Efficient exact inference in Ising graphical models
applied to Go

Dmitry Kamenetsky
Supervisor: Nicol N. Schraudolph

NICTA, Australian National University, Australia

Midterm, May 2008
**Background**

Game of Go
Computer Go
Graphical model for Go
Ising graphical model

**Our method**

Algorithm

**Experiments**

Graph abstraction
Features and parameters
Prediction Accuracy
Prediction speed

**Work Progress**
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 (liberties) 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 is used to determine the winner of the game
Why is Go challenging for computers?

- Huge branching factor
  - $\approx 200$ on $19 \times 19$ board, $\approx 40$ in Chess
  - $\approx 10^{172}$ legal board positions, $\approx 10^{50}$ in Chess
  - Standard alpha-beta min-max is too inefficient

- Position evaluation is difficult
  - Hard to judge strength of blocks statically
  - Stones have both local and long-range interactions
Heuristic-based programs

- Rely on hand-tuned patterns and results from local searches

**Advantages:** Strong locally, especially if pattern is known

**Disadvantages:**
- Weak at global play
- Weak at judging unseen situations
- Board evaluation is slow
Learning-based programs

- Learn an evaluation function using self-play or expert games

**Advantages:** Loads of expert games available

**Disadvantages:** Relatively weak playing strength

**Gut feeling:**
- State-space is too large
- Hard to define features
- Evaluation function is highly non-smooth
Sampling-based programs

- Bandit-based tree search (UCT)
  - Each tree node (board position) is a multi-arm bandit
  - Sample child positions, maximize total reward
  - Store all node statistics in a tree data structure
  - If a node is not in the tree then use an evaluation function

- Evaluation function
  - Playout position randomly until no moves remain. Final position is trivial to score
  - Enhanced through the use of patterns and other heuristics
Sampling-based programs

- **Advantages:**
  - Evaluation function is fast and accurate for many samples
  - Increase in samples gives increase in playing strength
  - Assymetric tree growth - more time spent on difficult positions
  - Best performance. Reached 3-dan (professional) level on 9 × 9

- **Disadvantages:**
  - Weak performance early in the game
  - Still weak on larger boards

- **Conclusion:**
  - Framework has good potential
  - But need to improve both search and evaluation
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

- Major problem is inference:
  - Approximate: Loopy Belief Propagation
  - Exact: Junction-Tree, Graph Cuts
**Junction Tree Algorithm**

- Exact method for computing partition function, marginals and MAP (maximum a posteriori) 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 $N$
Graph Cuts

- Exact method for computing MAP state of a binary-labeled problem
- Treat MAP computation as finding the min-cut of a particular graph (with positive edge weights)
- **Theorem:** finding the graph’s min-cut is equivalent to finding its max-flow
- Can use Ford-Fulkerson. Complexity is polynomial in graph size
Other methods?

- **Question:** Is there a method that can compute partition function and marginals like Junction Tree, but in polynomial time like Graph Cuts?

- **Answer:** Yes!
Other methods?

• **Question:** Is there a method that can compute partition function and marginals like Junction Tree, but in polynomial time like Graph Cuts?

• **Answer:** Yes!
**Ising problem**

- Graph $G = (V, E)$, binary variables (spins): $y_i \in \{+,-\}$

- Spins only interact in pairs. One energy for agreement: $\psi_{--} = \psi_{++}$, another for disagreement: $\psi_{-+} = \psi_{+-} = 0$

- Model distribution:

  $$P(y) = \frac{1}{Z(\psi)} \exp\left(\sum_{ij \in E} [y_i = y_j]\psi_{ij}\right)$$

  where

  $$Z(\psi) = \sum_y \exp\left(\sum_{ij \in E} [y_i = y_j]\psi_{ij}\right)$$

  is the partition function
Dimer problem

- How many perfect matchings does a graph have?

