
Learning CRFs with Hierarchical Features: An Application to Go

Scott Sanner

Department of Computer Science, University of Toronto, Toronto, ON, Canada

SSANNER@CS.TORONTO.EDU

Thore Graepel

Microsoft Research Ltd., Cambridge, UK

THOREG@MICROSOFT.COM

Ralf Herbrich

Microsoft Research Ltd., Cambridge, UK

RHERB@MICROSOFT.COM

Tom Minka

Microsoft Research Ltd., Cambridge, UK

MINKA@MICROSOFT.COM

Abstract

We investigate the task of learning grid-based CRFs with hierarchical features motivated by the task of territory prediction in Go. We first analyze various independent and grid-based CRF classification models and state-of-the-art training/inference algorithms to determine which offers the best performance across a variety of metrics. Faced with the performance drawbacks of independent models and the computational drawbacks of intractable CRF models, we introduce the BMA-Tree algorithm that uses Bayesian model averaging of tree-structured predictors to exploit hierarchical feature structure. Our results demonstrate that BMA-Tree is superior to other independent classifiers and provides a computationally efficient alternative to intractable grid-based CRF models when training is too slow or approximate inference is inadequate for the task at hand.

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 and the most successful attempts to date have been knowledge in-

Appearing in the *ICML ’2007 Workshop on Constrained Optimization and Structured Output Spaces*, Corvallis, OR, 2007. Copyright 2007 by the author(s)/owner(s).

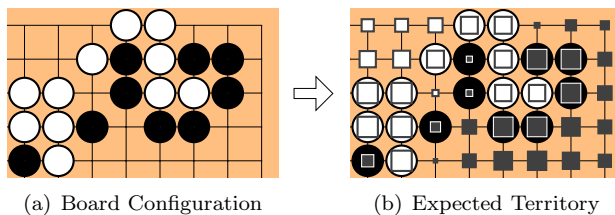


Figure 1. The territory prediction task: Given (a) the configuration of Black and White stones on the Go board, predict (b) the expected territory outcome for each board vertex (shown as superimposed squares whose size scales with the strength of the expectation).

tensive and require the management of complex board representations (see surveys by Bouzy and Cazenave (2001) and Müller (2002)).

Given the complexity of Go and the seeming impossibility of manually encoding all knowledge required for a master-level player, a machine learning approach would seem to be crucial to the success of Computer Go. And among various aspects of Go knowledge that could be learned, perhaps one of the simplest and most useful is that of territory prediction as shown in Figure 1. In this paper, we focus on a pure machine learning approach to territory prediction in Go. That is, given a set of expert training data and a set of hierarchical features, we want to find the best territory prediction algorithm w.r.t. a variety of metrics including training and inference time, memory requirements, and a range of measurements for prediction error.

Previous work by Stern et al. (2004) investigated grid-based conditional random fields (CRFs) (Lafferty et al., 2001) for the task of territory prediction in Go. We begin our work by evaluating alternate ap-

¹A wealth of additional information about Go can be found at <http://www.gobase.org>.

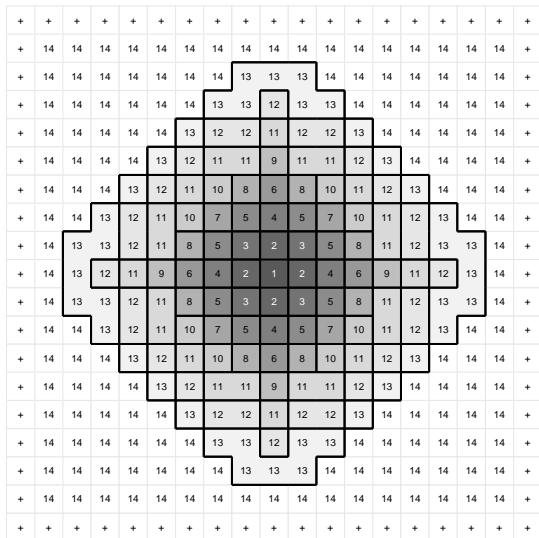


Figure 2. The sequence of nested pattern templates T_i with $i \in \{1, \dots, 14\}$; these patterns are similar to Groot (2005). T_{14} extends beyond the plot as indicated by “+”.

proaches for CRF training with augmented feature sets to improve upon their approach. Faced with the limited success of these methods, we introduce an alternate territory prediction framework based on Bayesian model averaging of tree-structured predictors (Oliver & Dowe, 1995) that offers computational efficiency and accurate score predictions in contrast to intractable grid-based CRF models.

