Modelling Go Positions with Planar CRFs

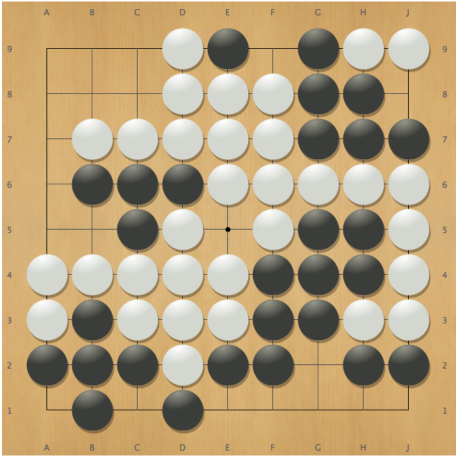
Dmitry Kamenetsky, Nicol N. Schraudolph,
Simon Günter, S.V. N. Vishwanathan

NICTA, Australian National University, Australia

University of Alberta, December 2007

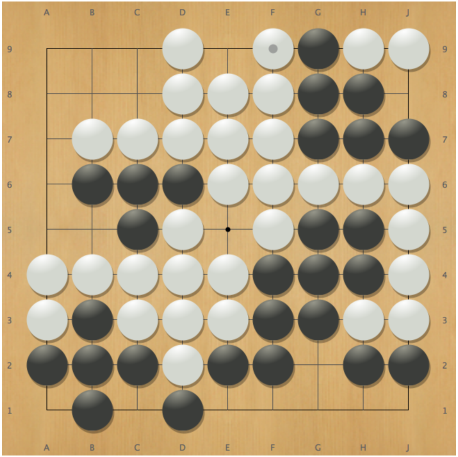
1. Background
   - Go
   - Learning in Go
   - Ising model
   - Dimer problem
2. Our work
   - Algorithm
   - Graph abstraction
   - Features and parameters
   - Results
3. Conclusion

What is Go?



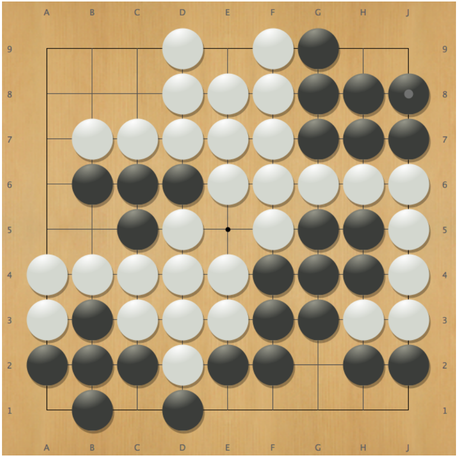
- Two players alternate in placing stones on the intersections of a grid
- Neighbouring stones of the same colour form a contiguous *block*
- A block can be *captured* if all its empty neighbours are occupied by opponent stones

What is Go?



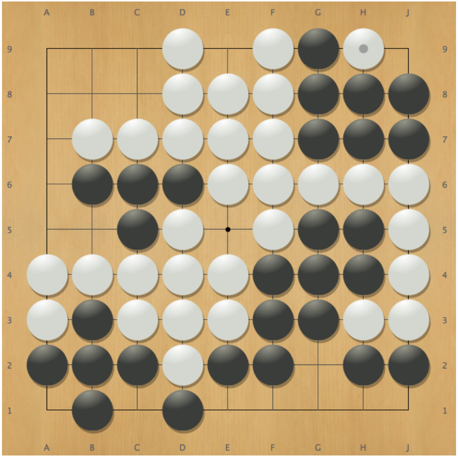
- Two players alternate in placing stones on the intersections of a grid
- Neighbouring stones of the same colour form a contiguous *block*
- A block can be *captured* if all its empty neighbours are occupied by opponent stones

What is Go?



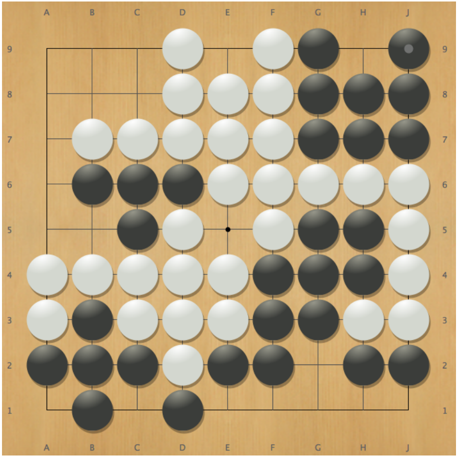
- Two players alternate in placing stones on the intersections of a grid
- Neighbouring stones of the same colour form a contiguous *block*
- A block can be *captured* if all its empty neighbours are occupied by opponent stones

What is Go?



