (non-diagonal) Gaussian with mean $\mathbf{A}_{\vec{v},c}^{\mathrm{T}} \mathbf{s}$ and covariance $\beta^2 \mathbf{A}_{\vec{v},c}^{\mathrm{T}} \mathbf{A}_{\vec{v},c}$. This probability can be efficiently approximated by *expectation propagation (EP)* [8] using a Gaussian prior $\mathcal{N}(\mathbf{z}; \mathbf{A}_{\vec{v},c}^{\mathrm{T}} \mathbf{s}, \beta^2 \mathbf{A}_{\vec{v},c}^{\mathrm{T}} \mathbf{A}_{\vec{v},c})$ and $|\mathcal{L}(c)| - 1$ factors $t_i(\mathbf{z}) = \mathbb{I}_{z_i > 0}$.[4]

Learning and inference The goal of learning is to determine the parameters $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}^2$ of the belief distribution $p(\mathbf{s}) = \mathcal{N}(\mathbf{s}; \boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma}^2))$ from training data . The Bayesian posterior is given by

$$p(\mathbf{s}|\vec{v}, c) = \frac{P(\vec{v}|c, \mathbf{s})\, p(\mathbf{s})}{P(\vec{v}|c)} \propto \int \mathcal{N}\left(\mathbf{z}; \mathbf{A}_{\vec{v},c}^{\mathrm{T}} \mathbf{s}, \beta^2 \mathbf{A}_{\vec{v},c}^{\mathrm{T}} \mathbf{A}_{\vec{v},c}\right) \mathcal{N}\left(\mathbf{s}; \boldsymbol{\mu}, \mathrm{diag}\left(\boldsymbol{\sigma}^2\right)\right) d\mathbf{z}.$$

In general, this posterior is no longer a Gaussian and has non-zero covariance. We use the *assumed density filtering* [8] approach where we seek the best (diagonal) Gaussian approximation $q(\mathbf{s}|\vec{v}, c)$ to the posterior $p(\mathbf{s}|\vec{v}, c)$ in the sense of minimum Kullback-Leibler divergence. This requires computing the mean and covariance of the truncated Gaussian (1) which is efficiently approximated using EP. Once a move at vertex $\vec{v}$ at configuration c has been incoroporated into the prior $p(\mathbf{s})$, the posterior $p(\mathbf{s}|\vec{v}, c)$ is used as the prior for the next expert move at the new board configuration.[5]

[4] For details, please note that the classification Bayes point machine algorithm in [8] is *exactly* computing the required approximation if the data points are chosen such that the version space is the positive orthant.

[5] The numerical details of the specific ADF/EP procedure are beyond the scope of this paper and will be presented elsewhere.

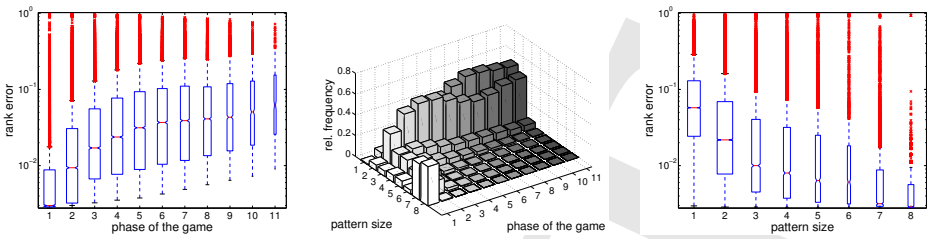

Figure 3: Test performance (on 500 games) in relation to phase of the game and pattern size. **Left:** Box plot of the rank error for different phases of the game, each phase corresponding to an interval of 30 moves. The rank error is the fraction of the rank of the professional move assigned by the algorithm over the number of legal moves. **Centre:** Series of histograms of the relative frequencies of pattern size for different phases of the game. **Right:** Box plot of the rank error for different pattern sizes $\alpha \in \{1, \ldots, 8\}$.