2. Hierarchical Pattern Features

Board and Pattern Representation: Our board and pattern representation is borrowed from Stern et al. (2006) with a few modifications owing to the fact that our patterns are not move-specific, but rather general properties of any position on the board. We represent the Go board as a lattice $\mathcal{G} := \{1, \dots, N\}^2$ 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 $\hat{\mathcal{G}} := \{\vec{v} + \vec{\Delta} : \vec{v} \in \mathcal{G}, \vec{\Delta} \in \mathcal{D}\}$ where the offset vectors are given by $\mathcal{D} := \{-(N-1), \dots, (N-1)\}^2$. We define a set of “colors” $\mathcal{C} := \{b, w, e, o\}$ (black, white, empty, off). Then a board configuration is given by a coloring function $c : \hat{\mathcal{G}} \rightarrow \mathcal{C}$ and we fix the position for off-board vertices, $\forall \vec{v} \in \hat{\mathcal{G}} \setminus \mathcal{G} : c(\vec{v}) = o$.

Our analysis is based on a fixed set \mathcal{T} of pattern templates $T \subseteq \mathcal{T}$ where each T consists of a set of vertices

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

$\vec{v} \in \mathcal{D}$ relative to the origin vertex $\vec{0}$. We define a set Π of patterns $\pi : T \rightarrow \mathcal{C}$ that will be used to represent the color configuration of all vertices in T . The patterns have the following properties (see Figure 2):

- (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 $(v_x, -v_y) \in T$, thus displaying 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 of hierarchical patterns (see Figure 2).

- (3) The set of patterns is closed under rotation, mirroring and color 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 $\tilde{\pi} \subset \tilde{\Pi}$.³

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} .

Pattern Matching and Storing: We do not use an explicit representation of the patterns but define a hash key $\tilde{k}_{\tilde{\pi}}$ for pattern classes $\tilde{\pi}$ and store their properties in a hash table (e.g., relevant statistics for $\tilde{\pi}$). We use a variant of Zobrist hashing (Zobrist, 1990) that is detailed in Stern et al. (2006) to calculate $\tilde{k}_{\tilde{\pi}}$.

Pattern Harvesting: From a database of expert Go game records 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. To limit storage requirements and to ensure generalization to as yet unseen positions we limit $|\mathcal{T}| = 8$ and include only those patterns that appear in expert games at least three times in the collection.⁴ We use the filtering technique from Stern et al. (2006) to perform this task efficiently without resorting to secondary storage.

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

⁴A pattern that appears in the same position over consecutive moves is only counted once.

Territory Labeling: We define a set of final territory outcomes $\mathcal{S} := \{+1 \text{ (black)}, -1 \text{ (white)}\}$. For convenience, we score from the point of view of black so that a position is valued +1 if it belongs to Black at the end of the game, and -1 if it belongs to White. The final territory outcome for a board is given by a scoring function $s : \mathcal{G} \rightarrow \mathcal{S}$. Go players will note that we are adopting the Chinese method of scoring empty as well as occupied intersections.

In order to have training and test sets of expert games for territory prediction, we must have a final territory labeling for each of these games. Since we often do not have information in expert game records pertaining to the final agreed-upon territory outcome, we use the following simple procedure to label outcomes: From the final position in each game⁵, we play out 500 games by randomly selecting moves for each player in turn according to a uniform distribution, avoiding only those moves that would force a player to fill in its own eyes.⁶ A final territory position is considered to be black (white) *iff* all 500 simulated games resulted in that position having a black (white) stone. Otherwise a position is not labeled with an outcome to avoid training and testing w.r.t. incorrectly labeled data.

3. CRFs for Territory Prediction

3.1. Models

For notational convenience in the following presentation, let $\vec{c} = (c_1, \dots, c_{|\mathcal{G}|})$ and $\vec{s} = (s_1, \dots, s_{|\mathcal{G}|})$ respectively denote a full board configuration and a full territory outcome. For every vertex index $i \in \{1, \dots, |\mathcal{G}|\}$, we assume there exists a unique $\vec{v} \in \mathcal{G}$ such that $i \mapsto \vec{v}$. Then we can map between our 2D board representation used for patterns and vectors \vec{c} and \vec{s} as follows: $i \mapsto \vec{v} \Rightarrow c_i = c(\vec{v})$ and $i \mapsto \vec{v} \Rightarrow s_i = s(\vec{v})$.

