# Ising Graphical Model 

## Dmitry Kamenetsky

A thesis submitted for the degree of
Doctor of Philosophy at The Australian National University
(c) Dmitry Kamenetsky

Typeset in Palatino by $\mathrm{T}_{\mathrm{E}} \mathrm{X}$ and $\mathrm{LT}_{\mathrm{E}} \mathrm{X} 2 \varepsilon$.

Except where otherwise indicated, this thesis is my own original work.

Dmitry Kamenetsky
18 May 2010

## Acknowledgements

It is the end of a long and difficult journey and I would like to thank everyone who has made it possible. A big thank you must go to all my supervisors: Nic Schraudolph, Scott Sanner, Tiberio Caetano and Gideon Dror. Nic you have inspired me with your skill and knowledge. You have taught me well and now I feel ready to tackle any research challenge. Scott and Tiberio, thank you for your support and guidance. Gideon thank you for your kindness and many great memories.

I also want to thank all the past and present staff of our Statistical Machine Learning group who have made me appreciate both the power and beauty of Machine Learning: Douglas Aberdeen, Marconi Barbosa, Olivier Buffet, Wray Buntine, Li Cheng, Kishor Gawande, Simon Günter, Markus Hegland, Marcus Hutter, Adam Kowalcyzk, Kee Siong Ng, Petteri Nurmi, Brian Parker, Mark Reid, Sylvia Richter, Conrad Sanderson, Alex Smola, Peter Sunehag, S. V. N. Vishwanathan, Christfried Webers and Markus Weimar.

A huge thank you to all my fellow students: Justin, Robby, Julian, Joel, James, Novi, Javen, Le, Choon Hui, Owen, Jin and Xinhua. You have made this journey fun and bearable. You have all been great friends and I hope to stay in touch no matter where life takes us. For those who haven't finished, remember that I feel your pain and that the end is near. Justin you have been a good friend and taught me how to appreciate nice tea. Robby I can't wait to have some fine Belgian beer with you and discuss the aesthetic value of skimpy bikinis. Julian, Le and Xinhua thank you for helping me out on so many occassions, I hope I wasn't too much of a pest. Joel it has been a pleasure working with you and I think we made a great team - one day we will make the best Go program. James, I guess I am now a doubly-finished man, which reminds me that we must finish our theory on the meaning of life. Novi thank you for making me laugh and putting a smile on my face. Javen I still remember our fun roadtrip and dancing lessons, we should do these again someday. Choon Hui don't give up on TopCoder and remember to never underestimate yourself. Owen you must teach me how to make such great photos and I really miss playing poker with you. Jin I am looking forward to going camping and fishing with you; good luck in your new position.

I would like to thank all my friends for distracting me from work and helping me to relax, in particular: Chris, Simon, Desmond, Ayman, Alexey, Linda, Caroline, Lairi, Rowan. Chris thank you for teaching me wonderful things about matrices, helping me with primes and for being a fatherly figure to me. Simon thank you for being such a great friend both at work and outside. I've missed your jokes and our never-ending debates. Desmond thank you for introducing me to skiing, I promise to get better. Ayman you have opened my mind and taught me many things about life, so I say thank you. Alexey I greatly enjoyed everything we did together and I am looking forward to many more good times ahead. Linda thank you for your support and encouragement. Caroline thank you for the fun times and your great help in job hunting. Lairi I enjoyed our tennis games and friendship. Rowan I will greatly miss your hot chocolates, our crazy dancing, our conversations about nothing, watching Tour de France and
infomercials at 2am, and playing board games. You have been the best man in every sense of that word. Stay strong and your freedom will come soon. Thank you also to all my inter-state friends who couldn't be here, but always supported me morally.

I would like to thank all the staff in the Deparment of Computing Science at the University of Alberta for their warm welcome, despite the cold weather. In particular, I thank Dale Schuurmans and David Silver for organizing my visit.

I thank NICTA and the Australian National University for financing my studies and giving me a nice office. I also thank Michelle Moravec, Deb Pioch, Diane Kossatz and Sue Van Haeften for keeping me on track and helping me with all the complex forms. My thank you goes to our Turkish corner shop for cooking tasty pizzas that helped me to get more work done.

Finally I would like to thank the people who are closest to me - my family. Without your continuous support, encouragement and words of wisdom I wouldn't get very far. My parents, you have always believed in my abilities and have helped me in difficult times. Last, but not least I would like to thank my dear wife Kate. Thank you for your patience and unconditional love. Your delicious dinners have given me the strength to continue and your love has given me complete happiness.

## Abstract

The Ising model is an important model in statistical physics, with over 10,000 papers published on the topic. This model assumes binary variables and only local pairwise interactions between neighbouring nodes. Inference for the general Ising model is NP-hard; this includes tasks such as calculating the partition function, finding a lowest-energy (ground) state and computing marginal probabilities.

Past approaches have proceeded by working with classes of tractable Ising models, such as Ising models defined on a planar graph. For such models, the partition function and ground state can be computed exactly in polynomial time by establishing a correspondence with perfect matchings in a related graph.

In this thesis we continue this line of research. In particular we simplify previous inference algorithms for the planar Ising model. The key to our construction is the complementary correspondence between graph cuts of the model graph and perfect matchings of its expanded dual. We show that our exact algorithms are effective and efficient on a number of real-world machine learning problems.

We also investigate heuristic methods for approximating ground states of non-planar Ising models. We show that in this setting our approximative algorithms are superior than current state-of-the-art methods.

## Contents

Acknowledgements ..... v
Abstract ..... vii
1 Introduction ..... 1
1.1 Thesis Contribution ..... 1
1.2 Thesis Outline ..... 2
2 Background ..... 3
2.1 Mathematical Notation ..... 3
2.2 Graph Notation ..... 3
2.2.1 Connectivity and Biconnectivity ..... 4
2.2.2 Planar Graphs ..... 4
2.3 Dimer Problem ..... 7
2.3.1 Counting Dimer Coverings ..... 8
2.3.2 Pfaffian ..... 9
2.3.3 Constructing the Kasteleyn Matrix ..... 11
2.3.4 Pfaffian Graphs ..... 12
2.3.5 Examples ..... 15
3 Review of the Ising Model ..... 17
3.1 Introduction ..... 17
3.1.1 Brief History ..... 19
3.2 Previous Work on the Partition Function of the Ising Model ..... 20
3.2.1 Ising Problem to Even Subgraphs ..... 20
3.2.2 Even Subgraphs to Perfect Matchings ..... 22
3.2.3 Non-Planar Ising Models ..... 25
3.3 Previous Work on the Ground States of the Ising Model ..... 27
3.3.1 Work by Bieche et al ..... 27
3.3.2 Work by Barahona ..... 29
3.3.3 Work by Thomas and Middleton ..... 30
4 Review of Graphical Models ..... 31
4.1 Introduction ..... 31
4.2 Undirected Graphical Models ..... 32
4.2.1 Conditional Independence Property ..... 32

## Contents

4.2.2 Factorization Property ..... 33
4.3 Inference Algorithms ..... 34
4.3.1 Variable Elimination ..... 34
4.3.2 Belief Propagation ..... 35
4.3.3 Loopy Belief Propagation ..... 36
4.3.4 Junction Tree Algorithm ..... 37
4.3.5 Graph Cuts ..... 38
4.3.6 Tree Reweighting ..... 39
4.3.7 Work by Globerson and Jaakkola ..... 41
5 Planar Ising Graphical Models: Inference ..... 43
5.1 Introduction ..... 44
5.1.1 Energy Minimization via Cuts ..... 46
5.1.2 Planarity Constraint ..... 47
5.1.3 Connectivity ..... 48
5.1.4 Biconnectivity ..... 48
5.2 Computing Ground States ..... 51
5.2.1 Expanded Dual Graph ..... 52
5.2.2 Complementary Perfect Matchings ..... 53
5.2.3 Computing the Ground State ..... 56
5.3 Computing the Partition Function and Marginal Probabilities ..... 56
5.3.1 Plane Triangulation ..... 57
5.3.2 Odd Edge Orientation ..... 59
5.3.3 Constructing the Kasteleyn Matrix ..... 60
5.3.4 Factoring Kasteleyn Matrices ..... 62
5.4 Conclusion ..... 66
6 Planar Ising Graphical Models: Experiments ..... 67
6.1 Parameter Estimation ..... 67
6.1.1 Maximum Likelihood ..... 67
6.1.2 Maximum Margin ..... 68
6.2 Synthetic Binary Image Denoising ..... 69
6.2.1 Noise Level ..... 71
6.2.2 Parameter Estimation ..... 71
6.2.3 Reconstruction ..... 72
6.3 Boundary Detection ..... 75
6.4 Territory Prediction in Go ..... 77
6.4.1 The Game of Go in Machine Learning ..... 77
6.4.2 Go Positions as Graph Abstraction Hierarchies ..... 80
6.4.3 Conditional Random Field Model ..... 82
6.4.4 Experiments ..... 86
6.4.5 Outlook and Discussion ..... 89
6.5 Conclusion ..... 90
7 Non-Planar Ising Graphical Models ..... 91
7.1 Edmond's Blossom-Shrinking Algorithm ..... 91
7.1.1 Application to Non-Planar Graphs ..... 92
7.2 Heuristics ..... 93
7.2.1 Low-Genus Embedding ..... 94
7.2.2 Minimax Spanning Tree ..... 94
7.2.3 Node Quenching with Implicit Lookahead ..... 95
7.2.4 Tree Quenching ..... 97
7.3 Experiments ..... 99
7.3.1 Experimental Setup ..... 99
7.3.2 Comparison with the Ground State ..... 100
7.3.3 Overall Comparison ..... 104
7.4 Conclusion ..... 108
8 Conclusion ..... 109
8.1 Future Work ..... 110
Bibliography ..... 111