- **Perfect Matching**: A set of non-overlapping 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

\[
K_{ij} = \begin{cases} 
1 & \text{if } i \to j \\
-1 & \text{if } i \leftarrow j \\
0 & \text{otherwise}
\end{cases}
\]
**Kasteleyn Theorem**

The Kasteleyn Theorem states that the number of perfect matchings in a graph can be computed by finding the determinant of a matrix constructed from the graph's edges.

Given the matrix $K$:

\[
K = \begin{pmatrix}
0 & 1 & 0 & 0 & 0 & -1 \\
-1 & 0 & 1 & 0 & 1 & 0 \\
0 & -1 & 0 & -1 & 0 & 0 \\
0 & 0 & 1 & 0 & -1 & 0 \\
0 & -1 & 0 & 1 & 0 & -1 \\
1 & 0 & 0 & 0 & 1 & 0 \\
\end{pmatrix}
\]

**Kasteleyn Theorem:**

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

- Let $G_\Delta$ be $G$ plane triangulated: each face becomes a triangle
- Let $G^*$ be the dual of graph $G_\Delta$: each face in $G_\Delta$ 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 perfect matchings in $G^*_e$ and agreement edge sets in $G$
Our method: overview

- No need to compute the dual $G^*$ and expanded dual $G_e^*$
- Show how to compute the marginals and hence perform parameter estimation
- Show how to compute the MAP state
- All computations are polynomial in graph size
Our method: overview

- Model distribution:

\[
P(y) = \frac{1}{Z(\psi)} \exp(\sum_{ij \in E} [y_i = y_j] \psi_{ij})
\]

\[
P(y|x; \theta) = \frac{1}{Z(x; \theta)} \exp(\sum_{ij \in E} [y_i = y_j] <\phi_{ij}(x), \theta>)
\]

- Restrictions:
  - Graph is planar: can be drawn without crossing edges
  - Binary labels
  - No node potentials (no external field)
  - Edge potentials: one for agreement, one for disagreement
Comparison to Graph Cuts

<table>
<thead>
<tr>
<th></th>
<th>Graph Cuts</th>
<th>Ising Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Need planarity?</td>
<td>No</td>
<td>For polynomial runtime</td>
</tr>
<tr>
<td>2-label problem</td>
<td>Exact and polynomial runtime</td>
<td></td>
</tr>
<tr>
<td>N-label problem</td>
<td>approx. with $\alpha$-expansion</td>
<td>Not yet</td>
</tr>
<tr>
<td>Node potentials?</td>
<td>Yes</td>
<td>Only outerplanar graph</td>
</tr>
<tr>
<td>Energy restriction</td>
<td>Submodularity: $E_{0,0} + E_{1,1} \leq E_{0,1} + E_{1,0}$</td>
<td>$E_0 = E_1 = 0$</td>
</tr>
<tr>
<td></td>
<td>non-submodular: partial sol.</td>
<td>$E_{0,1} = E_{1,0}$ ($= 0$)</td>
</tr>
<tr>
<td></td>
<td>$E_{0,0} = E_{1,1}$</td>
<td>$E_{0,0} = E_{1,1}$</td>
</tr>
<tr>
<td>Combined restriction</td>
<td>$E_{0,0} = E_{1,1} \leq 0, E_0 = E_1 = 0$: trivial solution</td>
<td></td>
</tr>
<tr>
<td>Partition function?</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Marginals?</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Parameter Estimation</td>
<td>Max-Margin</td>
<td>Max-Likelihood</td>
</tr>
</tbody>
</table>
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):
  - $K_{ij} = \pm e^{\psi_{ij}}$ 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 $v$:
    - $K_{2k, \text{prev}} = 1$ ($2 \rightarrow 8$)
  - else
    - $K_{2k-1, \text{prev}} = 1$ ($7 \rightarrow 1$)
    - $K_{2k-1,2k} = e^{\psi_k}$ ($7 \rightarrow 8$)

