Bayesian Pattern Ranking for Move Prediction in the Game of Go

David Stern DHS26@CAM.AC.UK
Cambridge University, Cambridge, UK

Ralf Herbrich RHERB@MICROSOFT.COM
Thore Graepel THOREG@MICROSOFT.COM
Microsoft Research Ltd., Cambridge, UK

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 over legal moves for professional play in a given position. This distribution has numerous applications in computer Go, including serving as an efficient stand-alone Go player. It would also be effective as a move selector and move sorter for game tree search and as a training tool for Go players. Our method has 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 (in two variants) 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 181,000 expert games and shows excellent prediction performance as indicated by its ability to perfectly predict the moves made by professional Go players in 34% 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.

Appearing in *Proceedings of the 23rd International Conference on Machine Learning*, Pittsburgh, PA, 2006. Copyright 2006 by the author(s)/owner(s).

Many legal moves are typically 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. This stands in stark contrast with the state of computer chess where computers play beyond human world class level. In order to tackle computer Go, global search (as used for chess) 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 (see surveys by Bouzy and Cazenave (2001) and Müller (2002)).

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 on territory prediction in Stern et al. (2004)). 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 with maximum probability, b) for move selection and move 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 centred 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 analy-

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

sis of connectivity, life & death and territory (Bouzy & Cazenave, 2001). Often, patterns may contain 'don't care' points or carry context-information such as the number of liberties of constituent chains (see (Cazenave, 2001) and GnuGo[2]). Typically, the patterns are handcrafted or constructed using search techniques (Cazenave, 1996). Some attempts have been made at learning patterns from expert game records (e.g. from 2000 games in Bouzy and Chaslot (2005)), or learning to predict expert moves from various features using a neural network trained on expert games (e.g., on 25,000 moves in van der Werf et al. (2002)).

Inspired by the software 'Moyogo Studio' by de Groot (2005) and the work of Stoutamire (1991) we focus on exact local patterns for move prediction. This restriction allows us to match the patterns very efficiently, enabling us to train our system on hundreds of thousands of games and to generate moves for play very quickly. Following the approach of de Groot (2005) we define a pattern as an exact arrangement of stones within a sub-region of the board, centred 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. Firstly we *harvest* sufficiently frequent patterns from game records and then we *learn* a ranking of these patterns. We investigate two probabilistic models for move prediction. One is based on the idea that an expert in a given board configuration chooses the move that maximises a latent 'value'. Each board configuration contains a subset of the harvested move-patterns of which the expert chooses one and thus indicates that its latent value is greater than that of the other move-patterns present. The other model makes the assumption that every move is played with a probability independent of the other available moves.

In Section 2 we describe the process by which we automatically harvest over 12 million such patterns from records of expert games. In Section 3 we describe the two move prediction models and the resulting methods for training the move predictor from observed moves. Section 4 covers experimental results and Section 5 presents some further discussion.

2. Pattern Matching

2.1. 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[3] $\hat{\mathcal{G}} := \{\vec{v} + \vec{\Delta} : \vec{v} \in \mathcal{G}, \vec{\Delta} \in \mathcal{D}\}$ 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, $\forall \vec{v} \in \hat{\mathcal{G}} \setminus \mathcal{G} : c(\vec{v}) = \mathrm{o}$.

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 respect to their origin, i.e., we have that $(v_x, v_y) \in T \Rightarrow (-v_x, v_y) \in T$ 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 convention that $i < j$ implies $T_i \subset T_j$, resulting in a nested sequence (see Figure 1 (right)).

3. We have $\pi(\vec{0}) = \mathrm{e}$ for all patterns because for each pattern 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 $\tilde{\pi} \subset \Pi$. [4]

We say that a *pattern* $\pi \in \Pi$ matches configuration 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 $\vec{v}$.