## Introduction

From the beginning of time, man has been trying to understand the world. To help us in this complex task, we have created statistical models that aim to capture all the relevant information of a system. We have models that explain many types of natural phenomena, ranging from gravity and movement of tectonic plates to human behaviour and spread of disease. Models help us to generalise concepts and transfer information between concepts. We can use models to answer many important questions about a system, such as what is its most likely state or what is the probability of a particular event occurring in that system. The procedure used to answer these questions is called statistical inference or simply inference. There are many types of inference, some are suitable for many types of models and questions, while others are specific to certain models and questions. Machine learning is the study of statistical models and inference methods that are best suitable for these models.

One important model in statistical physics is the Ising model. This model assumes binary variables and only local pairwise interactions between neighbouring variables. Despite its simplicity, the Ising model has been successfully used to describe many physical systems, such as ferromagnetism, fluid flow, neural activity, protein folding and many others. Inference in the general Ising model has been shown to be NP-hard [2,54], however it becomes tractable when certain conditions are met.

This thesis investigates tractable inference algorithms for the Ising model and discusses their application to machine learning.

### 1.1 Thesis Contribution

Due to its origins, the Ising model is often described in the language of statistical physics. Here we aim to describe it in the context of machine learning and assume no prior knowledge of statistical physics. The contributions of this thesis are the following:

1. We provide a thorough survey of the Ising model, focussing on the computation of its partition function and ground states.
2. We introduce the Ising graphical model: an undirected graphical model that obeys the constraints of the Ising model, i.e., a binary-labeled energy function that is a sum of edge disagreement costs. We show that the Ising graphical model with an additional bias node is equivalent to any binary-labeled pairwise undirected graphical model.
3. We describe an exact and efficient framework for computing the partition function, worst margin violators, marginal edge probabilities and lowest-energy (ground) states of a planar Ising
graphical model. By doing so, we simplify previous inference algorithms and provide important theoretical connections. In particular, we give the precise complementary connection between graph cuts of the model graph and perfect matchings of its expanded dual. We explain the role of planarity (resp. triangulation) in making that relation a surjection (resp. bijection).
4. We describe how our algorithms can be used in popular parameter estimation frameworks of maximum-margin and maximum-likelihood. We show that our inference algorithms are efficient and effective on a number of real-world machine learning problems.
5. We demonstrate that our algorithms for computing the ground state of a binary-labeled pairwise Markov Random Field outperform current state-of-the-art algorithms.

### 1.2 Thesis Outline

It is recommended that this thesis is read in chronological order, i.e., from Chapter 2 to Chapter 8. Chapters 2, 3 and 4 introduce the necessary background material, while Chapters 5, 6 and 7 describe the contribution of this thesis. The following is a brief outline of the thesis:

- Chapter 2 introduces key definitions and notations used throughout the thesis. The chapter also describes the solution to the Dimer problem, which is necessary for understanding Chapters 3, 5 and 6.
- Chapter 3 describes the Ising model explains previous approaches for the computation of the partition function and ground states in this model.
- Chapter 4 presents a review of graphical models and their most popular inference algorithms.
- Chapter 5 introduces the Ising graphical model, which is a binary-labeled undirected graphical model based on the Ising model. The chapter focusses on planar Ising graphical models and describes a framework for exact and efficient inference algorithms in these models. The material presented in this chapter is based on [110, 111]; the contribution of each author is assumed to be equal, unless explicitly stated. ${ }^{1}$
- In Chapter 6 we apply the algorithms from Chapter 5 to real-world machine learning problems. We provide an experimental comparison of two popular parameter estimation methods: maximum-margin and maximum-likelihood. The material presented in this chapter is based on [57, 58, 110, 111]; the contribution of each author is assumed to be equal, unless explicitly stated. ${ }^{1}$
- Chapter 7 discusses non-planar Ising graphical models and proposes heuristic algorithms for approximating ground states in these models. We compare these heuristic algorithms to the state-of-the-art algorithms described in Chapter 4 on binary-labeled pairwise Markov Random Fields. The work described in this chapter is in preparation for Journal of Machine Learning Research. ${ }^{1}$
- Chapter 8 contains a concise summary of our work and ideas for future work.

[^0]
## Background

This chapter introduces some of the key definitions and notations that are used throughout the thesis. In Section 2.3 we introduce the Dimer problem, which is a key concept in understanding the next chapter on the Ising model.

### 2.1 Mathematical Notation

We will use bold lowercase letters for vectors and bold uppercase for matrices. Calligraphic letters will refer to a set and an element of that set will be represented by the corresponding lowercase letter. Scalar quantities will be typeset in normal font. For example, $\boldsymbol{a}:=\left[a_{1}, a_{2}, \ldots, a_{n}\right]$ is a $n$-dimensional row vector, while $a$ is scalar. $\boldsymbol{B}:=\left[\boldsymbol{b}_{\mathbf{1}}, \boldsymbol{b}_{\mathbf{2}}, \ldots, \boldsymbol{b}_{\boldsymbol{m}}\right]$ is a $n \times m$ matrix, which can also be written as $\boldsymbol{B}=\left[b_{i j}\right]_{n \times m} . \mathcal{E}$ is a set and $e \in \mathcal{E}$ is an element of that set.

The meaning of $|\cdot|$ varies depending on the type of the argument. $|\boldsymbol{B}|=\operatorname{det} \boldsymbol{B}$ is the determinant of matrix $\boldsymbol{B},|\boldsymbol{b}|=n$ is the length of vector $\boldsymbol{b}$, while $|b|$ is the absolute value (magnitude) of the scalar $b$. If $\mathcal{S}$ is a set then $|\mathcal{S}|$ is the number of elements in $\mathcal{S}$, i.e., its cardinality. $\|\boldsymbol{b}\|:=\sqrt{b_{1}^{2}+\ldots+b_{n}^{2}}$ is the $L_{2}$-norm of $\boldsymbol{b} . \llbracket \cdot \rrbracket$ denotes the indicator function, its value is 1 if its argument is true and 0 otherwise, $e . g ., \llbracket a>b \rrbracket$ is 1 if $a>b$ and 0 if $a \leq b$. We will write sets using the curly brackets, i.e., $\{\cdot\}$. We will write permutations using the curved brackets, i.e., $(2,3,1,5,4)$ is a permutation of the first 5 natural numbers.
$\neg$ denotes Boolean negation, e.g., $\neg 1=0$ and $\neg 0=1 . e$ is the Euler's constant and $\ln x$ is the natural logarithm (base $e$ ) of $x$. Sometimes we will write $e^{x}$ as $\exp (x)$.
$\mathbb{P}(A)$ is the probability of event $A$, while $\mathbb{P}(A \mid B)$ is the probability of event $A$, given that event $B$ occured.

### 2.2 Graph Notation

This thesis uses a fair amount of graph theory, some of it standard and some of it not. Here we will describe the representation of graphs, graphical structures and common definitions.

Informally, a graph is an object which can be visualized as a collection of lines. These lines are the edges of the graph, they can be either undirected or directed. A graph is directed if all its edges are directed; undirected if all its edges are undirected. The endpoints of the edges are called the nodes (or
the vertices) of the graph.
Formally, we will write a graph as $G(\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is the set of vertices and $\mathcal{E}$ is the set of edges of $G$. A vertex (or node) $i$ will be represented as $i \in \mathcal{V}$. If $G$ is undirected then an edge between node $i$ and node $j$ will be represented as $(i, j) \in \mathcal{E}$ and $(i, j)=(j, i)$. If $G$ is directed then an edge from node $i$ to node $j$ will be represented as $(i, j) \in \mathcal{E}$. For directed graphs $G$, we will usually assume that if $(i, j) \in \mathcal{E}$ then $(j, i) \notin \mathcal{E}$. Often we will work with weighted graphs, i.e., graphs whose edges are associated with weights. We also need the definition of a clique:

Definition $1 A$ clique is a set of nodes, such that every pair of nodes in that set is connected by an edge. A maximal clique is a clique which is not a proper subset of another clique. A clique with n nodes is often called $K_{n}$.