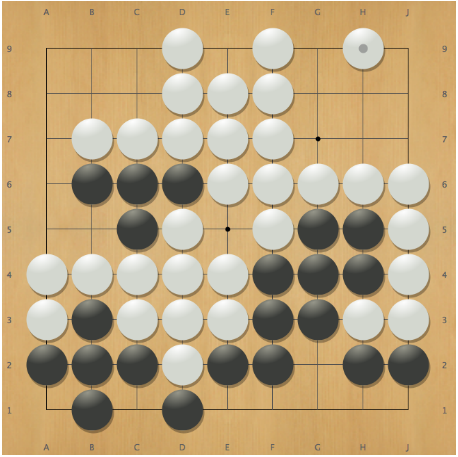
- Two players alternate in placing stones on the intersections of a grid
- Neighbouring stones of the same colour form a contiguous *block*
- A block can be *captured* if all its empty neighbours are occupied by opponent stones

What is Go?



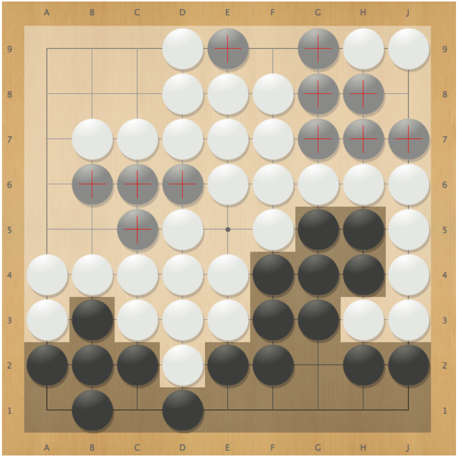
- Two players alternate in placing stones on the intersections of a grid
- Neighbouring stones of the same colour form a contiguous *block*
- A block can be *captured* if all its empty neighbours are occupied by opponent stones

What is Go?



- Two players alternate in placing stones on the intersections of a grid
- Neighbouring stones of the same colour form a contiguous *block*
- A block can be *captured* if all its empty neighbours are occupied by opponent stones

What is Go?



- The game terminates once players agree on the life status of blocks
- The blocks and their surrounding area count towards *territory*
- **Territory prediction:** Given a board position predict the owner of each intersection
- Challenging problem for ML!

Learning in Go

- Go is played on a grid graph G, so it is natural to model it with a graphical model such as CRF
- If we want to perform exact inference we can use the Junction Tree Algorithm (G is loopy)

Junction Tree Algorithm

- State-of-the-art exact method for computing partition function, marginals and MAP state
- Graph is **a tree**: complexity polynomial in graph size
- Graph is **not a tree**:
  - Convert the graph into a tree of cliques
  - Complexity exponential in the treewidth = size of the maximal clique
  - For $N \times N$ grid the treewidth is N

What can we do?

- It turns out that physicists working on the Ising model have found an answer back in the 1960's!
- The method has been introduced to the Graphical Models community only last year...

Ising model

- Particles modify their behaviour to conform with their neighbours
- What is the mean energy?
- Used in chemistry, physics, biology...
- More than 12,000 papers published!

Ising problem

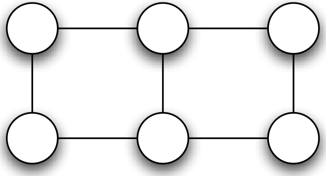
- Graph $G = (V, E)$
- Binary variables: $x_i \in \{-1, +1\}$
- No potential for disagreement edges: $\phi_{ij} = 0$ if $x_i \neq x_j$
- Model distribution

$$P(x) = \frac{1}{Z(\phi)} e^{\sum_{ij \in E} [x_i = x_j] \phi_{ij}} \text{ , where}$$

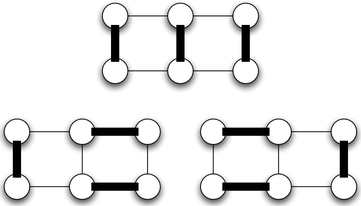
$$Z(\phi) = \sum_x e^{\sum_{ij \in E} [x_i = x_j] \phi_{ij}} \text{ is the partition function}$$

Dimer problem

- How many perfect matchings does a graph have?