2. See http://www.gnu.org/software/gnugo/.

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

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

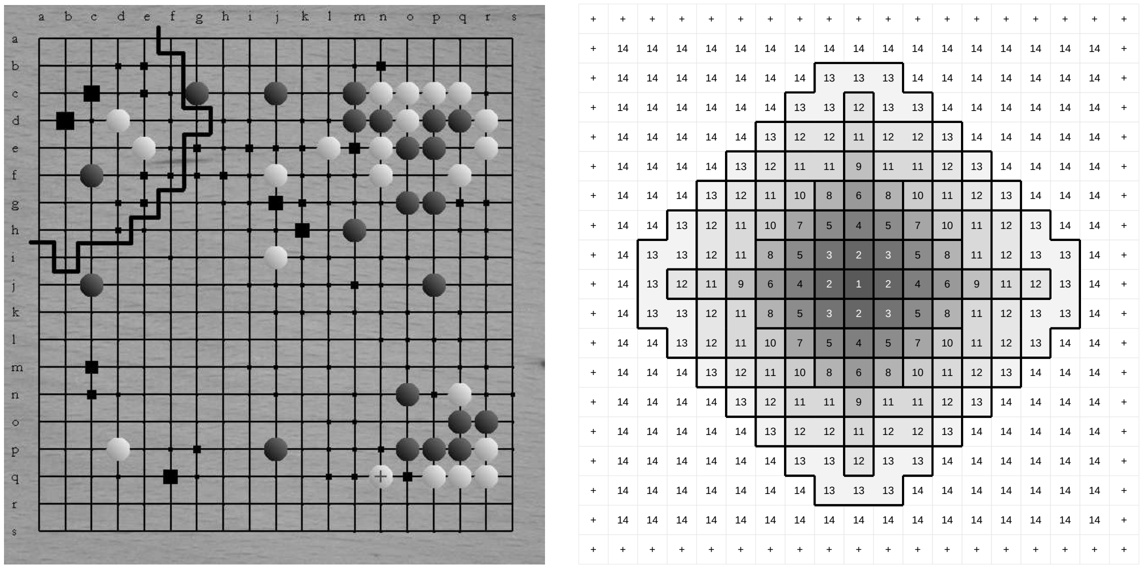

a b c d e f g h i j k l m n o p q r s
a b c d e f g h i j k l m n o p q r s

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_{11} is shown centred about the lower 2-4 (b,d) point of that corner. **Right:** The sequence of nested pattern templates T_i with $i \in \{1, \ldots, 14\}$. Note that T_{14} extends beyond the plot as indicated by "+". These are similar to the pattern templates as used by de Groot (2005).

2.2. Local Features

In order to extend the predictive power of the smaller patterns and hence improve generalisation we incorporate 8 additional binary features into each pattern. Guided by van der Werf et al. (2002) we selected the following features of a move:

- **Liberties of new chain (2 bits)** The number of liberties[5] of the chain of stones we produce by making the move. Values are $\{1, 2, 3, > 3\}$.
- **Liberties of opponent (2 bits)** The number of liberties of the closest opponent chain after making the move. Values are $\{1, 2, 3, > 3\}$.
- **Ko (1 bit)** Is there an active *Ko*[6]?
- **Escapes atari (1 bit)** Does this move bring a chain out of *atari*[7]?
- **Distance to edge (2 bits)** Distance of move to the board edge. Values are $\{< 3, 4, 5, > 5\}$.

We define the set of the labels of these features as $\mathcal{F} = \{1, ..., 8\}$. Given a move $\vec{v}$ in position c the function $f_c : \mathcal{F} \times \mathcal{G} \to \{1, 0\}$ maps each feature to its binary true/false value. For the larger patterns these features are already seen in the arrangement of stones within the template region so the larger patterns are less likely to be altered by the addition of these features.