Finally, we will define a cycle and a component:
Definition 2 A cycle of a graph $G(\mathcal{V}, \mathcal{E})$ is a closed path of its vertices $v_{1}, v_{2}, \ldots, v_{k}$, such that $\left(v_{k}, v_{1}\right) \in$ $\mathcal{E}$ andfor every $i<k:\left(v_{i}, v_{i+1}\right) \in \mathcal{E}$. The set of edges of the cycle is $O:=\left\{\left(v_{1}, v_{2}\right), \ldots,\left(v_{i}, v_{i+1}\right), \ldots,\left(v_{k}, v_{1}\right)\right\}$. A cycle is called simple if it has no repeated edges or vertices.

Definition 3 A component of a graph with respect to a given property is a maximal subgraph that has the property.

### 2.2.1 Connectivity and Biconnectivity

Most of the algorithms in this thesis assume that the graph is connected:
Definition 4 An undirected graph $G(\mathcal{V}, \mathcal{E})$ is connected if $\mathcal{E}$ contains at least one path between any two nodes of $\mathcal{V}$. If this is not the case then the graph is disconnected.

For example the graph in Figure 2.1(a) is connected, while the graph in Figure 2.1(b) is not. A connected component (Definition 3) is the maximal subgraph in which all nodes are connected. In Figure 2.1(b) there are 2 connected components. Clearly if a graph is connected then it must contain exactly one connected component. A related, but stronger requirement is that of biconnectivity:

Definition 5 A graph is biconnected iff it is connected and does not have any articulation vertices. An articulation vertex or a cut vertex is a vertex whose removal (along with any incident edges) disconnects the graph.

The graph in Figure 2.1(a) is connected, but not biconnected, because the central vertex is an articulation vertex. However, the graph in Figure 2.1(c) is biconnected.

### 2.2.2 Planar Graphs

Definition 6 A graph is planar if it can be drawn in the plane $\mathbb{R}^{2}$ (infinite flat surface) without edge intersections. The regions into which such a plane drawing partitions $\mathbb{R}^{2}$ are the faces of the drawing; the unbounded region is the external face.


Figure 2.1: (a) connected graph. (b) disconnected graph. (c) biconnected graph.


Figure 2.2: (a) a non-plane drawing of a planar graph; (b) a plane drawing of the same graph; (c) a different plane drawing of same graph, with the same planar embedding as (b); (d) a plane drawing of the same graph with a different planar embedding.

This definition might suggest that our algorithms must produce (or have access to) a plane drawing of the graph. In practice, the drawing of the graph contains information that is not useful for us, such as: the precise location of the vertices, and the exact shape of the edges. All we care about is the cyclic (say, clockwise) ordering of the edges incident upon each vertex. In topological graph theory, this is formalized in the notion of a rotation system [138, p. 21f]:

Definition 7 Let $G(\mathcal{V}, \mathcal{E})$ be an undirected, connected graph. For each vertex $i \in \mathcal{V}$, let $\mathcal{E}_{i}$ denote the set of edges in $\mathcal{E}$ incident upon $i$, considered as being oriented away from $i$, and let $\pi_{i}$ be a cyclic ordering of $\mathcal{E}_{i}$. A rotation system for $G$ is a set $\Pi=\left\{\pi_{i}: i \in \mathcal{V}\right\}$.

For example for vertex 4 in Figures 2.2(b) and 2.2(c), $\mathcal{E}_{4}=\{(4,3),(4,1),(4,2),(4,5)\}$ and $\pi_{4}=$ $((4,5),(4,3),(4,2),(4,1))$. With some abuse of notation, let $\pi_{i}(i, j)$ denote the successor edge of $(i, j)$ in the cyclic ordering $\pi_{i}$. For example, $\pi_{4}(4,2)=(4,1)$ and $\pi_{4}(4,1)=(4,5)$. Informally, this means that if we are located at vertex 4 and move clockwise from $(4,2)$ then we will reach edge $(4,1)$. To define the sets $\mathcal{E}_{i}$ of oriented edges more formally, construct the directed graph $G\left(\mathcal{V}, \mathcal{E}^{\prime}\right)$, where $\mathcal{E}^{\prime}$ contains a pair of directed edges (known as edgelets) for each undirected edge in $\mathcal{E}$, that is, $(i, j) \in \mathcal{E}^{\prime} \Longleftrightarrow[(i, j) \in \mathcal{E} \vee(j, i) \in \mathcal{E}]$. Then $\mathcal{E}_{i}=\left\{(j, k) \in \mathcal{E}^{\prime}: i=j\right\}$. Rotation systems directly correspond to topological graph embeddings in orientable surfaces:

Theorem 8 (White and Beineke [138]) Each rotation system determines an embedding of $G$ in some orientable surface $S$ such that $\forall i \in \mathcal{V}$, any edge $(i, j) \in \mathcal{E}_{i}$ is followed by $\pi_{i}(i, j)$ in (say) clockwise orientation, and such that the faces $\mathcal{F}$ of the embedding, given by the orbits (cycles) of the mapping

## $(i, j) \rightarrow \pi_{j}(j, i)$, are 2 -cells (topological disks).

Note that while in graph visualisation "embedding" is often used as a synonym for "drawing", in modern topological graph theory it stands for "rotation system". We adopt the latter usage, which views embeddings as equivalence classes of graph drawings characterized by identical cyclic ordering of the edges incident upon each vertex. The graphs in Figures 2.2(b) and 2.2(c) have the same embedding as evident from the equivalence of their $\pi_{i}$. The embedding in Figure 2.2(d) differs, since we have $\pi_{4}^{\prime}=((4,5),(4,1),(4,3),(4,2))$ and $\pi_{4}^{\prime}(4,2) \neq \pi_{4}(4,2)$. A sample face in Figure 2.2(b) or 2.2(c) is given by the orbit

$$
(4,1) \rightarrow\left[\pi_{1}(1,4)=(1,2)\right] \rightarrow\left[\pi_{2}(2,1)=(2,4)\right] \rightarrow\left[\pi_{4}(4,2)=(4,1)\right] .
$$

The same starting edge in Figure 2.2(d) gives the orbit

$$
(4,1) \rightarrow(1,2) \rightarrow(2,4) \rightarrow(4,5) \rightarrow(5,4) \rightarrow(4,1)
$$

Definition 9 The genus $g$ of a surface is the maximum number of non-intersecting closed curves which one can draw on the surface without disconnecting it.

Intuitively, the genus $g$ of the embedding surface $S$ is the number of "holes" in $S$. So if $S$ is a plane or a sphere then $g=0$. If $S$ is a torus (donut) then $g=1$. The genus can be determined from the Euler characteristic

$$
\begin{equation*}
|\mathcal{V}|-|\mathcal{E}|+|\mathcal{F}|=2-2 g, \tag{2.1}
\end{equation*}
$$

where $|\mathcal{F}|$ is found by counting the number of orbits in the rotation system, as described in Theorem 8. Since planar graphs are exactly those that can be embedded on a surface of genus $g=0$ (a topological sphere), we arrive at a purely combinatorial definition of planarity:

Definition 10 A graph $G(\mathcal{V}, \mathcal{E})$ is planar iff it has a rotation system $\Pi$ producing exactly $2+|\mathcal{E}|-|\mathcal{V}|$ orbits. Such a system is called a planar embedding of $G$, and $G(\mathcal{V}, \mathcal{E}, \Pi)$ is called a plane graph.

A nice characterization of planar graphs was given by Kuratowski [78]:
Theorem 11 A graph is planar iff it does not contain a subgraph that is homeomorphic to $K_{3,3}$ or $K_{5}$ (see Figure 2.3 and Definition 1).

Theorem 11 implies that the graphs in Figure 2.3 are not planar. Indeed it is not possible to draw them on a plane without any crossing edges. The graph in Figure 2.3(c) is homeomorphic to the graph in Figure 2.3(b). A graph $G_{1}$ is homeomorphic to a graph $G_{2}$ if a sequence of allowed steps transform it into $G_{2}$. The allowed steps are:

1. Edge $(i, j)$ can be subdivided into two edges $(i, w)$ and $(w, j)$ via the addition of vertex $w$.
2. Two edges $(i, w)$ and $(w, j)$ can be merged into a single edge $(i, j)$ via the removal of vertex $w$, provided that the degree of $w$ is 2 .


Figure 2.3: Examples of non-planar graphs. (a) Utility graph $K_{3,3}$. (b) Complete graph of 5 nodes or $K_{5}$. (c) A graph that is homeomorphic to $K_{5}$.