Return $K - K^\top$
Algorithm: Parameter estimation

- Compute partition function: \( Z(\psi) = 2 \sqrt{|K|} \)
- Compute gradients:

\[
\frac{\partial \ln Z(\psi)}{\partial \theta_k} = \frac{2}{Z(\psi)} \frac{\partial \sqrt{|K|}}{\partial \psi_k} = \ldots = -[K^{-1} \odot K]_{2k-1,2k}
\]
- Computing inverse and determinant takes at most \( O(n^3) \) time
**Algorithm: MAP state**

- Maximum *a posteriori* state (MAP):

\[ y^* = \arg\max_y \mathbb{P}(y|x; \theta^*) , \text{ where} \]

\[ \theta^* = \arg\min_\theta \mathcal{L}(\theta) , \quad \mathcal{L}(\theta) = \frac{||\theta||^2}{2\sigma^2} - \sum_{k=1}^{m} \ln \mathbb{P}(y|x; \theta) \]

- Max-weight perfect matching on \( G_e^* \) gives the max-weight agreement edge set. Use blossom-shrinking (Edmonds 1965)

- This takes \( O(n^2 \log(n)) \) time
Algorithm: Numerical problems

- Computation of $|K|$ and $K^{-1}$ are prone to numerical problems
- Method 1: for skew-symmetric matrices $K$ and constant $q$:
  \[ |K| = \frac{|qK|}{q^n} \]
- Method 2: use LU decomposition of $K$:
  \[ K^{-1} = U^{-1}L^{-1}, \quad \ln |K| = \sum_{i=1}^{n} \ln U_{i,i} \]
**Territory prediction in Go**

- 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!
Graph abstraction: common fate graph

- Grid graph $G$ does not capture the fact that stones in a block always live or die as a unit
- Common fate graph $G_{cfg}$ (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 $G_{cfg}$, 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 node $v \in G_b$, compute feature vector $F$:

  $$F_k = \text{num. intersections in } v \text{ with } k \text{ neighbours in } v$$

- Provides a powerful summary of the region’s shape

$$F = \{2, 4, 2, 1\}$$
**Feature engineering: edges**

- Given nodes $v^1, v^2 \in G_b$, compute their corresponding features $F^1$ and $F^2$:

  \[
  F^1_k = \text{num. intersections in } v^1 \text{ with } k \text{ neighbours in } v^2 \\
  F^2_k = \text{num. intersections in } v^2 \text{ with } k \text{ neighbours in } v^1
  \]

- Provide information of node’s liberties and boundary shape

\[
F^1 = \{3, 3, 1\}, \quad F^2 = \{6, 3, 0\}
\]
Datasets

- 9 × 9 games: Van der Werf et al. collection, 1000 training and 906 testing
- 19 × 19 games: scored by our cooperative scorer, 1000 for training and testing
- Oversize games: 22 games manually scored. Sizes range from 21 × 21 to 38 × 38. Only for testing
Task

- Given an endgame position determine the label *(black or white)* of each intersection
- We train our CRF on the block graph $G_b$, using BFGS as the optimizer
- Prediction determined using MAP state of the group graph $G_g$
Controls

- **Naive**: assume all stones are alive

- **GnuGo 3.6**: open source program. Uses Go-specific knowledge and local searches

- **NN**: neural net classifier (van der Werf et al. 2005). Uses 63 Go-specific features of various board abstractions

- **MRF**: a simple MRF on G with just 6 parameters (Stern et al. 2004). Inference via 50 iterations of LBP. Prediction via marginal expectations at each intersection
## Prediction Accuracy