2.3. 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 (Zobrist, 1990), 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 $\hat{\mathcal{G}}$. We also generate a random number for each of the features (see Section 2.2), $l : \mathcal{F} \to \{0, \ldots, 2^{64} - 1\}$. The hash-key, $k(\pi, \vec{v}, c)$, of a given pattern π at vertex $\vec{v}$ in board configuration c can be calculated by XORing ($\oplus$) together the corresponding random numbers,

$$k(\pi, \vec{v}, c) \quad := k_\pi \oplus k_{\vec{v},c},$$

where,

$$k_\pi := \bigoplus_{\vec{\Delta} \in T(\pi)} h_{\pi(\vec{\Delta})} \text{ and } k_{\vec{v},c} := \bigoplus_{i \in \mathcal{F}} l(i) f_c(i, \vec{v}) \, .$$

Both adding a stone and removing a stone of colour $a \in \{\mathrm{b}, \mathrm{w}\}$ at position $\vec{\Delta}$ correspond to the same oper-

5. The number of 'liberties' of a chain of stones is the lower bound on the number of opponent moves needed to capture the chain.

6. A *Ko* is a situation where a move is illegal because it would cause an earlier board position to be repeated.

7. A chain is in *atari* if it can be captured immediately.

ation $k_\pi \leftarrow k_\pi \oplus h_a$. Due to the commutativity of XOR 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. Such information may include the game-record the move was found in or relevant statistics.

2.4. 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 $|\mathcal{T}| \cdot |\hat{\mathcal{G}}|$-table $\mathbf{H}$ of hash-keys corresponding to each of the pattern templates T_i 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 $\mathbf{H}_{i,\vec{v}} = \tilde{k}(\tilde{\pi})$. 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 181,000 game records with an average length of 250 moves and $|\mathcal{T}| = 14$ different pattern templates we have about 600 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 made by an expert twice in the collection. We use a Bloom filter (Bloom, 1970) 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 so, 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. Models for Move Prediction

We now present two alternative models 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 $\vec{v}$.

3.1. Full Ranking Model

To define a full Bayesian ranking model we use a Gaussian belief $p(\mathbf{u}) = \mathcal{N}(\mathbf{u}; \boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma}^2))$ over values $u(\tilde{\pi})$ of pattern classes $\tilde{\pi}$. Then the predictive distribution is given by $P(\vec{v}|c) = \int P(\vec{v}|c, \mathbf{u}) p(\mathbf{u})\, d\mathbf{u}$ (see Figure 1 (left) for an illustration). Our model, $P(\vec{v}|c, \mathbf{u})$, is defined via the notion of a latent, unobserved value $x(\tilde{\pi})$ for each pattern class, where $p(x|u) = \mathcal{N}(x; u, \beta^2)$ is also assumed to be Gaussian with mean u and a fixed variance β^2; the quantity β expresses the variability of the value 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 value, hence,

$$P(\vec{v}|c, \mathbf{u}) := P\left(\operatorname*{argmax}_{\vec{v}' \in \mathcal{L}(c)} \{x(\tilde{\pi}_{\max}(\vec{v}', c))\} = \vec{v} \right). \quad (1)$$

Efficient inference in this model is possible by approximate message passing using *expectation propagation* as the approximation method (Minka, 2001). The factor graph (Figure 2 (left)) expresses the joint distribution $p(\vec{v}, \mathbf{u}, \mathbf{x}|c)$:

$$p(\vec{v}, \mathbf{u}, \mathbf{x}|c) = \prod_{i=1}^{n} s_i(u_i) \prod_{j=1}^{n} g_j(x_j, u_j) \prod_{k=2}^{n} h_k(x_1, x_k),$$

where

$$\begin{aligned} s_i(u_i) &= \mathcal{N}(u_i; \mu_i, \sigma_i^2), \\ g_j(x_j, u_j) &= \mathcal{N}(x_j; u_j, \beta^2), \\ h_k(x_1, x_k) &= \mathbb{I}(x_1 > x_k). \end{aligned}$$

We are interested in determining the marginals $p(u_i)$ of the joint distribution defined above. This can be accomplished by the sum-product algorithm (Jordan & Weiss, 2002).