All planar graphs have a corresponding dual graph:
Definition 12 The dual $G^{*}(\mathcal{F}, \mathcal{E})$ of an embedded graph $G(\mathcal{V}, \mathcal{E}, \Pi)$ has a vertex for each face of $G$, with edges connecting vertices corresponding to faces that are adjacent (i.e., share an edge) in $G$.
$G^{*}$ is a pseudograph, i.e., it may contain loops and multiple edges (see Figure 2.4). It is important to note that $G^{*}$ has the same set of edges as $G$ : a dual edge crosses its corresponding edge in $G$. We will use this fact to simplify many algorithms.


Figure 2.4: The original graph $G$ (blue) with its corresponding dual graph $G^{*}$ (red). Every edge in $G$ is crossed by an edge in $G^{*}$ and vice versa.

### 2.3 Dimer Problem

The dimer problem, also known as the domino tiling problem, is a fundamental problem in graph theory, which underpins many of the definitions and algorithms described in this thesis. We will begin by introducing the problem, followed by a detailed description of its solution.

Informally the dimer problem asks us in how many ways can we cover the edges of a graph with dimers. A dimer is a graph that consists of a single undirected edge. For example, there are 3 ways to cover the graph in Figure 2.5(a) with dimers, shown in Figures 2.5(b, c, d).


Figure 2.5: A graph (a) and all its 3 corresponding dimer coverings (in bold) shown in (b), (c) and (d).

More formally, the problem is to count the number of dimer coverings (Definition 13) or equivalently the number of perfect matchings (Definition 14) in a graph:

Definition 13 A dimer configuration of a graph $G$ is a subgraph of $G$, whose connected components (Definitions 3, 4) are dimers. A dimer covering of $G$ is a dimer configuration which spans all the vertices of $G$.

Definition 14 A perfect matching of a graph $G(\mathcal{V}, \mathcal{E})$ is a subset $\mathcal{M} \subseteq \mathcal{E}$ of edges wherein exactly one edge is incident upon each vertex: $\forall v \in \mathcal{V},|v|=1$ in $G(\mathcal{V}, \mathcal{M})$. Its weight $w(\mathcal{M})$ is the sum of the weights of its edges.

It is important to note that not all graphs have a perfect matching. For example, graphs with an odd number of nodes cannot have a perfect matching. A dimer covering can be represented as a set of edges (dimers) that are included in the covering. For example, a dimer covering of Figure 2.5(b) can be written as $\left\{\left(v_{4}, v_{1}\right),\left(v_{3}, v_{6}\right),\left(v_{2}, v_{5}\right)\right\}$. Note that there are many ways to write the same dimer covering, since we can interchange the vertices within each dimer, as well as, the order of the dimers. In the next section, we will present a canonical representation of dimer coverings.

### 2.3.1 Counting Dimer Coverings

Let $G(\mathcal{V}, \mathcal{E})$ be a graph that admits a perfect matching, where $\mathcal{V}=\left\{v_{1}, \ldots, v_{2 n}\right\}$ is the set of $2 n$ vertices and $\mathcal{E}$ is the set of edges. Remember that a dimer covering must span all the vertices, thus a dimer covering of $G$ will contain exactly $n$ dimers. As discussed in the previous section, the order of vertices and dimers in a dimer covering does not matter. Thus each dimer covering can be represented in $2^{n} n$ ! ways. From these representations we would like to select a single (unique) representation.

We will begin by defining a canonical representation of a dimer covering. Define the $k^{\text {th }}$ dimer in the representation to be the edge connecting nodes $v_{i_{k}}$ and $v_{j_{k}}$. For uniqueness we will write this edge as ( $v_{i_{k}}, v_{j_{k}}$ ) where $i_{k}<j_{k}$. We can define a total order over dimers based on their first vertices: for
each pair of dimers, let $\left(v_{i_{k}}, v_{j_{k}}\right)<\left(v_{i_{m}}, v_{j_{m}}\right)$ iff $i_{k}<i_{m}$. Now we can simply sort the dimers in a dimer covering and obtain its unique representation:

$$
\begin{equation*}
\left\{\left(v_{i_{1}}, v_{j_{1}}\right),\left(v_{i_{2}}, v_{j_{2}}\right), \ldots,\left(v_{i_{n}}, v_{j_{n}}\right)\right\} \tag{2.2}
\end{equation*}
$$

where $i_{k}<i_{k+1} \forall k$. For example, using this representation the dimer covering in $2.5(\mathrm{~b})$ becomes $\left\{\left(v_{1}, v_{4}\right),\left(v_{2}, v_{5}\right),\left(v_{3}, v_{6}\right)\right\}$. In summary we have partitioned the set of vertex indices $\{1,2, \ldots, 2 n\}$ into $n$ pairs: $\left\{\left(i_{1}, j_{1}\right),\left(i_{2}, j_{2}\right), \ldots,\left(i_{n}, j_{n}\right)\right\}$ such that $i_{k}<j_{k}, i_{k}<i_{k+1} \forall k$. Now to count the dimer coverings, all we need is a sum over the set of such partitions. It turns out that the form of this sum looks very similar to the Pfaffian of a particular matrix $\boldsymbol{K}$.

### 2.3.2 Pfaffian

In the previous section we have hinted that the number of dimer coverings can be computed via a Pfaffian of a particular matrix $\boldsymbol{K}$. Now we will provide a formal definition of the Pfaffian, which will be followed by the construction of $\boldsymbol{K}$.

Consider a generic $2 n \times 2 n$ matrix $\boldsymbol{A}=\left[a_{i j}\right]_{2 n \times 2 n}$. Let $\alpha$ be a partition of the set $\{1,2, \ldots, 2 n\}$ into $n$ pairs: $\alpha:=\left\{\left(i_{1}, j_{1}\right),\left(i_{2}, j_{2}\right), \ldots,\left(i_{n}, j_{n}\right)\right\}$ such that $i_{k}<j_{k} \forall k$ and $i_{1}<i_{2}<\ldots<i_{n} . P$ is the set of all such partitions $\alpha$. Let

$$
\begin{align*}
\pi_{\alpha} & :=\left(\begin{array}{cccccccccc}
1 & 2 & 3 & 4 & \ldots & 2 k-1 & 2 k & \ldots & 2 n-1 & 2 n \\
i_{1} & j_{1} & i_{2} & j_{2} & \ldots & i_{k} & j_{k} & \ldots & i_{n} & j_{n}
\end{array}\right)  \tag{2.3}\\
& =\left(i_{1}, j_{1}, i_{2}, j_{2}, \ldots, i_{n}, j_{n}\right) \tag{2.4}
\end{align*}
$$

be the permutation ${ }^{2}$ corresponding to $\alpha$, so $\pi_{\alpha}(1)=i_{1}, \pi_{\alpha}(2)=j_{1}, \ldots, \pi_{\alpha}(2 k-1)=i_{k}, \pi_{\alpha}(2 k)=j_{k}$ and so on. Notice that (2.4) corresponds precisely to the representation of dimer coverings in (2.2). The Pfaffian of $\boldsymbol{A}$ is then given by

$$
\begin{align*}
\operatorname{Pf} \boldsymbol{A} & :=\sum_{\alpha \in P} \operatorname{sgn} \pi_{\alpha} \boldsymbol{A}_{\boldsymbol{i}_{1}, \boldsymbol{j}_{\mathbf{1}}} \boldsymbol{A}_{\boldsymbol{i}_{2}, \boldsymbol{j}_{2}} \ldots \boldsymbol{A}_{\boldsymbol{i}_{n}, \boldsymbol{j}_{n}} \\
& =\sum_{\alpha \in P} \operatorname{sgn} \pi_{\alpha} \boldsymbol{A}_{\pi_{\alpha}(\mathbf{1}), \pi_{\alpha}(\mathbf{2})} \boldsymbol{A}_{\boldsymbol{\pi}_{\alpha}(\mathbf{3}), \boldsymbol{\pi}_{\alpha}(\mathbf{4})} \ldots \boldsymbol{A}_{\boldsymbol{\pi}_{\alpha}(\mathbf{2 n - 1}), \pi_{\alpha}(\mathbf{2 n})} \\
& =\sum_{\alpha \in P} \operatorname{sgn} \pi_{\alpha} \prod_{k=1}^{n} \boldsymbol{A}_{\pi_{\alpha}(\mathbf{2} k-\mathbf{1}), \pi_{\alpha}(\mathbf{2 k})}, \tag{2.5}
\end{align*}
$$

where $\operatorname{sgn} \pi_{\alpha}$ is the signature of $\pi_{\alpha}$ and is defined as +1 if $\pi_{\alpha}$ is even and -1 if $\pi_{\alpha}$ is odd. A permutation is even if it can be expressed as a composition of an even number of transpositions, and otherwise it is odd. A transposition $(i, j)$ is a function that maps $i$ to $j$ and vice-versa. For example, the permutation that turns the list $\{1,2,3,4,5\}$ into $\{3,4,5,2,1\}$ is odd because it can only be written as a composition of an odd number of transpositions: $(1,3,5)(2,4)=(1,3)(1,5)(2,4)$.