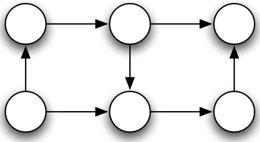


- **Perfect Matching**: A set of non-overlaping edges (dimers) that cover all vertices



Counting Matchings

- Every planar graph has a **Pfaffian orientation**: each face (except possibly outer) has an odd number of edges oriented clockwise



- Define a skew-symmetric matrix K such that:

$$K_{ij} = \begin{cases} 1 & \text{if } i \rightarrow j \\ -1 & \text{if } i \leftarrow j \\ 0 & \text{otherwise} \end{cases}$$

Kasteleyn Theorem

$$K = \begin{array}{|c|c|c|c|c|c|} \hline 0 & 1 & 0 & 0 & 0 & -1 \\ \hline -1 & 0 & 1 & 0 & 1 & 0 \\ \hline 0 & -1 & 0 & -1 & 0 & 0 \\ \hline 0 & 0 & 1 & 0 & -1 & 0 \\ \hline 0 & -1 & 0 & 1 & 0 & -1 \\ \hline 1 & 0 & 0 & 0 & 1 & 0 \\ \hline \end{array}$$

Kasteleyn Theorem:

Number of perfect matchings is $\mathrm{Pf}(K) = \sqrt{|K|}$

The connection

- Let $G_\triangle$ be G plane triangulated: each face becomes a triangle
- Let G^* be the dual of graph $G_\triangle$: each face in $G_\triangle$ is a vertex in G^*
- Let G^*_e be the expanded version of G^*: each vertex is replaced with 3 vertices in triangle
- **Connection**: There is a 1:1 correspondence between agreement edge sets in G and perfect matchings in G^*_e

From physics to ML

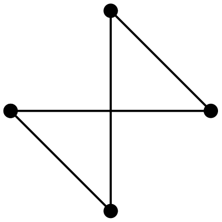
- Globerson & Jaakkola 2006 use previous results (Ising model) to compute the partition function exactly
- Restrictions:
    - Graph is planar: can be drawn without crossing edges
    - Binary-valued labels
    - Only edge potentials, no node potentials
- Complexity **polynomial** in graph size!

Overview

- Faster and simpler version of Globerson and Jaakkola algorithm
- No need to compute the dual G^* and expanded version G_e^*
- Showed how to compute gradients and thus perform parameter estimation
- Applied to territory prediction in Go

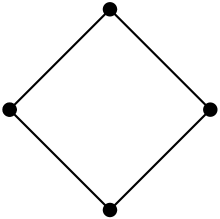
Algorithm

- Original graph $G = (V, E)$



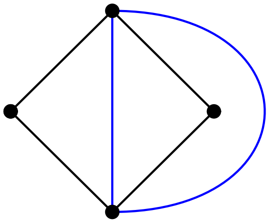
Algorithm: Step 1

- Obtain a planar embedding
- Using Boyer-Myrvold algorithm the complexity is $O(n)$, where $n = |E|$



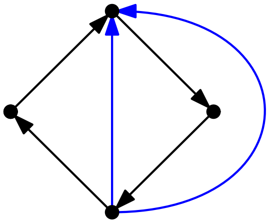
Algorithm: Step 2

- Add edges to plane triangulate the graph
- Using simple ear-clipping the complexity is $O(n)$



Algorithm: Step 3

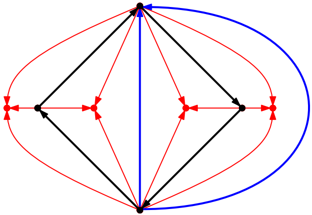
- Orient the edges such that each vertex has odd in-degree
- Equivalent to having a Pfaffian orientation in the dual graph
- Complexity is $O(n)$

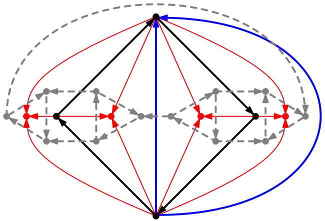


Algorithm: Step 4 (intuition)

- Add nodes to each face
- Orient edges towards those nodes
- Equivalent to expansion in the dual graph

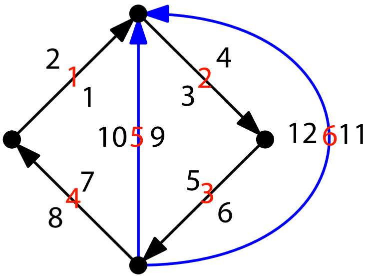
- Construct a skew-symmetric $2|E| \times 2|E|$ matrix K (for dual edges):
  - $K_{ij} = \pm e^{\phi_{ij}}$ if ij crosses **original**
  - $K_{ij} = \pm 1$ if ij crosses added
- Complexity is $O(n)$





Algorithm: Step 4 (implementation)

