Modelling Go Positions with Planar CRFs

Dmitry Kamenetsky D .K @ . . .
Nicol N. Schraudolph N .S @ . .
Simon Günter S .G @ . .
S.V. N. Vishwanathan SVN.V @ . .
Statistical Machine Learning, NICTA, Locked Bag 8001, Canberra ACT 2601, Australia; and
College of Engineering and Computer Science, Australian National University, Canberra ACT 0200, Australia

1. Introduction

We apply Conditional Random Fields (CRFs) to territory prediction in the game of Go. We propose a two-stage graph reduction of the Go position, with the first stage used for CRF parameter estimation and the second for inference. The interaction potentials in our model are calculated from generic shape features; the associated parameters are shared between semantically equivalent feature types. Our experiments indicate that this architecture is very efficient at propagating relevant information across the graph.

Go is a board game that originated in China over 4000 years ago. Two players (black and white) alternate in placing stones on the intersections of the grid. Once a stone is placed it cannot be moved, but it can be removed (*captured*) if all its empty neighbours (*liberties*) are occupied by opponent stones. Neighboring stones of the same color form a contiguous *block*. Players aim to create blocks that cannot be captured; the area these occupy counts as their final *territory*. Territory prediction is thus an important subproblem in Go: given a board position, determine which player *controls* each intersection.

2. Graph Abstraction for Go Positions

Grid graph. Go positions are most naturally represented as graphs, suggesting the use of graphical models. The board itself defines a square grid graph G whose intersections can be in one of three states: *black*, *white*, or *empty* (Figure 1a).

Common fate graph. It is the properties of blocks, rather than individual stones, that determines their fate: blocks always live or die as a unit. It is therefore inherently wasteful to model the individual stones of a block as G does. Consequently, the *common fate graph* G_f (Enzenberger, 1996; Graepel et al., 2001) merges all black stones in a block into a single node ● and similarly all white stones into ○ (Figure 1b). Information such as the block's size and shape can be represented in the node's features.

Block graph. Empty regions do not have the common fate property, as they are usually divided up between the players; collapsing them as well results in too impoverished a representation. Not collapsing them, on the other hand, produces large, unwieldy graphs, especially early in the game. Our *block graph* G_b is a compromise between these two extremes. It is produced by collapsing empty intersections of G_f into *black surround* ■, *white surround* □ and *neutral* ⋄ depending on whether the Manhattan distance to the nearest black stone is less than, equal to, or greater than the distance to the nearest white stone (Figure 1c).

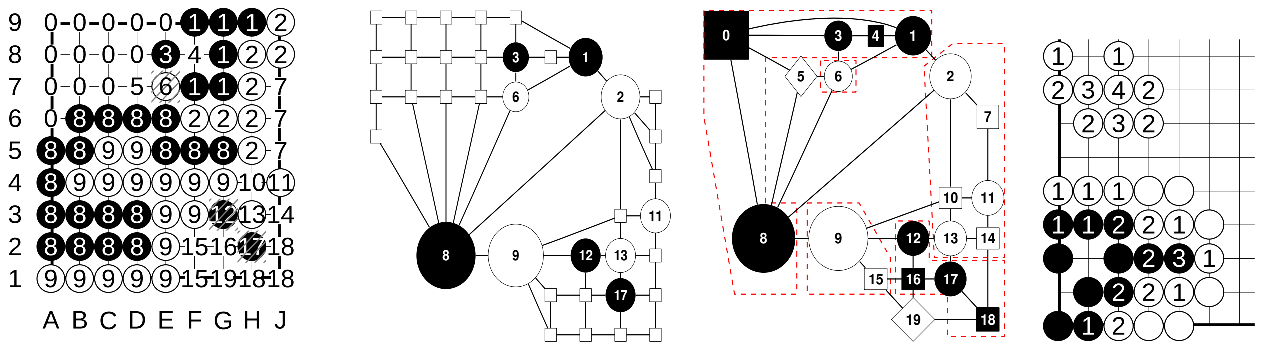
G_b provides a more succinct representation than G_f, yet by classifying empty regions into three types preserves the kind of information needed for predicting territory. For instance, in Figure 1a stones 12 and 17 are dead because they are surrounded by opponent's territory (nodes 9, 13, 14 and 15) and have little room to make life (nodes 16 and 18). We note that G_b is planar since it is derived from G, which is planar by definition. This makes G_b amenable to exact polynomial-time parameter estimation (Globerson & Jaakkola, 2007), a variant of which we employ.

Group graph. Blocks of one color that share the same surround are very unlikely to be *cut* into disjoint regions by the opponent. Since such groups of blocks share the same fate, we can group them together (dashed lines in Figure 1c). We use the resulting *group graph* G_g for inference (*i.e.*, prediction); for parameter estimation we prefer G_b because the fate of the group, albeit shared, very much depends on the shape features of its constituent blocks.

3. Features and Parameters

Nodes. For node features we adopt the *neighbour classification* of Vilà & Cazenave (2003): for each point compute the number of adjacent points that are in the same region (Figure 1d top). A region's shape feature is a vector of length four whose k^{th} element indicates the number of points with k neighbours, all in the same region. Neighbor classification provides a powerful way to summarize a region's shape, which is particularly important for empty regions. Vilà & Cazenave (2003) showed that in Go, regions with identical neighbour classification have identical eye-forming potential.