[^1]
## Skew-Symmetry

A skew-symmetric matrix is a matrix $\boldsymbol{A}$ such that $\boldsymbol{A}=-\boldsymbol{A}^{\top}$ (or $\boldsymbol{A}_{\boldsymbol{i} j}=-\boldsymbol{A}_{\boldsymbol{j} \boldsymbol{i}}$ for all $i, j$ ). The Pfaffian of a skew-symmetric matrix has some nice properties that we can exploit. Consider a single term of such a Pfaffian computation $t_{\pi}:=\operatorname{sgn} \pi \boldsymbol{A}_{\boldsymbol{\pi}(\mathbf{1}), \pi(2)} \ldots \boldsymbol{A}_{\boldsymbol{\pi}(\mathbf{2 n - 1}), \pi(2 n)}$. The following modifications of $\pi=\left(i_{1}, j_{1}, \ldots, i_{n}, j_{n}\right)$ leave $t_{\pi}$ unchanged:

1. Interchange of any two partners: $i_{k}$ and $j_{k} \cdot \operatorname{sgn} \pi$ changes sign, but then
$\boldsymbol{A}_{\boldsymbol{\pi}(\mathbf{2 k - 1}), \boldsymbol{\pi}(\mathbf{2 k})}$ becomes $\boldsymbol{A}_{\pi(\mathbf{2 k}), \boldsymbol{\pi}(\mathbf{2 k - 1})}=-\boldsymbol{A}_{\pi(\mathbf{2 k - 1 ) , \pi ( 2 k )}}$, and thus $t_{\pi}$ remains unchanged.
2. Interchange of any two pairs of partners: $\left(i_{k}, j_{k}\right)$ and $\left(i_{m}, j_{m}\right)$. This does not affect $\operatorname{sgn} \pi$ and only interchanges $\boldsymbol{A}_{\boldsymbol{\pi}(\mathbf{2 k - 1}), \boldsymbol{\pi}(\mathbf{2 k})}$ with $\boldsymbol{A}_{\boldsymbol{\pi}(\mathbf{2 m - 1}), \boldsymbol{\pi}(\mathbf{2 m})}$, which leaves $t_{\pi}$ unchanged.
3. Obviously any combination of 1 and 2.

In 1849 Cayley [15] proved that the Pfaffian of any skew-symmetric matrix $\boldsymbol{A}$ can be computed via its determinant:

$$
\begin{equation*}
\mathrm{Pf}^{2} \boldsymbol{A}=\operatorname{det} \boldsymbol{A} \tag{2.6}
\end{equation*}
$$

Note that the Pfaffian of an odd $n \times n$ skew-symmetric matrix is defined to be zero, as the determinant of an odd skew-symmetric matrix is zero. This can be seen as

$$
\left.\operatorname{det} \boldsymbol{A}=\operatorname{det} \boldsymbol{A}^{\top}=\operatorname{det}(-\boldsymbol{A})\right)=(-1)^{n} \operatorname{det} \boldsymbol{A}
$$

When $n$ is odd, this implies that $\operatorname{det} \boldsymbol{A}=-\operatorname{det} \boldsymbol{A}$, i.e., $\operatorname{det} \boldsymbol{A}=0$. (2.6) is a very useful property, because it means that we can compute Pfaffians in polynomial time in the number of elements. From now on when we refer to the Pfaffian of a matrix, we will assume that the matrix is even and skew-symmetric.

## Example

We now present a detailed example of a Pfaffian computation. Consider the matrix $\boldsymbol{A}$ defined as

$$
\boldsymbol{A}:=\left(\begin{array}{cccc}
0 & a & b & c \\
-a & 0 & d & e \\
-b & -d & 0 & f \\
-c & -e & -f & 0
\end{array}\right)
$$

The following table shows all the possible allowed partitions $\alpha$ of the set $\{1,2,3,4\}$. For each $\alpha, \pi_{\alpha}$ is the associated permutation, $\operatorname{sgn} \pi_{\alpha}$ is its signature and the right-most column is the contribution to the Pfaffian sum.

| $\alpha$ | $\pi_{\alpha}$ | $\operatorname{sgn} \pi_{\alpha}$ | contribution |
| :---: | :---: | :---: | :---: |
| $\{(1,2),(3,4)\}$ | $I$ | +1 | $+a f$ |
| $\{(1,4),(2,3)\}$ | $(2,4)(2,3)$ | +1 | $+c d$ |
| $\{(1,3),(2,4)\}$ | $(2,3)$ | -1 | $-b e$ |

Summing the contribution column we obtain the final answer: $\operatorname{Pf} \boldsymbol{A}=a f-b e+d c$. Now consider the determinant of $\boldsymbol{A}$. Ignoring the terms that are 0 , we have:

$$
\begin{aligned}
\operatorname{det} \boldsymbol{A} & =2 a f d c-2 a f b e-2 b e d c+a^{2} f^{2}+b^{2} e^{2}+d^{2} c^{2} \\
& =(a f-b e+d c)^{2}=\operatorname{Pf}^{2} \boldsymbol{A}
\end{aligned}
$$

### 2.3.3 Constructing the Kasteleyn Matrix

We now discuss what $\boldsymbol{K}$ looks like and state precisely its relationship to $G$. We will refer to $\boldsymbol{K}$ as the Kasteleyn matrix, due to its founder Kasteleyn [59, 61].

First of all, we know that the row and column indices of $\boldsymbol{K}$ must correspond to the vertices of $G$, i.e., $\boldsymbol{K}$ is a $|\mathcal{V}| \times|\mathcal{V}|$ sized matrix. We also know that the terms in Pf $\boldsymbol{K}$ that do not correspond to dimer coverings must be 0 . To ensure this, we set $\boldsymbol{K}_{i j}:=0$ if there is no edge between $v_{i}$ and $v_{j}$. This must also include $\boldsymbol{K}_{i i}:=0$, since we are not allowed to have any self-loops.

Since we want to evaluate $\operatorname{Pf} \boldsymbol{K}$ via (2.6) we must make $\boldsymbol{K}$ skew-symmetric. This is achieved by letting $\boldsymbol{K}_{i j}:=-\boldsymbol{K}_{\boldsymbol{j} i}$, which still leaves us with a lot of freedom.

We now have that all non-zero terms in $\operatorname{Pf} \boldsymbol{K}$ correspond to actual dimer coverings. In order to count the dimer coverings correctly, these terms must contribute +1 to the sum in Equation 2.5. Also they can all be -1 , since $\operatorname{Pf}^{2} \boldsymbol{K}=(-\operatorname{Pf} \boldsymbol{K})^{2}=\operatorname{Pf}^{2}(-\boldsymbol{K})$ will still produce a valid real-valued determinant. To summarize, we want all non-zero terms in $\operatorname{Pf} \boldsymbol{K}$ to have the same sign. This implies that the product of any pair of non-zero terms, corresponding to two dimer coverings, should be positive:

$$
\begin{equation*}
\operatorname{sgn} \pi_{\beta} \operatorname{sgn} \pi_{\gamma} \prod_{k=1}^{n} K_{\pi_{\beta}(2 k-1), \pi_{\beta}(2 k)} K_{\pi_{\gamma}(2 k-1), \pi_{\gamma}(2 k)}>0 \tag{2.7}
\end{equation*}
$$

where $\beta, \gamma \in P$ are the partitions corresponding to the two dimer coverings. Although $\beta$ and $\gamma$ are fixed, we have freedom in selecting $\pi_{\beta}$ and $\pi_{\gamma}$, provided that $\operatorname{sgn} \pi_{\beta}$ and $\operatorname{sgn} \pi_{\gamma}$ remain unchanged.

Let us try to interpret the product in (2.7). To facilitate further discussion it is convenient to define a superposition of two dimer coverings. The superposition $S\left(D_{i}, D_{k}\right)$ of two dimer coverings $D_{i}$ and $D_{k}$ is a union of $D_{i}$ and $D_{k}$ whose edges alternate between $D_{i}$ and $D_{k}$.

We now describe how a superposition is constructed referring to the example in Figure 2.6. Let $D_{\beta}$ (Figure 2.6 bold) and $D_{\delta}$ (Figure 2.6 dashed) be the dimer coverings that are generated with partitions $\beta:=\{(1,4),(2,3),(5,6)\}$ and $\gamma:=\{(1,4),(2,5),(3,6)\}$ respectively. We begin with an arbitrary starting node, say $v_{1}$. This node (as all other nodes) is adjacent to exactly one edge in $D_{\beta}$. Following this edge we arrive at $v_{4} . v_{4}$ will be adjacent to a single edge in $D_{\gamma}$. Following that edge, we will either arrive at our initial node (which is the case here) or a new node. Since there is a finite number of nodes, we will always arrive at our starting node, thus completing a cycle. Once we have completed a cycle we select a new arbitrary starting node (that is not already in a cycle) and repeat the procedure. At the end of this procedure, each node will belong to exactly one cycle, called a superposition cycle. The superposition cycles visit every node and therefore constitute a cycle covering, in which every cycle has an even length. In Figure 2.6 the superposition cycles of $S\left(D_{\beta}, D_{\delta}\right)$ are: 1-4-1 and 2-3-6-5-2. The permutation of this superposition can be conveniently written as a product of even cyclic groups corresponding to each superposition cycle, i.e., $(1,4)(2,3,6,5)$.