For any variable, v_i, connected to its neighbouring factors, $f_k \in \mathrm{neigh}(v_i)$, the marginal distribution of v_i is given by

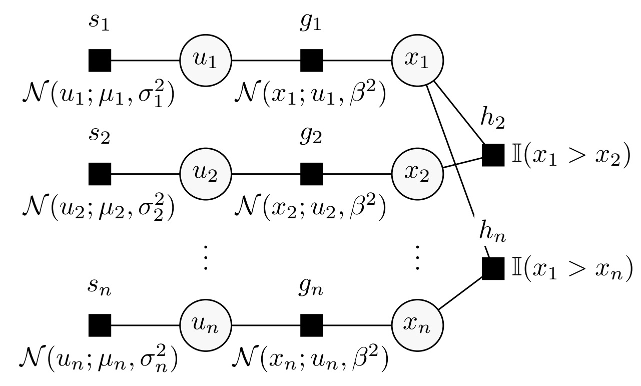
$$p(v_i) = \prod_{f_k \in \mathrm{neigh}(v_i)} m_{f_k \to v_i}(v_i), \quad (2)$$

where $m_{f_k \to v_i}(v_i)$ denotes a 'message' function passing from factor f_k to variable v_i. Messages are calculated as follows to perform exact inference on a factor tree[8]:

$$m_{f_k \to v_0}(v_0) = \int f_k([v_0; \mathbf{v}]) \prod_{j=1}^{n} m_{v_j \to f_k}(v_j) d\mathbf{v}, \quad (3)$$

$$m_{v_i \to f_k}(v_i) = \prod_{j \in \mathrm{neigh}(v_i) \setminus \{f_k\}} m_{f_k \to v_j}(v_j). \quad (4)$$

8. For notational convenience, we only state the factor-to-variable message equation for the first variable, v_0.


s_1
u_1
g_1
x_1
$\mathcal{N}(u_1; \mu_1, \sigma_1^2)$
$\mathcal{N}(x_1; u_1, \beta^2)$
h_2
$\mathbb{I}(x_1 > x_2)$
s_2
u_2
g_2
x_2
$\mathcal{N}(u_2; \mu_2, \sigma_2^2)$
$\mathcal{N}(x_2; u_2, \beta^2)$
h_n
$\mathbb{I}(x_1 > x_n)$
s_n
u_n
g_n
x_n
$\mathcal{N}(u_n; \mu_n, \sigma_n^2)$
$\mathcal{N}(x_n; u_n, \beta^2)$

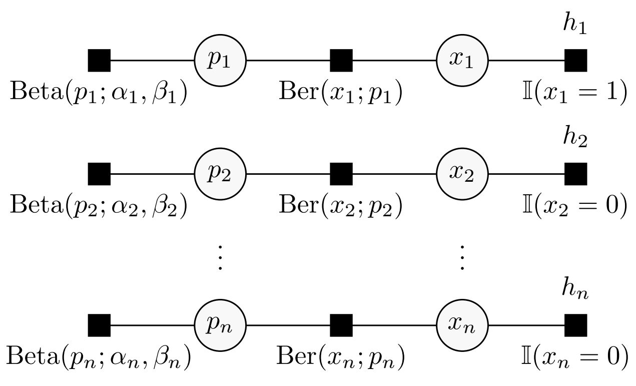


Figure 2. Factor graphs describing the two move prediction models for a particular Go position. In both cases the factors h_i encode which pattern is chosen by the expert. **Left:** Full Bayesian ranking model. **Right:** Independent Bernoulli Model (note that, $x_i \in \{0, 1\}$). We used the shorthand notation $\text{Ber}(x, p) = p^x(1-p)^{1-x}$.

These equations derive from the fact that we can make use of the conditional independence structure of the joint distribution to rearrange the marginalisation integral and thus simplify it (Jordan & Weiss, 2002).