Edges. We extend the above definition to edge features: given two adjacent nodes $v_1, v_2 \in G_b$, for each point in v_1 compute the number of adjacent points that are in v_2, and *vice versa* (Figure 1d bottom). It is worth noting that our edge features provide additional information that is not conveyed by the features of the nodes they connect. This contrasts with the common approach of making edge fea-

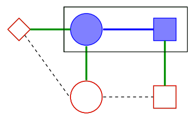


(a) G (b) G_f (c) G_b and G_g (d) Features

Figure 1: (a) A typical 9×9 board position represented as a grid graph G. Dead stones are shaded; all other stones are alive. (b) Corresponding common fate graph G_f. (c) Corresponding block graph G_b. ○ represent *stones*, □ represent *surrounds*, and ◇ are *neutral*. Dashed lines indicate nodes of the group graph G_g. (d) Top: Node features, resulting in feature vector [2, 4, 2, 1]. Bottom: Edge features, resulting in feature vectors [6, 3, 0] and [3, 3, 1] for edges ● → ○ and ○ → ●, respectively.

Table 1: Parameters and features used to compute the potential of one particular edge (blue) in a small block graph (bottom left).

	Current Edge		Neighbour Edges	
	Param.	Feat.	Param.	Feat.
Nodes	$\boldsymbol{\theta}_{\circ}$	●	$\boldsymbol{\theta}^n_{\diamond}$	◇
	$\boldsymbol{\theta}_{\square}$	■	$\boldsymbol{\theta}^n_{\circ}$	○
			$\boldsymbol{\theta}^n_{\square}$	□
Edges	$\boldsymbol{\theta}_{\circ\square}$	● → ■	$\boldsymbol{\theta}^n_{\diamond\circ}$	◇ → ●
	$\boldsymbol{\theta}_{\square\circ}$	■ → ●	$\boldsymbol{\theta}^n_{\circ\diamond}$	● → ◇
			$\boldsymbol{\theta}^n_{\circ\circ}$	○ → ●
				● → ○
			$\boldsymbol{\theta}^n_{\square\square}$	□ → ■
				■ → □



tures simply some arithmetic combination of the features of their nodes.

Parameter sharing. There are three types of node parameters, one for each node type: *stone* ($\boldsymbol{\theta}_{\circ}$), *surround* ($\boldsymbol{\theta}_{\square}$), and *neutral* ($\boldsymbol{\theta}_{\diamond}$); and eight types of edge parameters resulting from all possible types of node pairings. To assist propagation of Go knowledge along the graph, for each edge we also include the features, weighted by $\boldsymbol{\theta}^n$ of its neighbouring nodes and edges. Table 1 shows the resulting features and parameters for one particular edge.

4. Results

For our experiments we use 9×9 endgame positions of van der Werf et al. (2005). Each grid point is labeled as either belonging to or . Note that we do not predict on neutral points. We use the CRF's MAP assignment as our prediction. We use 4 measures for test error: *block* — percentage of misclassified *stone* nodes in G_b, *vertex* — percentage of misclassified *non-neutral* nodes in G, *game* — percentage of games that have at least one vertex error and *winner* — percentage of games whose winner is predicted incorrectly using Chinese Scoring. The naive errors are computed by assuming that all stones are alive, *i.e.*, we label ● as , and ○ as . Our system compares favourably to other models (Table 2). Incorporating additional features and refining the division of empty regions should lead to further improvements.

Table 2: Prediction error (% and standard deviation) of different models.

	Error	
Algorithm	Block	Vertex
Naive	17.57 ± 0.40	6.79 ± 0.09
Stern et al. (2004)	7.36 ± 0.27	4.77 ± 0.22
Block graph	3.56 ± 0.19	2.36 ± 0.06
Block graph + nbr. edges	2.63 ± 0.17	1.70 ± 0.05
Algorithm	Game	Winner
Naive	75.70 ± 1.43	30.79 ± 1.53
Stern et al. (2004)	38.30 ± 1.62	13.80 ± 1.15
Block graph	13.02 ± 1.12	4.53 ± 0.69
Block graph + nbr. edges	9.05 ± 0.95	2.87 ± 0.55

References

Enzenberger, M. (1996). The integration of a priori knowledge into a Go playing neural network. Available at http://www.cs.ualberta.ca/ emarkus/neurogo/.

Globerson, A., & Jaakkola, T. (2007). Approximate inference using planar graph decomposition. In *NIPS*.

Graepel, T., Goutrié, M., Krüger, M., & Herbrich, R. (2001). Learning on graphs in the game of Go. In *ICANN*.

Stern, D. H., Graepel, T., & MacKay, D. J. C. (2004). Modelling uncertainty in the game of Go. In *NIPS*.

van der Werf, E. C. D., van den Herik, H. J., & Uiterwijk, J. W. H. M. (2005). Learning to score final positions in the game of Go. *Theoretical Computer Science*, *349*(2).

Vilà, R., & Cazenave, T. (2003). When one eye is sufficient: A static classification. In *Advances in Computer Games*.