Figure 2.6: Superposition $S\left(D_{\beta}, D_{\delta}\right)$ of two dimer coverings $D_{\beta}$ and $D_{\gamma}$.

It should now become apparent that it is possible to select $\pi_{\beta}$ and $\pi_{\gamma}$ such that the product in (2.7) corresponds to a superposition of $D_{\beta}$ and $D_{\delta}$. In particular, $\operatorname{sgn} \pi_{\beta} \operatorname{sgn} \pi_{\gamma}$ can be written as $\operatorname{sgn} \pi$ where $\pi$ is a product of even cyclic groups. Each such group thus contributes a factor of ( -1 ) to $\operatorname{sgn} \pi$. We now need to select terms in $\boldsymbol{K}$ to cancel out these minus signs.

The simplest case is when the cyclic group has length 2. In this case the minus sign is automatically compensated, because the factors contributing to (2.7) take the form $\boldsymbol{K}_{\boldsymbol{i j}} \boldsymbol{K}_{\boldsymbol{j i}}=-\boldsymbol{K}_{\boldsymbol{i} \boldsymbol{i}}^{\mathbf{2}}$, which is negative.

For all other cyclic groups we require an odd number of negative terms to cancel out the minus sign of the signature. Let us represent the sign of a matrix element $\boldsymbol{K}_{i j}$ by an orientation of the edge $\left(v_{i}, v_{j}\right)$. In particular if $\boldsymbol{K}_{i j}=+1$ then the edge is oriented from $v_{i}$ to $v_{j}$, otherwise it is oriented from $v_{j}$ to $v_{i}$. An odd number of negative terms now corresponds to an odd number of edges oriented in the same direction. We can now formalize the requirement for the signs of the matrix elements:

Lemma 15 Each cycle occurring in a superposition of two dimer coverings must contain an odd number of edges oriented in the same direction.

Here it is convenient to introduce the orientation of the graph:
Definition 16 An orientation of an undirected $\operatorname{graph} G(\mathcal{V}, \mathcal{E})$ is a set $\mathcal{E}^{\prime}$ of oriented edges with $\left|\mathcal{E}^{\prime}\right|=|\mathcal{E}|$ such that for each $(i, j) \in \mathcal{E}, \mathcal{E}^{\prime}$ contains either $(i, j)$ or $(j, i)$, but not both.

An orientation of the edges of $G$ satisfying Lemma 15 is called an admissible orientation or a Pfaffian orientation. We say that a graph $G$ is Pfaffian if there exists a Pfaffian orientation for $G$. Notice that we do not require that every even cycle of $G$ has an odd orientation parity, but only those cycles that are a result of a superposition of two dimer coverings. This is a real restriction, without which the condition would be difficult to satisfy. In the next section we show which graphs are Pfaffian and how to construct an admissible orientation.

### 2.3.4 Pfaffian Graphs

The complete characterization of Pfaffian graphs remains an unsolved problem. However, Kasteleyn $[60,61]$ made considerable progress by showing the following fundamental theorem

Theorem 17 (Kasteleyn Theorem) Every planar graph has a Pfaffian orientation.

To prove Theorem 17 we need to show that every planar graph can be oriented such that every superposition cycle has an odd number of edges oriented in the same direction. The direction of an edge in a cycle refers to the direction of that edge in reference to the center of the cycle. For convenience we
chose the clockwise direction as our reference direction. Thus we require that every superposition cycle has an odd number of edges oriented clockwise, i.e., each of its cycles is clockwise odd. We call a face clockwise odd if the cycle on its perimeter is clockwise odd. We begin by showing that

Lemma 18 Every plane graph $G(\mathcal{V}, \mathcal{E}, \Pi)$ can be oriented such that every face (except possibly one) is clockwise odd. This orientation is called the clockwise odd orientation.

Proof (based on [61, p. 93]) We will prove this Lemma constructively. Select an arbitrary face $f_{1} \in \mathcal{F}$ and orient all its edges except one arbitrarily. We can always orient the last edge, such that $f_{1}$ is clockwise odd. Now select an adjacent face $f_{2}$, one that shares an edge with $f_{1} . f_{2}$ must have one or more other edges. We orient all these edges except one arbitrarily. Again we can choose the last edge to make $f_{2}$ clockwise odd and move on to an adjacent face. We continue this process until all faces have been oriented. This process encounter problems if orienting the last edge of $f_{i}$ completes the orientation of two faces at once. If this occurs while orienting the very last edge then the Lemma 18 still holds, since we would have already oriented $|\mathcal{F}|-1$ faces.

However we need to consider carefully if this occurs earlier (Figure 2.7). This will happen only if we select faces in such way that they form a region that is not simply connected, i.e., a region that contains "holes" (such as face 5 in Figure 2.7). We can overcome this problem by working with the dual graph $G^{*}\left(\mathcal{V}^{*}, \mathcal{E}^{*}\right)$. Each dual node $i_{k} \in \mathcal{V}^{*}$ correspond to a face $f_{k}$ of $G$. Starting from an arbitrary node $i_{n} \in \mathcal{V}^{*}$, we traverse $G^{*}$ (in any order) and output the list of visited nodes in postorder $i_{1}, i_{2}, \ldots, i_{n}$. We now give a clockwise odd orientation to the faces of $G$ in the order $f_{1}, f_{2}, \ldots, f_{n}$. It may or may not be possible to give a clockwise odd orientation to the last face $f_{n}$, however Lemma 18 still holds.

The reason this algorithm works is the following. If we are orienting face $f_{i}$ with $i<n$ then there must exist $k>i$ such that $f_{k}$ borders $f_{i}$ and $f_{k}$ hasn't been oriented. This means that $f_{i}$ has at least one unoriented edge - the edge shared by $f_{i}$ and $f_{k}$. We can choose how to orient this edge and thus make $f_{i}$ clockwise odd. Therefore, the first $i$ such that $f_{i}$ cannot be given a clockwise odd orientation is $i=n$, in which case $n-1$ faces have a clockwise odd orientation and Lemma 18 holds. For further explanation and pseudocode see Section 5.3.2.

Lemma 19 In a plane graph $G(\mathcal{V}, \mathcal{E}, \Pi)$, if all faces are clockwise odd then every cycle enclosing an even number of vertices is clockwise odd.

Proof (based on [87, p. 321]) For every cycle, the number of clockwise edges and the number of vertices that it encloses must have opposite parity. Let $C$ be any cycle in $G$. Suppose there are $f$ faces inside $C$ and let $c_{i}$ be the number of clockwise edges on the boundary of face $i$ for $i=1, \ldots, f$. Since all faces are clockwise odd, each $c_{i}$ is odd and hence

$$
\begin{equation*}
f \equiv \sum_{i=1}^{f} c_{i} \quad(\bmod 2) \tag{2.8}
\end{equation*}
$$

Let $v$ be the number of vertices enclosed by $C, e$ the number of edges inside $C, k$ the number of edges on $C$ and $c$ the number of those edges that are clockwise. Thus the number of vertices in $C$ is $v+k$, the


Figure 2.7: This graph is embedded on a sphere and now the "external face" 6 simply becomes another face. If the faces are oriented in the order $1,2,3,4,5,6$ then there is no way to orient the edge $(a, b)$ such that both faces 4 and 5 are clockwise odd. Since face 6 is also not clockwise odd, there will be two faces that are not clockwise odd, thus violating the requirement of Lemma 18.
number of edges in $C$ is $e+k$ and the number of faces in $C$ is $f+1$ (we also count the outer face). Thus by Euler's formula we have

$$
|\mathcal{V}|-|\mathcal{E}|+|\mathcal{F}|=v-e+f+1=2
$$

and hence

$$
\begin{equation*}
e=v+f-1 \tag{2.9}
\end{equation*}
$$

Every edge inside $C$ will border exactly two faces. In one of these faces it will be clockwise, while in the other it will be counter-clockwise. Thus counting all clockwise edges for each face is equivalent to counting all edges inside $C$, so we have $\sum_{i=1}^{f} c_{i}=c+e$. Combining this with Equation 2.9 we can now rewrite Equation 2.8 as

$$
\begin{equation*}
f \equiv \sum_{i=1}^{f} c_{i}=c+e=c+v+f-1 \quad(\bmod 2) \tag{2.10}
\end{equation*}
$$

and hence $c+v-1 \equiv 0(\bmod 2)$. This implies that $c+v$ is odd, meaning that $c$ has an opposite parity to $v$.