We make the approximation that all messages are Gaussian densities to reduce storage requirements (messages can be represented by two numbers) and to simplify the calculations. For factors f_k of general form, the factor-to-variable messages calculated by (3) are not necessarily Gaussian, e.g. h_i in Figure 2 (left). Therefore we approximate these messages by a Gaussian which minimises the Kullback-Leibler divergence between the marginal distribution, $p(v_i) = m_{f_k \to v_i}(v_i) \cdot m_{v_i \to f_k}(v_i)$, and its Gaussian approximation, $q(v_i) = \hat{m}_{f_k \to v_i}(v_i) \cdot m_{v_i \to f_k}(v_i)$, where $\hat{m}_{f_k \to v_i}(v_i)$ is the approximate (Gaussian) message from factor f_k to variable v_i. That is,

$$\hat{m}_{f_k \to v_i}(v_i) = \frac{\text{MM}\left[m_{f_k \to v_i}(v_i) \cdot m_{v_i \to f_k}(v_i)\right]}{m_{v_i \to f_k}(v_i)} \qquad (5)$$

where MM denotes 'Moment Match'.

The goal of learning is to determine (from training data) the parameters μ_i and σ_i^2 of the belief distribution $p(u_i) = \mathcal{N}(u_i; \mu_i, \sigma_i^2)$ for the value of each pattern. We calculate the posterior $p(\mathbf{u}|\vec{v}, c)$ by first propagating messages about the graph according to (3), (4) and (5) until convergence. The approximate posterior distributions we require are

$$p(u_i) = m_{s_i \to u_i}(u_i) \cdot m_{g_i \to u_i}(u_i). \qquad (6)$$

Once a move at vertex $\vec{v}$ at configuration c has been incorporated into the prior $p(\mathbf{u})$, the posterior $p(\mathbf{u}|\vec{v}, c)$ is used as the prior for the next expert move at the new board configuration. This approach is a form of *assumed-density filtering* (Minka, 2001).

3.2. Independent Bernoulli Model

We also consider an alternative and simpler approach where we assume that each pattern class is played independently of the other available pattern classes with probability $p_{\vec{v},c} := p(\tilde{\pi}_{\max}(\vec{v}, c))$ (the probability of the maximal pattern class matched at vertex $\vec{v}$ in position c). The probability of a move at location $\vec{v}$ in position c given the pattern probabilities, $\mathbf{p}$, is

$$p(\vec{v}|c, \mathbf{p}) = p_{\vec{v},c} \cdot \prod_{\vec{v}' \in \mathcal{L}(c) \setminus \vec{v}} (1 - p_{\vec{v}',c}).$$

Our uncertainty on the $p_{\vec{v},c}$ is modelled by a conjugate Beta prior $p(p_{\vec{v},c}) = \text{Beta}(p_{\vec{v},c}; \alpha_{\vec{v},c}, \beta_{\vec{v},c})$ so the marginal probability of a move $P(\vec{v}|c, \boldsymbol{\alpha}, \boldsymbol{\beta})$ is

$$\int p(\vec{v}|c, \mathbf{p}) \prod_{\vec{v}' \in \mathcal{L}(c)} \text{Beta}(p_{\vec{v}',c}; \alpha_{\vec{v}',c}, \beta_{\vec{v}',c}) d\mathbf{p}$$
$$= \frac{\alpha_{\vec{v},c}}{\alpha_{\vec{v},c} + \beta_{\vec{v},c}} \cdot \prod_{\vec{v}' \in \mathcal{L}(c) \setminus \vec{v}} \left(1 - \frac{\alpha_{\vec{v}',c}}{\alpha_{\vec{v}',c} + \beta_{\vec{v}',c}}\right),$$

where $\alpha_{\vec{v},c}$ corresponds to the number of times this pattern class $\tilde{\pi}_{\max}(\vec{v}, c)$ matched for a move played by an expert in the training data and $\beta_{\vec{v},c}$ corresponds to number of times the pattern class matched $\tilde{\pi}_{\max}(\vec{v}, c)$ for moves that were available but not chosen.