Given a board configuration \vec{c} , we would like to model the joint probability distribution $P(\vec{s}|\vec{c})$ over all territory outcomes \vec{s} so that we can calculate the territory expectation at each vertex, $E_{P(\vec{s}|\vec{c})}[s_i]$. We can approximately factor $P(\vec{s}|\vec{c})$ in the form of a conditional random field (CRF) (Lafferty et al., 2001):

$$p(\vec{s}|\vec{c}) = \frac{1}{Z(\vec{c})} \prod_{f \in \mathcal{F}} \psi_f(\vec{s}_f, \vec{c}_f). \quad (1)$$

$Z(\vec{c})$ is the standard normalizing factor for CRFs. Each factor $f \in \mathcal{F}$ is a member of the exponential family and represents a potential over all possible con-

figurations of the subset of variables \vec{s}_f and \vec{c}_f in f . In this paper, we focus solely on two types of factors: a set of independent pattern-based factors \mathcal{F}_u and a set of coupling factors \mathcal{F}_c .⁷ Three different ways of combining these factors into CRF models are shown in Figures 3(a,b,c)

Independent Pattern-based Factors: We have independent factors $i \in \mathcal{F}_u$ for each vertex index i so that $\psi_i(\vec{s}, \vec{c}) = \psi_i(s_i, \vec{c})$. These factors are pattern-based so that there is one weight $\lambda_{\tilde{\pi}}$ for each $\tilde{\pi}$. For every pattern class $\tilde{\pi}$, we define an indicator function $\mathbb{I}_{\tilde{\pi}}(\vec{c}, i)$ that indicates when $\tilde{\pi}$ matches \vec{c} at vertex index i . We only provide the factor model for $s_i = +1$ since $\psi_i(s_i = -1, \vec{c}) = 1$.

$$\psi_i(s_i = +1, \vec{c}) = \exp \left(\sum_{\tilde{\pi} \in \tilde{\Pi}} \lambda_{\tilde{\pi}} \cdot \mathbb{I}_{\tilde{\pi}}(\vec{c}, i) \right) \quad (2)$$

Coupling Factors: We have coupling factors $(i, j) \in \mathcal{F}_c$ for each pair of neighboring vertices i, j so that $\psi_{(i,j)}(\vec{s}, \vec{c}) = \psi_{(i,j)}(s_i, s_j, c_i, c_j)$. We assign a weight λ_k to each of the 36 distinct configurations of these factors and use an indicator \mathbb{I}_k for each configuration:⁸

$$\psi_{(i,j)}(s_i, s_j, c_i, c_j) = \exp \left(\sum_{k=1}^{36} \lambda_k \cdot \mathbb{I}_k(s_i, s_j, c_i, c_j, k) \right). \quad (3)$$

3.2. Inference

Exact Inference: It is efficient (and trivial) to perform exact inference in the Independent Classifiers model. However, exact inference in our Coupling CRF models are impractical due to prohibitive time requirements. See Stern et al. (2004) for a discussion.

Loopy Belief Propagation: Where exact inference is intractable, *loopy belief propagation (BP)* (Weiss, 1997) is a *biased* approximate inference algorithm that is nonetheless widely used for its relative efficiency and reasonable empirical performance in many domains.

Sampling Methods: While sampling methods for CRF inference are unbiased, they tend to be much slower than Loopy BP. Stern et al. (2004) provide a discussion of *Gibbs* and *Swendsen-Wang* sampling.

3.3. Training

Maximum Likelihood: The objective of *maximum likelihood* CRF training is to maximize the following

⁷ $\mathcal{F} = \mathcal{F}_u \cup \mathcal{F}_c$ and $\mathcal{F}_u \cap \mathcal{F}_c = \emptyset$.

⁸We note that after enforcing parameter tying for symmetries in color reversal (swapping white/black) and direction reversal (swapping i, j), there are only 11 distinct parameters per factor. We also tie parameters across all factors $\psi_{(i,j)}$.

⁵Near-finished positions permit high accuracy labeling.