The combination of Lemmas 18 and 19 tells us that every plane graph can be oriented such that every cycle enclosing an even number of points is clockwise odd. To complete the proof of Theorem 17 we need the following to hold

Lemma 20 In a plane graph $G(\mathcal{V}, \mathcal{E}, \Pi)$, all superposition cycles enclose an even number of vertices.
Proof (based on [61, p. 94]) By definition, a plane graph cannot contain any crossing edges. This implies that the superposition cycles cannot cross. Thus, each superposition cycle encloses either zero or
more complete superposition cycles. Since each superposition cycle contains an even number of vertices, we can conclude that each superposition cycle will enclose an even number of vertices (including zero).

Since the set of superposition cycles is a subset of all cycles, combining Lemma 20 with Lemmas 18 and 19 proves that every plane graph can be oriented such that every superposition cycle is clockwise odd. In other words we have shown that every planar graph is Pfaffian. Finally it must be noted that the construction used for Lemma 18 is a polynomial-time algorithm for constructing a Pfaffian orientation for plane graphs.

It turns out that the class of Pfaffian graphs is much larger than just those that are planar [127]:

- Every finite graph with no subgraph homeomorphic to $K_{3,3}$ is Pfaffian [85].
- A graph $G$ is Pfaffian if it can be drawn in the plane (possibly with crossings) so that every perfect matching of $G$ intersects itself an even number of times, i.e., has an even crossing number [96].

Note that the above only show the existence of a Pfaffian orientation, while constructing the Pfaffian orientation in polynomial-time may still be impossible. For a general graph with genus $g$, Kasteleyn [61] noted that the number of perfect matchings can be computed by evaluating $4^{g}$ determinants. This result was independently proven by Galluccio and Loebl [36] and Tesler [125].

### 2.3.5 Examples

We now present a complete example of counting the number of perfect matchings in a graph. Consider the graph in Figure $2.5(\mathrm{a})$. We orient the edges of the graph, such that each face (except possibly the outer face) has an odd number of edges oriented clockwise (Figure 2.8).


Figure 2.8: Pfaffian or clockwise-odd orientation of a graph. Notice that the outer face 1-2-3-6-5-4-1 does not need to be clockwise-odd, although it is in this case.

Based on this oriented graph $G\left(\mathcal{V}, \mathcal{E}^{\prime}\right)$, we construct the corresponding skew-symmetric matrix $\boldsymbol{K}$ using the following rules:

$$
\boldsymbol{K}_{i j}= \begin{cases}+1 & \text { if }\left(v_{i}, v_{j}\right) \in \mathcal{E}^{\prime}  \tag{2.11}\\ -1 & \text { if }\left(v_{j}, v_{i}\right) \in \mathcal{E}^{\prime} \\ 0 & \text { if } \mathrm{i}=\mathrm{j} \\ 0 & \text { otherwise }\end{cases}
$$

Using the above, we obtain the matrix

$$
\boldsymbol{K}=\left(\begin{array}{rrrrrr}
0 & +1 & 0 & -1 & 0 & 0 \\
-1 & 0 & +1 & 0 & +1 & 0 \\
0 & -1 & 0 & 0 & 0 & -1 \\
+1 & 0 & 0 & 0 & +1 & 0 \\
0 & -1 & 0 & -1 & 0 & +1 \\
0 & 0 & +1 & 0 & -1 & 0
\end{array}\right) .
$$

Now, the number of perfect matchings can be computed as the Pfaffian of $\boldsymbol{K}$ using (2.6): Pf $\boldsymbol{K}=$ $\sqrt{\operatorname{det} \boldsymbol{K}}=\sqrt{9}=3$. This agrees with the total number of perfect matchings shown in Figures 2.5(b-d).

Kasteleyn [59], Temperley and Fisher [124] independently showed that the dimer problem on the $2 n \times 2 m$ lattice has a closed-form solution. A Pfaffian orientation of a lattice can be achieved by orienting all horizontal edges in the same direction, while orienting vertical edges in alternating directions as shown in Figure 2.9. Computing the Pfaffian of the corresponding matrix we find that the number of dimer coverings is

$$
\begin{equation*}
\prod_{i=1}^{n} \prod_{k=1}^{m}\left(4 \cos ^{2} \frac{\pi i}{2 n+1}+4 \cos ^{2} \frac{\pi k}{2 m+1}\right) \tag{2.12}
\end{equation*}
$$



Figure 2.9: An example of a Pfaffian orientation of a lattice.

## Chapter 3

## Review of the Ising Model

This chapter introduces the Ising model - a classical model in statistical physics. We give a description of the model and provide some motivation for its use. This is followed by a brief history of the main developments associated with the model. We then focus on two key problems in the Ising model: computation of the partition function (Section 3.2) and the ground state (Section 3.3).

### 3.1 Introduction

Consider a large lake. As winter approaches the air becomes cooler and so does the temperature of the water. Then at around $0^{\circ}$ degrees the surface of the lake freezes. This occurs because the water molecules change their state from liquid to solid, known as a phase transition. Amazingly the lake freezes across the whole surface at roughly the same time. How do the molecules know at which temperature to freeze and how do they "communicate" with each other to achieve that feat? The same phenomenon is observed in other systems, such as when a cooling lump of iron suddenly becomes magnetic at a certain temperature.

In 1925, the German physicist Ernst Ising introduced a simple graphical model, called the Ising model [53], that can model such phenomena. The Ising model can be formulated on any graph as follows: consider an undirected graph $G=(\mathcal{V}, \mathcal{E})$, where $\mathcal{V}=\{1, \ldots, N\}$ is a set of $N$ sites, and $\mathcal{E}$ is a set of edges representing the interactions between these sites. Every site $i$ has a corresponding spin variable $s_{i}$. These spins are binary-valued ${ }^{3}$, taking values +1 for "up" or -1 for "down". Two spins $s_{i}$ and $s_{j}$ may interact with each other. The energy of such an interaction depends on whether the values of the participating spins are the same or different: it is given by $-J_{i j} s_{i} s_{j}$, where $J_{i j}$ is the strength of the interaction. The $J_{i j}$ are independent identically distributed random variables. If $J_{i j}>0$ the interaction is called ferromagnetic, if $J_{i j}<0$ the interaction is called antiferromagnetic, and otherwise $J_{i j}=0$ and the two spins do not interact. For each pair of interacting spins $s_{i}$ and $s_{j}\left(i . e ., J_{i j} \neq 0\right)$ there exists a corresponding edge $(i, j) \in \mathcal{E}$. The state of the model $s$ is an assignment of all $N$ variables $s_{i}, 1 \leq i \leq N$. The set of all possible configurations is $\Omega=\{-1,+1\}^{N}$. As well as pair-wise interactions there can also be an external field that affects each site $i$ with energy $-h_{i} s_{i}$. Thus in the general case the energy of a

[^2]configuration $s \in \Omega$ is given by the so-called Edwards-Anderson Hamiltonian [25]:
\[

$$
\begin{equation*}
\mathcal{H}(s):=-\sum_{(i, j) \in \mathcal{E}} J_{i j} s_{i} s_{j}-\sum_{i \in \mathcal{V}} h_{i} s_{i} \tag{3.1}
\end{equation*}
$$

\]

When $h_{i}=0 \forall i \in \mathcal{V}$ the system is said to have no external field (also called zero magnetic field condition), in which case the energy of the configuration $s$ becomes

$$
\begin{equation*}
\mathcal{H}(s)=-\sum_{(i, j) \in \mathcal{E}} J_{i j} s_{i} s_{j} \tag{3.2}
\end{equation*}
$$

Note that negating all the spins $s_{i}$ gives a different configuration $s^{\prime}$ whose energy is the same: $\mathcal{H}(s)=$ $\mathcal{H}\left(s^{\prime}\right)$. From now on, we will regard $\boldsymbol{s}$ and its negation $\boldsymbol{s}^{\prime}$ as a single state. The system prefers lower energy states, i.e., those $s$ that minimize $\mathcal{H}(s)$. Hence such states are more probable; the degree of preference depends on the temperature $T$ of the system. Formally, the probability that the system with the energy (3.2) is in configuration $s$ is given by

$$
\begin{equation*}
\mathbb{P}(\boldsymbol{s})=\frac{1}{Z} \exp \left(-\frac{\mathcal{H}(\boldsymbol{s})}{\kappa T}\right), \tag{3.3}
\end{equation*}
$$

where $\kappa$ is the Boltzmann constant. For the sake of simplicity we will define $\beta:=1 /(\kappa T)$. $Z$ is known as the partition function and is defined as

$$
\begin{equation*}
Z(\beta):=\sum_{s \in \Omega} \exp (-\beta \mathcal{H}(s)) \tag{3.4}
\end{equation*}
$$