4 Experiments and Results

Patterns were harvested (see Section 2) from a training set of 22,000 Go games between professional Go players[6]. Starting from prior values $\boldsymbol{\mu} = \mathbf{0}$ and $\boldsymbol{\sigma} = \mathbf{1}$ the values of μ_i and σ_i for each pattern were learnt (see Section 3) from the same training set. Each move was represented by the largest pattern matched for each move. For the purpose of testing we ranked all the moves played in 500 separate expert Go games (again from GoGod) by again matching the largest pattern for every possible move and ranking the moves according to the μ values for the corrsponding patterns.

Figure 2 shows that the Bayesian ranking system ranks 26% of all expert moves first, 55% in the top 5, 68% in the top 10, and 81% in the top 20. The graph illustrates that we have improved on the performance of [10] despite the fact that we use local stone patterns only, while [10] use a number of features including stone configuration, ko activity, liberty counts, capture information, and nearby stones. In particular, it should be noted that they use the feature "distance to previous opponent move", which according to [10] stands out as an excellent feature for prediction due to the abundance of local tactical sequences, but is of dubious value for play because it encourages the resulting player to passively react to the opponent rather than to take the initiative. The box plots[7] in Figure 3 (left) compare the performance of the system at different stages of the game. The system performs extremely well at the early stages of the game where moves more commonly correspond to standard plays. The system ranks about 50% of expert moves first during the first 30 moves of the game. Figure 3 (centre) together with Figure 3 (right) provides an explanation of the excellent early-game performance: The system is more likely to match larger patterns with better predictive performance earlier in the game. Figure 3 (centre) shows that for the first 30 moves we frequently match full board patterns (size 8) which correspond to standard *Fuseki* (opening) moves. For the next 30 moves we still often match large patterns (size 6 and 7) which correspond to *Joseki* (standard corner plays). In fact, it can be argued that the system has learnt a great number of these opening patterns by the systematic assignment of values to move-patterns. Later in the game we match only smaller, less discriminative patterns (as seen in Figure 3 (right)) and hence the prediction performance decreases. Note, that in almost all cases where we match a full board pattern the resulting ranking gives a perfect prediction of expert play. There is also a strong dependence of the ranking accuracy

[6]The GoGoD database, April 2003 (http://www.gogod.demon.co.uk).

[7]Lower and upper sides of box: quartiles; vertical line across: median; width: number of data; whiskers: approximate support; dots: outliers.

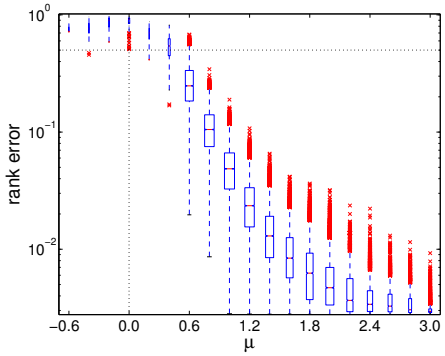

Figure 4: Box plot of the test rank error (on 500 games) for consecutive intervals of μ. The dotted horizontal line marks 50% rank error which corresponds well with the mean $\mu = 0$ of the prior distribution over score values of moves as indicated by the dotted vertical line. The plot indicates that only expert moves for which the system found a mean of $\mu > 0$ are suitable for prediction better than random.

on the mean score μ (see Figure 4). In fact, during play we can use the fact that only for patterns with $\mu_i > 0$ the prediction performance of the system is better than random. If a low μ is assigned to an expert move there were presumably many alternative moves with comparatively low μ on the board that make the identificaiton of the expert move hard.

We also tried out the ability of our system to play Go. The system currently does not excel in its playing strength due to its complete ignorance with respect to tactics. However, it surprised the authors with remarkably accurate and human-looking moves. Remarkably, the system appears to play "better" against better opponents due to its roots in expert patterns. The reader can get an idea of its playing style from the diagram in Figure 5.