4. Experiments and Results

Patterns were harvested from a training set of 181,000 Go games between professional Go players. 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 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 4400 separate expert

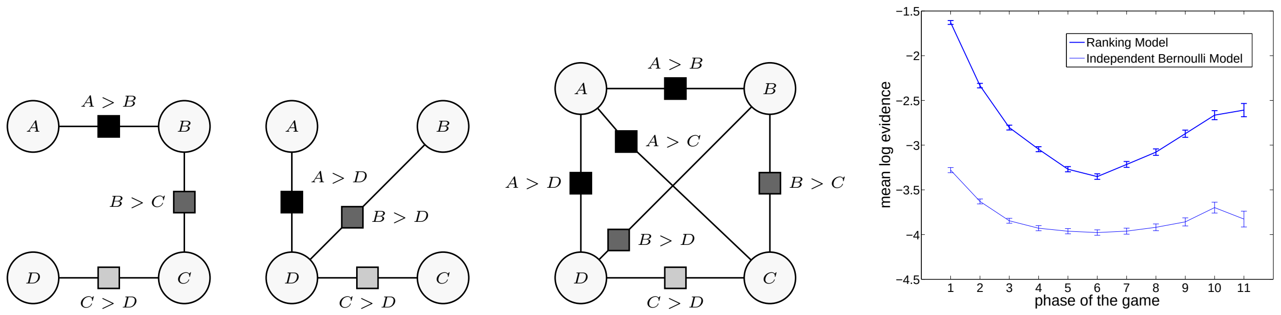


Figure 3. Each factor graph represents a set of observations of the ordering of pairs of variables. The global preference order of these variables is $A > B > C > D$ and the goal of a ranking algorithm is to infer this order from the observations. **Right:** Model evidence for the two Bayesian ranking models as a function of game phase.

Go games by again matching the largest pattern for every possible move and ranking the moves according to the μ values for the corresponding patterns.

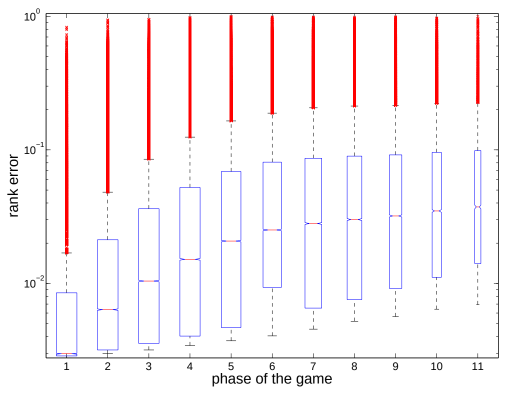
Figure 5 shows that the Bayesian (with the full ranking model) ranking system ranks 34% of all expert moves first, 66% in the top 5, 76% in the top 10, and 86% in the top 20. The graph illustrates that we have significantly improved on the performance of van der Werf et al. (2002). It is worth pointing out that 34% is an extremely high score at this task, comparable with strong human level performance. Note that strong human players will often disagree about which is the best move on the board.

The two move prediction models perform similarly despite the difference in model evidence shown in Figure 3. While this result is somewhat surprising given the simplicity of the Independent Bernoulli model it can be explained by considering different scenarios appearing in training. Figure 3 presents three hypothetical situations in which we want to infer a global ranking from limited observations of pairs of patterns from $A > B > C > D$. Each factor (box) represents an observation of which pattern is preferred in a particular pair. The first graph shows a situation where we observe a chain of preference pairs $A > B$, $B > C$ and $C > D$. The full Bayesian ranking model can infer a global ranking from these pairs. In contrast, the Independent Bernoulli model cannot infer a global ranking for the chain. It would learn the probabilities $\{A, B, C, D\} = \{1/1, 1/2, 1/2, 0/1\}$ where each fraction is (times played)/(total times seen). In the second graph the data do not provide enough information to learn a global ranking. This graph corresponds to a *joseki* sequence at the beginning of a game of Go. When a joseki move-pattern is seen it is typically preferred to any other pattern on the board and is played. It disappears from the board only to be replaced by the next joseki move-pattern in the sequence. Thus it is sufficient to learn that all joseki patterns are good to play - the ordering follows from the rules of the game. The third graph typifies an end game sequence in Go. Here, many of the different competing move-patterns co-occur on the board so the graph is fully connected and even the independent model can learn a global ranking $\{A, B, C, D\} = \{3/3, 2/3, 1/3, 0/3\}$. It is only in the first case (the chain), which does not correspond to a typical Go pattern scenario, that the full ranking model has more expressive power.