⁶An eye is - roughly speaking - an empty vertex surrounded in the four cardinal directions by stones of the same color or off-board vertices.

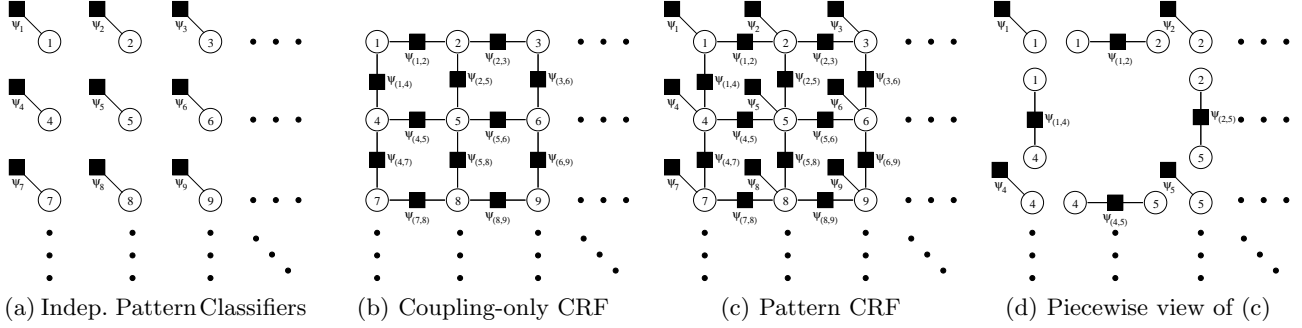


Figure 3. (a,b,c) Three different structured prediction models. We abbreviate *Indep. Pattern Classifiers* as *Indep* and *Coupling-only CRF* as *CRF*. (d) A view of the Independent + CRF model used during Piecewise training.

conditional log likelihood given a data set D :

$$l(\vec{\lambda}) = \sum_{d \in D} \left(\sum_{f \in \mathcal{F}} \log \psi_f(\vec{s}_f^{(d)}, \vec{c}_f^{(d)}) - \log Z(\vec{c}^{(d)}) \right) \quad (4)$$

When the parameters $\vec{\lambda}$ that maximize this log likelihood cannot be solved for in closed-form, one must resort to gradient ascent techniques using the following gradient that requires CRF inference for *every* $d \in D$:

$$\frac{\partial l(\vec{\lambda})}{\partial \lambda_j} = \sum_{d \in D} \left(\mathbb{I}_j(\vec{s}^{(d)}, \vec{c}^{(d)}) - \sum_{\vec{s}} \mathbb{I}_j(\vec{s}, \vec{c}^{(d)}) P(\vec{s} | \vec{c}^{(d)}) \right). \quad (5)$$

Maximum Pseudolikelihood: We define an edge-based variant of this *pseudolikelihood* that is factor-rather than variable-oriented and has been shown to have performance comparable to maximum likelihood in many cases (Sutton & McCallum, 2005). We denote the Markov blanket of factor f (i.e., the set of variables occurring in factors that *share* variables with f , non-inclusive of the variables in f) w.r.t. a set of factors \mathcal{F} as $\text{MB}_{\mathcal{F}}(f)$. This yields the following edge-based conditional log pseudolikelihood:

$$pl(\vec{\lambda}) = \sum_{d \in D} \sum_{f \in \mathcal{F}} \log P(\vec{s}_f^{(d)} | \vec{c}_f^{(d)}, \text{MB}_{\mathcal{F}}(f)^{(d)}) \quad (6)$$

The gradient of this function can be calculated similarly to Eq. 5 except that $P(\vec{s} | \vec{c}^{(d)})$ is also conditioned on the Markov blanket of the enclosing factor f , thus leading to *very* fast inference for every $d \in D$.