5 Discussion and Conclusions

We present an application of Bayesian ranking to the problem of move prediction in the game of Go. Despite its rigid pattern definition the prediction performance of our system matches that of state-of-the-art pattern matching systems and—to a certain degree—is capable of capturing the notion of "urgency" by simultaneously considering all possible legal moves at any given time. Since we maintain a probabilistic ranking over patterns we can use our system both as a tutoring/study tool for Go players and as an efficient move selection mechanism for tree search or biased Monte Carlo Go [4]. Tracking the uncertainty of pattern scores provides our system with the added advantage of associating a confidence to the prediction of the expert move. The proposed move prediction algorithm is fast (despite the significant memory footprint due to the pattern database in memory) and we are currently working with MSN games to incorporate it as a server-side Go AI into their service (http://zone.msn.com/en/root/default.htm).

The version described in this paper is already based on an impressive training sample of $20,000$ games ($\approx 5,000,000$ moves) played by expert Go players. This limits us to $1,000,000$ harvested patterns; otherwise the number of training example per pattern would be too small. Our future work aims at building a ranked pattern database from $1,000,000$ games ($\approx 250,000,000$ moves) played between Go players of varying strength, which the system can model by varying β. Since in non-opening configurations mostly small patterns ($\alpha \in \{1, 2\}$) are active, another fruitful research direction is to encorporate context information into the small patterns to broaden their horizon.

Acknowledgements We would like to thank David MacKay, Nici Schraudolph, John Winn, Tom Minka for interesting discussions. Also, we would like to thank Erik van der Werf for his useful inputs. Finally, we would like to thank the participants of the computer Go mailing list (http://computer-go.org/mailman/listinfo/computer-go) for sharing their thoughts.

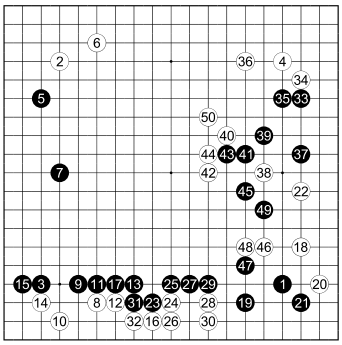

Figure 5: Diagram of the first 50 moves in a match of the Bayesian pattern ranking system against itself. Up to move 38 the game develops along standard Fuseki lines. In the remaining moves, a fight develops from White's attack on the Black group in the top right. Some of the pattern system's moves look surprisingly insightful despite the fact that they are only the result of local pattern matching and evaluation. Not surprisingly, the system wins the game.

References

[1] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[2] B. Bouzy and T. Cazenave. Computer go: An AI oriented survey. *Artificial Intelligence*, 132(1):39–103, 2001.

[3] B. Bouzy and G. Chaslot. Bayesian generation and integration of K-nearest-neighbor patterns for 19x19 go. In G. Kendall and S. Lucas, editors, *Proceedingso of the IEEE 2005 Symposium on Computational Intelligence and Games*, pages 176–181, 2005.

[4] B. Bruegmann. Monte Carlo Go, 1993.

[5] T. Cazenave. Generation of patterns with external conditions for the game of Go. In H. J. van den Herik and B. Monien, editors, *Advances of Computer Games 9*.

[6] T. Cazenave. Automatic acquisition of tactical Go rules. In *Proceedings of the Game Programming Workshop in Japan'96*, 1996.

[7] F. de Groot. Moyogo studio, 2004/2005.

[8] T. P. Minka. *A family of algorithms for approximate Bayesian inference*. PhD thesis, Massachusetts Institute of Technology, 2001.

[9] D. Stern, T. Graepel, and D. MacKay. Modelling uncertainty in the game of Go. In *Advances in Neural Information Processing Systems 16*, pages 33–40, 2004.

[10] E. van der Werf, J. Uiterwijk, E. Postma, and J. van den Herik. Local move prediction in Go. In *3rd International Conference on Computers and Games*, Edmonton, 2002.

[11] A. Zobrist. A new hashing method with applications for game playing. *ICCA Journal*, 13(2):69–73, 1990.