However, the full ranking model has the advantage that it provides a normalised probability distribution over moves in a given position whereas the independent model does not because it lacks the constraint that only exactly one move-pattern is played in a given position (see Figure 2). Since the full ranking model concentrates its probability mass on only the possible set of outcomes we would expect the evidence, $p(\vec{v}|c)$ to be larger, as is in fact the case. Over a test set of 750 games the log evidence for the full ranking model is $-450,000$ whereas the log evidence for the Independent Bernoulli model is $-630,000$. Figure 3 (right) shows the average log evidence as a function of the stage of the game. The ranking model outperforms the Independent Bernoulli model in every stage of the game. Both models appear to perform better at the beginning and end of the game as might be expected from the discussion above.

The box plots [9] in Figure 4 (left) compare the performance of the full ranking model 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 most

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



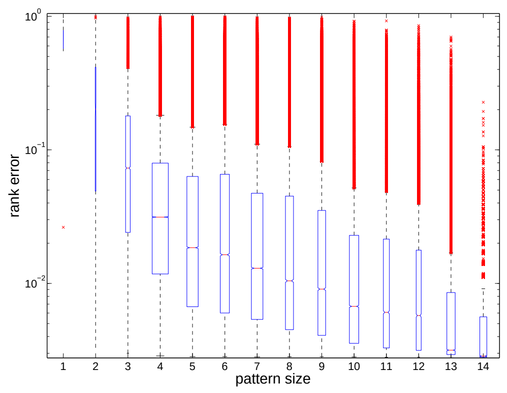


Figure 4. Test performance of the full ranking model (on 4400 games) as it depends on phase of the game and pattern size. **Left:** Box plot of the rank error for different phases of the game, each phase corresponding to an interval of 30 moves. The rank error is the fraction of the rank of the professional move assigned by the algorithm over the number of legal moves. **Right:** Box plot of the rank error for different pattern sizes.

expert moves first during the first 30 moves of the game. Figure 4 (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. For the first 30 moves we frequently match full board patterns (size 14) which correspond to standard *fuseki* (opening) moves. For the next 30 moves we still often match large patterns (size 12 and 13) which correspond to *joseki* (standard corner plays). The combination of matching large patterns and the systematic assignment of their values means that the system can reliably predict entire joseki sequences. Later in the game we match only smaller, less discriminative patterns (as seen in Figure 4 (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.

We also tested the ability of our system to play Go. In the earlier stages of the game, where playing standard sequences is prevalent, the program is perceived to be quite strong. However, it is perceived to play weaker in parts of the game where tactical reading is important. Since the system has no module for solving such local tactical search problems this is not surprising. A number of Go players, some of which high level dan players, have played the program. They found it challenging in different ways depending on their skill and estimated its playing strength between 10 and 15 kyu. One interesting observation is that the system appears to play better against better opponents due to its roots in expert patterns (see Figure 6 for an example game).

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 significantly out-performs all previously published work on this task. To a surprisingly high degree it seems capable of capturing the notion of *urgency*[10] by simultaneously considering all possible legal moves at any given time. Since we maintain a probabilistic ranking over patterns we could use our model as an efficient move selection mechanism for tree search or biased Monte Carlo Go (Brügmann, 1993). Tracking the uncertainty of pattern values provides our system with the added advantage of associating a confidence with 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).

The version described in this paper is based on a training sample of 181,000 games ($\approx$ 45,000,000 moves) played by expert Go players. This limits us to 12,000,000 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.

10. In the Go literature 'urgent' points are tactical points which should be played before moves which seem to directly increase territory and influence (so called 'big' moves). An example would be a move which prevents a group of stones being captured.