Piecewise Training: *Piecewise* training is a recently introduced technique for calculating CRF parameters in closed-form by decomposing the models into disjoint pieces and training each separately by maximum likelihood (Sutton & McCallum, 2005). Figure 3(d) shows part of the piecewise decomposition of Figure 3(c). A slight variant of this method known as *shared unary*

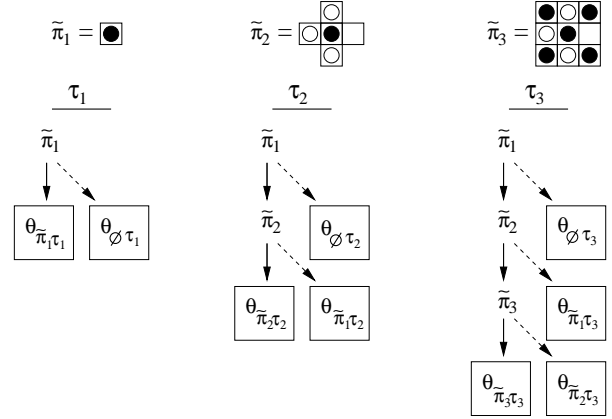


Figure 4. Hierarchical patterns organized into tree-structured Bernoulli models. A solid (dashed) line indicates a pattern $\tilde{\pi}_i$ (does not) match. Leaf nodes provide a Bernoulli distribution over the territory outcome corresponding to the respective data sorted into that leaf.

piecewise is given in Sutton and Minka (2006) where all independent factors are first trained and then duplicated among the higher-order factors to correct for the bias contributed by the independent factors.

4. Bayesian Model Averaging of Trees

Here we derive an alternative to the independent classifier model of CRFs given by Eq. 2 that specifically exploits the hierarchical pattern structure of our features. For insight, we examine Figure 4, which shows three different pattern classes $\tilde{\pi}_1$, $\tilde{\pi}_2$, and $\tilde{\pi}_3$.⁹

Now, we exploit the hierarchical nature of the patterns and organize them into three tree-structured Bernoulli models τ_1 , τ_2 and τ_3 described in Figure 4.¹⁰

⁹We assume that a pattern $\tilde{\pi}_0$ represents the empty pattern and matches every possible position.

¹⁰These trees are not stored but rather efficiently computed by starting with the largest pattern class $\tilde{\pi}$ and removing context from the pattern key $k_{\tilde{\pi}}$ to retrieve smaller

For a set of training data D , we can maintain empirical counts of respective black and white territory outcomes $c_b(\tilde{\pi}_i\tau_j)$ and $c_w(\tilde{\pi}_i\tau_j)$ for the training data sorted into each *leaf*. This allows us to derive a simple Bernoulli model of territory outcome at each leaf: $\theta_{\tilde{\pi}_i\tau_j} = c_b(\tilde{\pi}_i\tau_j)/(c_b(\tilde{\pi}_i\tau_j) + c_w(\tilde{\pi}_i\tau_j))$. The key idea behind the tree models (vs. the individual pattern models) is that the *tree models provide a prediction for all data*; this will be crucial to our following derivation.

Let us now introduce Bayesian model averaging. Given the task to predict $P(\vec{s}|\vec{c}, D)$, we derive the following:

$$P(\vec{s}|\vec{c}, D) = \prod_{j=1}^{|\mathcal{G}|} P(s_j|\vec{c}, D) = \prod_{j=1}^{|\mathcal{G}|} \sum_{\tau \in \Upsilon} P(s_j|\tau, \vec{c}, D) P(\tau|\vec{c}, D). \quad (7)$$

This equation is the essence of Bayesian model averaging – it provides a “weighted” average of each model τ ’s prediction $P(s_j|\tau, \vec{c}, D)$ over all data $d \in D$.

Let us now examine the “weight” $P(\tau|\vec{c}, D)$ – following Bayes rule, we can rewrite it as the following:

$$P(\tau|\vec{c}, D) = \frac{P(D|\tau, \vec{c})P(\tau|\vec{c})}{\sum_{\tau \in \Upsilon} P(D|\tau, \vec{c})P(\tau|\vec{c})} \quad (8)$$

Following (Oliver & Dowe, 1995), we select the “path-set” of tree-models for Υ . This is the set of trees corresponding to pruning out pattern decision nodes matching at a vertex in order from largest to smallest; our example in Figure 4 provides the path-set if $\tilde{\pi}_3$ matches a given vertex. We assume each vertex can potentially choose a different pruning of the tree model.

For the prior on trees, we assume $P(\tau|\vec{c}) = P(\tau)$ and provide results using both a uniform tree prior and an exponential tree prior recommended by (Oliver & Dowe, 1995).