- Number each edge
- **Number** the sides of each edge k
  - LHS $= 2k$
  - RHS $= 2k - 1$



Pseudo Code

For each vertex v:

- For each edge k incident on v (clockwise):
  - if k points away from v:
    - $K_{2k,p} = 1$ $(2 \to 8)$
    - $p = 2k - 1$
  - else
    - $K_{2k-1,p} = 1$ $(7 \to 1)$
    - $K_{2k-1,2k} = e^{\phi_k}$ $(7 \to 8)$
    - $p = 2k$

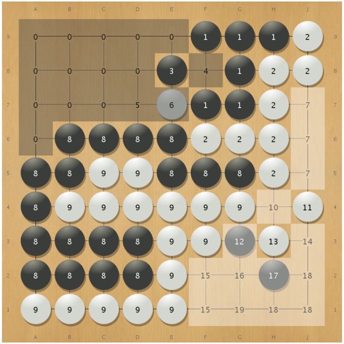
Return $K - K^T$

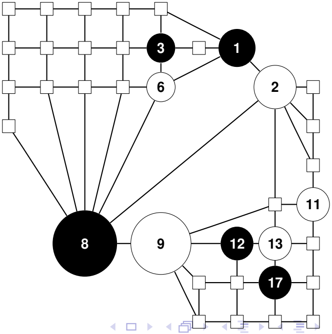
Algorithm: Parameter estimation

- Compute partition function: $Z(\phi) = 2\sqrt{|K|}$
- Compute gradients: $\frac{\partial \ln Z(\phi)}{\partial \phi_k} = -[K^{-1} \odot K]_{2k-1,2k}$
- Computing inverse and determinant takes at most $O(n^3)$ time

Graph abstraction: common fate graph

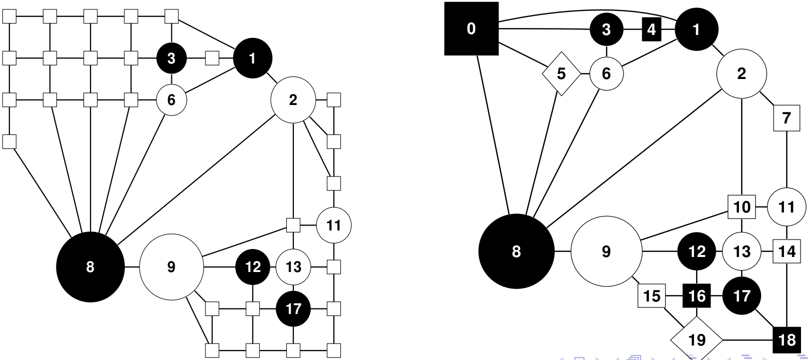
- Blocks always live or die as a unit; Grid graph G does not capture this
- *Common fate graph* G_f (Graepel et al., 2001) merges all stones in a block into a single node





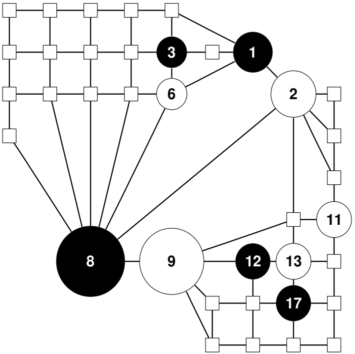
Graph abstraction: block graph

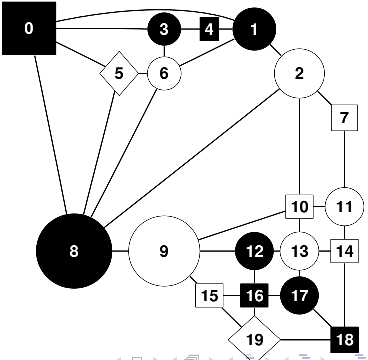
- Use Manhattan distance to classify empty regions into 3 types: *black surround* (■), *neutral*(◇) and *white surround*(□)
- Collapse empty regions to form the *block graph* G_b



Graph abstraction: block graph

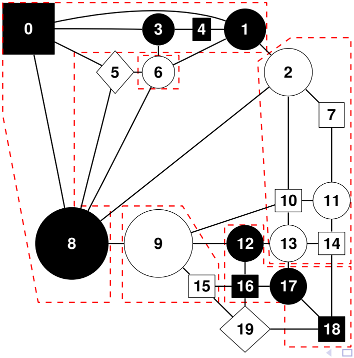
- Surrounds encode the possibility for obtaining territory
- G_b is more concise than G_f, but preserves the kind of information required for predicting territory