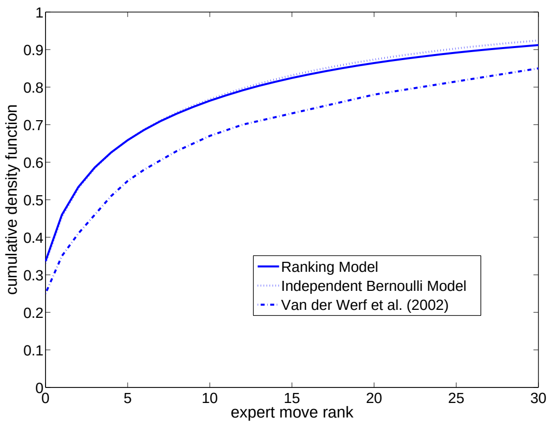


Figure 5. Cumulative distribution of the ranks the Bayesian models assigns to the moves played by expert players. The models were tested on 4400 as yet unseen expert games. For comparison, we also show the corresponding curve from van der Werf et al. (2002), which was obtained on games from the same collection.

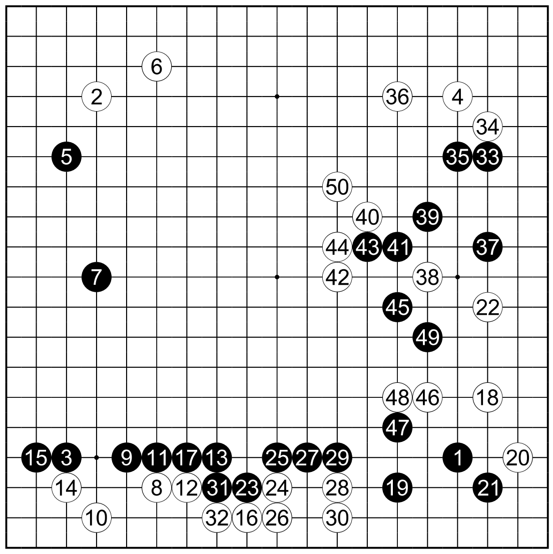


Figure 6. Diagram of the first 50 moves in a match of the Bayesian pattern ranking system against itself. Up to move 38 the game develops along standard fuseki lines. In the remaining moves, a fight develops from White's attack on the Black group in the top right. Some of the pattern system's moves look surprisingly insightful despite the fact that they are only the result of local pattern matching and evaluation.

The system performs least well in situations where tactical problem solving is required. We plan to address this by incorporating additional features (see Section 2.2): the results of running local searches to solve local problems (for example determining if a stone can be captured).

Acknowledgements

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

References

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

Bouzy, B., & Cazenave, T. (2001). Computer Go: An AI oriented survey. *Artificial Intelligence*, *132*, 39–103.

Bouzy, B., & Chaslot, G. (2005). Bayesian generation and integration of K-nearest-neighbor patterns for 19x19 Go. *Proceedingso of the IEEE 2005 Symposium on Computational Intelligence and Games* (pp. 176–181).

Brügmann, B. (1993). Monte Carlo Go ftp://ftp.cgl.ucsf.edu/pub/pett/go/ladder/mcgo.ps.

Cazenave, T. (1996). Automatic acquisition of tactical Go rules. *Proceedings of the Game Programming Workshop in Japan'96.*

Cazenave, T. (2001). Generation of patterns with external conditions for the game of Go. *Advances of Computer Games 9.*

de Groot, F. (2005). Moyogo studio http://www.moyogo.com.

Jordan, M. I., & Weiss, Y. (2002). Graphical models: probabilistic inference. In M. Arbib (Ed.), *Handbook of neural networks and brain theory.* MIT Press. 2nd edition.

Minka, T. P. (2001). *A family of algorithms for approximate Bayesian inference.* Doctoral dissertation, Massachusetts Institute of Technology.

Müller, M. (2002). Computer Go. *Artificial Intelligence*, *134*, 145–179.

Stern, D., Graepel, T., & MacKay, D. (2004). Modelling uncertainty in the game of Go. *Advances in Neural Information Processing Systems 16* (pp. 33–40).

Stoutamire, D. (1991). Machine learning applied to Go. Master's thesis, Case Western Reserve University.

van der Werf, E., Uiterwijk, J., Postma, E., & van den Herik, J. (2002). Local move prediction in Go. *3rd International Conference on Computers and Games.* Edmonton.

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