Finally, we note that calculating $P(D|\tau, \vec{c})$ can be done very efficiently for any tree τ_j given that the pattern-specific counts $c_{b/w}(\tilde{\pi}_i)$ are stored for each $\tilde{\pi}_i$. Then the tree-specific leaf counts $c_{b/w}(\tilde{\pi}_i\tau_j)$ can be easily calculated due to the subset/superset relationship of all patterns. Given this data, we only need multiply each Bernoulli parameter of each leaf by its corresponding data count to compute the likelihood. We note that this likelihood can be efficiently calculated in *linear time* w.r.t. the maximum depth of the tree.

parent patterns known to be in the hash-table. These conveniences are afforded to us by the Zobrist hashing and frequency-based pattern harvesting described in Section 2.

5. Empirical Results

5.1. Algorithms

Monte Carlo: Monte Carlo algorithms for Go (Bouzy & Helmstetter, 2003) have proven to be extremely capable predictors of territory (but can be slow). The *Monte Carlo* algorithm we use for baseline comparison is identical to that used for territory labeling in Section 2 except that the final expectations for each board position are simply the average over 500 simulations.

Indep / Smallest & Largest Pattern: These are independent classifiers where pattern size is respectively limited to $|\mathcal{T}| = 1$ and the largest size $|\mathcal{T}|$ matching a vertex. The independent factors reduce to Bernoulli models, thus training is by closed-form maximum likelihood.

Indep / BMA-Tree: An independent classifier described previously in Section 4. We provide results for both *Uniform* and *Exponential* priors.

Indep / Logistic Regression: This is an independent classifier that uses maximum likelihood training of the pattern weights in Eq. 2 via gradient ascent. Exact inference is simple and efficient.

CRF / Loopy BP: Since our results for max likelihood training of the weights of a grid-based CRF model are comparable to the Boltzmann5 coupling-only model of Stern et al. (2004), we use their results for direct comparison of previous work. Inference during training was Loopy BP; inference at run-time is via loopy BP, unless otherwise specified.

Pattern CRF / (S.U.) Piecewise: This is a CRF with both independent pattern-based factors and coupling factors. Piecewise (or the shared unary variant) training was used for the coupling weights and BMA-Tree was used to train the independent weights (since it outperforms piecewise training of the independent weights). Inference at run-time is via loopy BP.

Pattern CRF / Pseudolikelihood: This is a CRF with both independent pattern-based factors and coupling factors trained via pseudolikelihood. In the *Edge* variant, independent factors were trained by logistic regression and coupling factors trained via pseudolikelihood. Inference at run-time is via loopy BP.

5.2. Summary of Results

Time and Memory: We trained the territory prediction algorithms on 4524 labeled games and tested on 462 held-out labeled games as summarized in Tables 1 and 2. The first three closed-form training methods

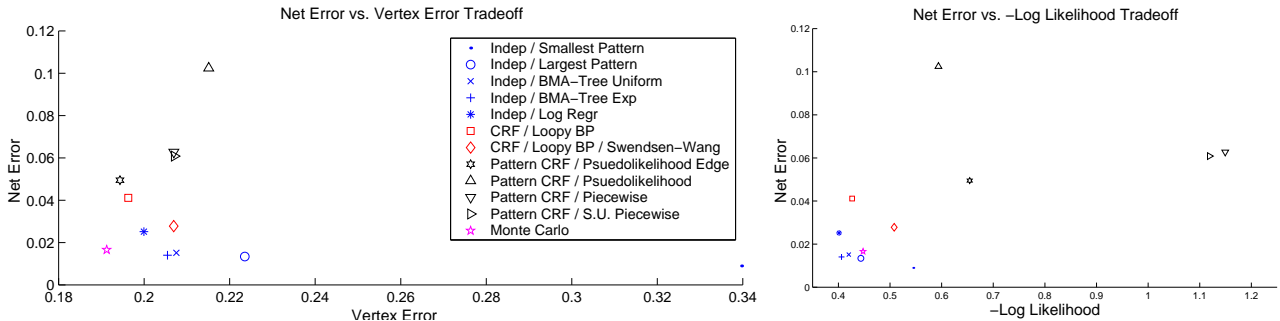


Figure 5. An analysis of the tradeoffs between Net Error, Vertex Error and -Log Likelihood. Smaller values indicate better performance and all differences are statistically significant at a 99% level. Algorithms are specified by the model / training algorithm (/ inference algorithm, if not default). See text for full explanation.

Table 1. Approximate time for various CRF models / training approaches to reach convergence.