<table>
<thead>
<tr>
<th>Size</th>
<th>Algorithm</th>
<th>Error (%)</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Block</td>
<td>Stone</td>
<td>Game</td>
</tr>
<tr>
<td>9 × 9</td>
<td>Naive</td>
<td>17.57</td>
<td>8.80</td>
<td>75.70</td>
</tr>
<tr>
<td></td>
<td>MRF</td>
<td>8.19</td>
<td>5.97</td>
<td>38.41</td>
</tr>
<tr>
<td></td>
<td>CRF approx</td>
<td>2.73</td>
<td>2.46</td>
<td>9.93</td>
</tr>
<tr>
<td></td>
<td>CRF exact</td>
<td>2.57</td>
<td>2.32</td>
<td>9.05</td>
</tr>
<tr>
<td></td>
<td>GnuGo*</td>
<td>-</td>
<td>0.05</td>
<td>1.32</td>
</tr>
<tr>
<td></td>
<td>NN*</td>
<td>≤ 1.00</td>
<td>0.19</td>
<td>1.10</td>
</tr>
<tr>
<td>19 × 19</td>
<td>Naive</td>
<td>16.52</td>
<td>6.96</td>
<td>98.30</td>
</tr>
<tr>
<td></td>
<td>MRF</td>
<td>4.91</td>
<td>3.80</td>
<td>63.90</td>
</tr>
<tr>
<td></td>
<td>CRF approx</td>
<td>5.25</td>
<td>4.93</td>
<td>49.00</td>
</tr>
<tr>
<td></td>
<td>CRF exact</td>
<td>3.93</td>
<td>3.81</td>
<td>43.40</td>
</tr>
<tr>
<td></td>
<td>GnuGo</td>
<td>-</td>
<td>0.11</td>
<td>5.10</td>
</tr>
<tr>
<td>greater</td>
<td>Naive</td>
<td>19.64</td>
<td>10.25</td>
<td>100.00</td>
</tr>
<tr>
<td>than 19×19</td>
<td>MRF</td>
<td>7.80</td>
<td>6.83</td>
<td>100.00</td>
</tr>
<tr>
<td></td>
<td>CRF approx</td>
<td>7.51</td>
<td>6.84</td>
<td>81.82</td>
</tr>
<tr>
<td></td>
<td>CRF exact</td>
<td>4.52</td>
<td>5.02</td>
<td>81.82</td>
</tr>
</tbody>
</table>

* Was used to label data
Errors made by different methods

- ○ misclassified by Naive, MRF and CRF
- □ by Naive and MRF
- △ by Naive only
- Gnugo made no errors
- MRF inconsistent due to use of marginals
CRF’s perfect prediction for an oversize game
Prediction speed: methods

- **GnuGo 3.6**: scoring in *aftermath* mode
- **LBP**: 50 iterations of LBP for marginal expectations (Stern et al. 2004)
- **Brute force**: variable elimination with arbitrary elimination ordering
- **Variable elimination**: using min-fill heuristic (Kjaerulff 1990)
- **Our method**: blossom-shrinking (Edmonds 1965)
Prediction Speed
This work

- Not much luck with conference papers
- Going to publish this as a journal paper in JMLR
- Want to try territory prediction for middle-game positions and move prediction
- Want to apply this method to images and compare directly to Graph Cuts
Bandit-based tree search

- Assume each node $n_i$ has a reward distribution $X_i$
- UCT samples node

$$n^*_i = \arg\max_{n_i} (\mathbb{E}(X_i) + c \cdot \text{Var}(X_i))$$
- Instead assume $X_i = B(\alpha_i, \beta_i)$. Now sample node

$$n^*_i = \arg\max_{n_i} (x_i \sim X_i)$$
- Performance not as good as UCT’s
- Now want to try Gittins indices
Bandit-based tree search

- UCT falsely assumes that arms (siblings) are independent
- Instead sample from dependant arms (Pandey et al., 2007)
  - Cluster arms (eg. based on group graph)
  - Step 1: Select a cluster to sample
  - Step 2: Select an arm within that cluster to sample
  - Update statistics of all arms in that cluster
- Can expect huge speed-up
Other Go work

- Cooperative Scorer
- Fast influence function
- Go playout on GPU
  - GPUs are designed for floating point and matrix operations
  - Nvidia Tesla has up to 128 parallel cores, 512 Gflops
  - Developed a random Go player that only uses matrix operations
  - Huge potential if works on the GPU!
Questions?