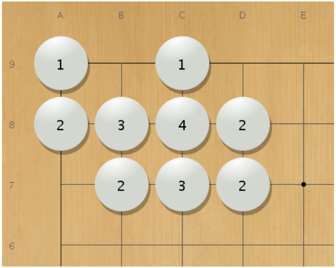
Graph abstraction: group graph

- *Group*: set of blocks of the same colour that share at least one surround
- Construct the *group graph* G_g by collapsing groups of G_b



Feature engineering: nodes

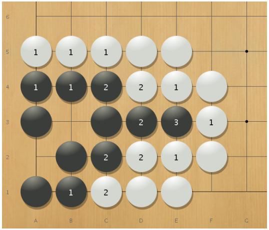
- Given a node $v \in G_b$, for each point $i \in v$ compute the number of adjacent points A_i that are also in v
- Node's feature is a vector F, where $F_k = |\{i : A_i = k\}|$
- Provides a powerful summary of the region's shape



$$F = \{2, 4, 2, 1\}$$

Feature engineering: edges

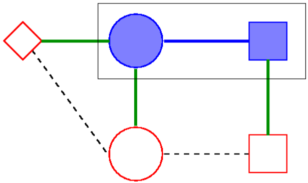
- For two nodes $v_1, v_2 \in G_b$, A_i^1 is the number of points in v_2 that are adjacent to $i \in v_1$ and vice-versa for A_i^2
- Edge's features are two vectors F^1 and F^2 that are constructed using A^1 and A^2 respectively
- Provide information of node's liberties and boundary shape



$F^1 = \{3, 3, 1\}, F^2 = \{6, 3, 0\}$

Parameter sharing

Parameter sharing takes into account all relevant symmetries



	Current Edge		Neighbour Edges	
	Param.	Feat.	Param.	Feat.
Nodes	$\vec{\theta}_{\bigcirc}$	●	$\vec{\theta}^{n}_{\diamond}$	◇
	$\vec{\theta}_{\square}$	■	$\vec{\theta}^{n}_{\bigcirc}$	○
			$\vec{\theta}^{n}_{\square}$	□
Edges	$\vec{\theta}_{\bigcirc\square}$	● $\longrightarrow$ ■	$\vec{\theta}^{n}_{\diamond\bigcirc}$	◇ $\longrightarrow$ ●
	$\vec{\theta}_{\square\bigcirc}$	■ $\longrightarrow$ ●	$\vec{\theta}^{n}_{\bigcirc\diamond}$	● $\longrightarrow$ ◇
			$\vec{\theta}^{n}_{\bigcirc\bigcirc}$	○ $\longrightarrow$ ● ● $\longrightarrow$ ○
			$\vec{\theta}^{n}_{\square\square}$	□ $\longrightarrow$ ■ ■ $\longrightarrow$ □

Experiments: Learning

- 9×9 endgame positions of van der Werf et al., 2005
- 1000-2000 games
- Use the block graph G_b
- Optimization with LBFGS

Experiments: Prediction

- 906 games
- Currently compute MAP state using variable elimination (exponential)
- Can be done in **polynomial time** with min-weight perfect matching!
- Can also use marginals from each node
- **Problem:** Computed labeling is for edges, not nodes
- Use the group graph G_g

Results

	Error (%)			
Algorithm	Vertex	Block	Winner	Game
Naive	6.79	17.57	30.79	75.70
Stern et al., 2004	4.77	7.36	13.80	38.30
Block graph	2.36	3.56	4.53	13.02
Block graph + neighbour features	1.87	2.76	3.42	9.60
Block graph + other enhancements	1.54	2.20	2.09	7.90
* GnuGo	-	-	-	1.32
* van der Werf et al., 2005	0.19	$\leq$ 1.00	0.50	1.10

*: employs Go-specific features and was used to label data

Conclusion

- Algorithm improvements:
  - No need to compute the dual nor the expanded graph
  - Compute gradients and hence perform parameter estimation
- Model novelty:
  - 2-stage graph reduction of the Go positions. The first used for learning, the later for prediction
  - Generic node and edge features. Parameter sharing between equivalent node and edge types

Future work

- Find better ways to classify empty regions
- Add more domain-specific knowledge
- Extend to 19×19 games
- Compute MAP state using min-weight perfect matching

Extensions

- Middle-game positions, move prediction
- Incorporate into a Monte-Carlo based program:
  - Goanna (with Joel Veness)
  - UCT-based, 2250 on 9×9 CGOS, 5th
  - Can be used for random playouts and prior knowledge

Questions?

"The more you let yourself GO, the less others let you GO" - Friedrich Nietzsche