ALGORITHM	TRAINING TIME
INDEP / LARGEST PATTERN	< 45 MIN
INDEP / BMA-TREE	< 45 MIN
PATTERN CRF / PIECEWISE	~ 2 HRS
INDEP / LOG REGR	~ 5 HRS
PATTERN CRF / PSEUDOLIKELIHOOD	~ 12 HRS
CRF / LOOPY BP	> 2 DAYS

beat all other methods while among the three slower gradient ascent methods, CRF / Loopy BP performs worst as it requires Loopy BP inference on each training example. For test performance, all inference in the Indep models required ≤ 6 ms per Go board whereas Loopy BP inference in the CRF (+ Indep) models is at least an order of magnitude more expensive. Monte Carlo and CRF / Swendsen-Wang are much slower due to sampling.

Pattern harvesting yielded 3.6 million patterns, for which pattern and parameter storage requires 432 Mb. Coupling-only CRF models do not use patterns, requiring ≤ 1 Kb for parameter storage.

Performance Comparison: There are three performance metrics that we would like to optimize:

$$\text{Vertex Error: } \frac{1}{|\mathcal{G}|} \sum_{i=1}^{|\mathcal{G}|} \mathbb{I}(\text{sgn}(E_{P(\vec{s}|\vec{c}^{(d)})}[s_i]) \neq \text{sgn}(s_i^{(d)}))$$

$$\text{Net Error: } \frac{1}{2|\mathcal{G}|} \left| \sum_{i=1}^{|\mathcal{G}|} E_{P(\vec{s}|\vec{c}^{(d)})}[s_i] - \sum_{i=1}^{|\mathcal{G}|} s_i^{(d)} \right|$$

$$\text{Log Likelihood: } \log P(\vec{s}^{(d)}|\vec{c}^{(d)})$$

Vertex Error corresponds to the average classification error per vertex while Net Error corresponds to the average error in net score calculation. Log Likelihood provides an indicator of model fit to the data. All metrics are averaged over test cases at all game stages.

In Figure 5 (left), we see a few notable trends. Monte Carlo shows perhaps the best tradeoff between Vertex

Table 2. Average time for various CRF models / inference algorithms to evaluate $P(\vec{s}|\vec{c})$ on a 19×19 Go board.

ALGORITHM	INFERENCE TIME
INDEP / SM. & LARGEST PATTERN	1.7 MS
INDEP / BMA-TREE & LOG REGR	6.0 MS
CRF / LOOPY BP	101.0 MS
PATTERN CRF / LOOPY BP	214.6 MS
MONTE CARLO	2,967.5 MS
CRF / SWENDSEN-WANG	10,568.7 MS

and Net Error and serves as our baseline prediction algorithm to beat with machine learning methods (where inference is typically also much faster). All models that could be efficiently trained by non-gradient methods exhibit low Net Error predictions with Indep / BMA-Tree Exp providing the best Vertex Error among these models. While the slower gradient training of the Indep and CRF models does offer lower Vertex Error in some cases, we see this is at the tradeoff of worse Net Error performance.

These trends can be partially explained by the comparative examples in Figure 6: In (a), we see that Indep models (in this case BMA-Tree) can make inconsistent predictions w.r.t. neighboring vertices that can be remedied by the influence of coupling factors in the more complex CRF models. While this leads to lower Vertex Error for the CRF models, we see in (b) that biased Loopy BP inference can be over-confident in its predictions when compared to unbiased inference, thus leading to increased Net Error. And finally in (c), the poor Net Error performance of Piecewise training can be seen in the compounding bias of Piecewise trained parameters and Loopy BP inference.

While one might expect Net Error to be correlated with -Log Likelihood since both reward accurate probability predictions, Figure 5 (right) shows this is not necessarily the case. In some sense, Net Error gives an

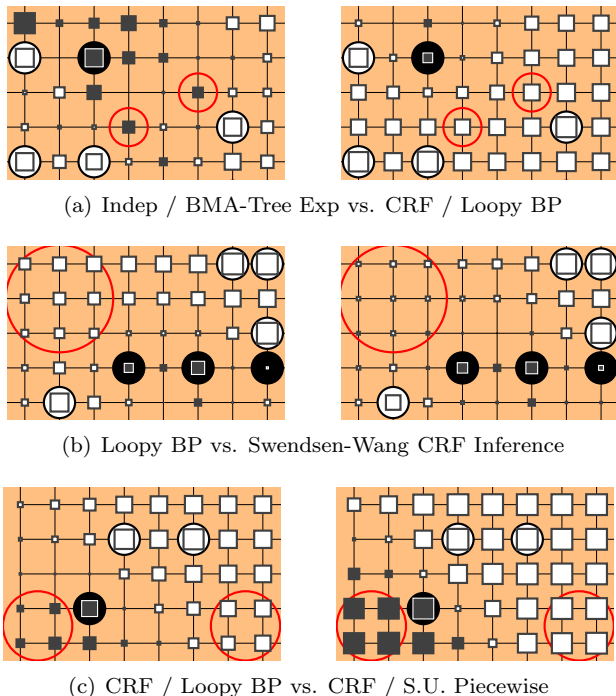


Figure 6. (a,b) Pairwise performance comparisons of CRF models / training algorithms. (c) A comparison of two inference methods on the same underlying CRF. See text for full explanation.

indicator of error correlation: if errors are highly random, then they will average out to 0 in the long run; however, if errors are correlated, as they are likely to be with the cyclic feedback of Loopy BP, an algorithm can score a low $-\text{Log Likelihood}$ but a high Net Error. Thus, aside from its utility as an accurate score prediction in Go, good performance on Net Error could be a useful additional performance measure for general structured prediction tasks as it discourages correlated errors that may be a sign of poor approximations.

6. Concluding Remarks

We have investigated the task of learning grid-based CRFs with hierarchical features motivated by the task of territory prediction in Go. Our overall results demonstrate that the BMA-Tree approach to exploiting hierarchical feature structure in an independent classification model poses a competitive alternative to grid-based CRF training when viewed in terms of its computational efficiency and predictive performance, most notably its low $-\text{Log Likelihood}$ and Net Error (important for accurate score predictions in Go).

There are two general conclusions that we can draw from these results. First, while coupled pattern CRF classifiers should theoretically be better than indepen-

dent pattern classifiers, their time cost is high and when trying to save time with approximate inference, one can suffer worse performance than by simply using independent classifiers. Secondly, when using independent pattern classifiers, the problem of choosing an appropriate neighborhood can be finessed by Bayesian averaging techniques. Altogether, the BMA-Tree algorithm provides the CRF learning community with an effective and efficient alternative to intractable grid-based CRF models when training is too slow or approximate inference is inadequate for the task at hand.

Acknowledgements

The authors thank David Stern for providing the the data and base software used to run these experiments and Mykel Kochenderfer for providing the \LaTeX source used to generate the Go board diagrams.

References

- Besag, J. E. (1975). Statistical analysis of non-lattice data. *The Statistician*, 24, 179–195.
- Bouzy, B., & Cazenave, T. (2001). Computer go: An AI oriented survey. *Artificial Intelligence*, 132, 39–103.
- Bouzy, B., & Helmstetter, B. (2003). Developments on Monte Carlo Go. *Advances in Computer Games*.
- Coulom, R. (2006). Efficient selectivity and backup operators in Monte-Carlo tree search. *5th ICCG-2006*.
- de Groot, F. (2005). Moyogo studio. <http://www.moyogo.com>.
- Lafferty, J., McCallum, A., & Pereira, F. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. *ICML-2001*.
- Müller, M. (2002). Computer go. *Artificial Intelligence*, 134, 145–179.
- Oliver, J. J., & Dowe, D. L. (1995). On pruning and averaging decision trees. *ICML-1995*.
- Stern, D., Herbrich, R., & Graepel, T. (2006). Bayesian pattern ranking for move prediction in the game of Go. *ICML-2006*.
- Stern, D. H., Graepel, T., & MacKay, D. J. C. (2004). Modelling uncertainty in the game of Go. *NIPS-2004*.
- Sutton, C., & McCallum, A. (2005). Piecewise training for undirected models. *UAI-05*.
- Sutton, C., & Minka, T. (2006). *Local training and belief propagation* (Technical Report MSR-TR-2006-121). Microsoft Research, Cambridge, UK.
- Weiss, Y. (1997). *Belief propagation and revision in networks with loops* (Technical Report). M.I.T., Cambridge, MA, USA.
- Zobrist, A. (1990). A new hashing method with applications for game playing. *ICCA Journal*, 13, 69–73.