The partition function plays the role of a normalizing constant, ensuring that the probabilities add up to one. This can be easily seen if we compute the total probability:

$$
\begin{equation*}
\sum_{s \in \Omega} \mathbb{P}(\boldsymbol{s})=\frac{1}{Z} \sum_{s \in \Omega} \exp (-\beta \mathcal{H}(\boldsymbol{s}))=\frac{1}{Z} Z=1 \tag{3.5}
\end{equation*}
$$

In Equation 3.3 as the temperature $T$ decreases the configurations with lower energy become more likely. In fact in the limit $T \rightarrow 0$ only the configurations of lowest energy have a non-zero probability of occurring. Hence an important task is to find such configurations $\boldsymbol{s}^{*}$ that minimize the energy of the system. Such a configuration is known as a ground state and is defined as

$$
\begin{equation*}
s^{*}:=\underset{s \in \Omega}{\operatorname{argmin}} \mathcal{H}(s) . \tag{3.6}
\end{equation*}
$$

In Equation 3.2, if $J_{i j}>0$ then having both $s_{i}$ and $s_{j}$ in the same state (i.e., $s_{i}=s_{j}$ ) can only decrease the energy. Therefore if $J_{i j}>0 \forall(i, j) \in \mathcal{E}$ then it is clear that $s^{*}$ is the configuration where all spins are in the same state. However the problem becomes far from trivial when $J_{i j}$ can be negative. The Ising model can be used to answer many other questions such as:

- What is the most likely spin at a given site?
- At what temperature does a phase transition occur?
- What is the correlation between spins at two given sites?

For some Ising models the above questions can be answered exactly, e.g., when the underlying graph is planar (see Sections 3.2 and 3.3). In the general case, however, exact methods are too slow and one must resort to approximations. A plethora of heuristic methods have been used to study the Ising model, for example: Monte Carlo simulation [55, 80, 120], Simulated Annealing [63], Parallel Tempering [49, 88], Extremal Optimisation [10] and genetic algorithms [48].

Although the Ising model can be formulated on any graph, often a 2-dimensional rectangular lattice is used. When the size of the model lattice is too small, the behavior of the spins at the boundary of the lattice may produce erratic results. For this reason, the model is often given periodic boundary conditions, i.e., nodes on one side of the lattice are connected to nodes on its opposite side. The resulting graph is a toroidal lattice.

### 3.1.1 Brief History

The original motivation for the model was the phenomenon of ferromagnetism. Iron is magnetic, so once it is magnetized it stays magnetized for a long time compared to any atomic time scale. Once the electron's spin was discovered, it became clear that magnetism is due to a large number of electrons spinning in the same direction. It was then natural to ask how the electrons all know which direction to spin, if they can only influence their neighbours. The Ising model was designed to investigate whether a large fraction of the electrons could be made to spin in the same direction using only local forces.

Interestingly, the Ising model was first proposed by Ernst Ising's research director Lenz [82] in 1920, but the model is more commonly attributed to Ising. In fact even Lenz himself has never made any attempt to claim credit for suggesting the model, and even his colleagues at Hamburg University were not aware of his contribution [13]. In his PhD thesis, Ising [52] solved the one-dimensional Ising model and showed that it does not exhibit a phase transition. On the basis of this result, he concluded that the Ising model does not have a phase transition in any dimension.

In 1936 Peierls proved Ising wrong, using a probabilistic argument to show that a phase transition can occur in the two-dimensional Ising model [100]. In 1941 Kramers and Wannier [74] showed that a matrix formulation of the model allows the partition function to be related to the largest eigenvalue of the matrix, and established the exact location of the phase transition - the Curie point. A breakthrough came in 1944 when Lars Onsager computed the partition function of the Ising model on a two-dimensional square lattice with no external field [97]. In 1952 Kac and Ward [56] attempted to provide a simpler alternative to Onsager's formula based on a combinatorial interpretation. Kac and Ward [56] viewed the partition function as a generating function for the problem of counting even subgraphs in an $n \times m$ lattice, which can be computed by finding the determinant of an appropriately constructed $4 m n \times 4 m n$ matrix. Their argument, however, was found to be incomplete, since some of the closed polygons were not being counted correctly. The clue to the correct approach was found by Feynman who conjectured a relation between functions of graphs and closed paths on lattices [45]. In 1960 Sherman [117] proved Feynman's conjecture, thus making the Kac-Ward method completely rigorous.

Also in 1960, Hurst and Green [50] used field theory techniques to arrive at a new method that combines the algebraic solution of Onsager [97] with the combinatorial solution of Kac and Ward [56], but is simpler than either. This was the first method that used the Pfaffian of a skew-symmetric matrix
to solve the Ising model. The Pfaffian method was developed further in the fundamental papers of Kasteleyn [59, 60] and Fisher [30, 31].

In 1982 Barahona [2] showed that finding the ground state of an Ising model defined on a square lattice with an external field is NP-hard. He also showed that finding the ground state of an Ising model defined on a two-level square lattice without an external field is NP-hard. Based on Barahona's work, in 2000 Istrail [54] proved that computing the partition function of a three-dimensional Ising model to be NP-complete. He explained that NP-completeness arises due to the non-planarity of the model involved, rather than its number of dimensions. Finally, Bertoni et al. [6] showed that the ground state of a threedimensional Ising model is not approximable, meaning that there is no polynomial-time algorithm that approximates the energy of the ground state within a certain ratio.

### 3.2 Previous Work on the Partition Function of the Ising Model

The partition function of an Ising model can be viewed as a sum over its spin configurations. It is well known that the partition function of a planar Ising model with no external field (3.4) can be found exactly by computing the Pfaffian of a particular matrix derived from the model graph $G$ [56]. The mapping to the Pfaffian is constructed in two steps: first the Ising partition function on $G$ is mapped to a weighted sum over even subgraphs in some relevant graph $G_{0}$, then the weighted sum over even subgraphs in $G_{0}$ is mapped to the dimer problem (Section 2.3), i.e., a perfect matching on decorated graph of $G_{0}$, which we call $G_{d}$. In the following sections we describe these mapping steps.

### 3.2.1 Ising Problem to Even Subgraphs

As mentioned earlier, the Ising partition function on $G$ can be mapped to a weighted sum over even subgraphs (Definition 22) in some relevant graph $G_{0}$. There are two main methods for obtaining the required mapping: in the first, $G_{0}$ is the primal model graph $G$ with transformed edge weights; in the second, $G_{0}$ is the dual graph $G^{*}$ with unmodified edge weights.

## Primal Graph Construction

For the sake of simplicity assume that the model graph $G$ is a $L \times L$ square lattice with $N:=L^{2}$ spins. We begin with an important Lemma:

Lemma $21 e^{ \pm x}=\cosh (x)(1 \pm \tanh (x)), \forall x \in \mathbb{R}$.

## Proof

$$
\begin{aligned}
\text { RHS } & =\cosh (x)(1 \pm \tanh (x))=\cosh (x) \pm \sinh (x) \\
& =\frac{e^{x}+e^{-x}}{2} \pm \frac{e^{x}-e^{-x}}{2}=e^{ \pm x}=\mathrm{LHS}
\end{aligned}
$$

Since $s_{i} s_{j}= \pm 1 \forall(i, j) \in \mathcal{E}$, we can apply Lemma 21 to the expression of the partition function (3.4):

$$
\begin{align*}
Z(\beta) & =\sum_{s \in \Omega} \prod_{(i, j) \in \mathcal{E}} e^{\beta J_{i j} s_{i} s_{j}} \\
& =\sum_{s \in \Omega} \prod_{(i, j) \in \mathcal{E}} \cosh \left(\beta J_{i j}\right)\left(1+s_{i} s_{j} \tanh \left(\beta J_{i j}\right)\right) \\
& =\sum_{s \in \Omega}\left[\prod_{(i, j) \in \mathcal{E}} \cosh \left(\beta J_{i j}\right) \prod_{(i, j) \in \mathcal{E}}\left(1+s_{i} s_{j} \tanh \left(\beta J_{i j}\right)\right)\right] \\
& =\prod_{(i, j) \in \mathcal{E}} \cosh \left(\beta J_{i j}\right)\left[\sum_{s \in \Omega} \prod_{(i, j) \in \mathcal{E}}\left(1+s_{i} s_{j} \tanh \left(\beta J_{i j}\right)\right)\right] \\
& \propto \sum_{s \in \Omega} \prod_{(i, j) \in \mathcal{E}}\left(1+s_{i} s_{j} \tanh \left(\beta J_{i j}\right)\right) \tag{3.7}
\end{align*}
$$

Thus we have transformed a sum over $e^{-\beta \mathcal{H}(s)}$ to a sum over polynomials in $\tanh (\beta \boldsymbol{J})$ [29, 106]. Expanding the product in (3.7) gives rise to $2^{2 N}$ terms of varying order in $\tanh (\beta \boldsymbol{J})$. A $k^{\text {th }}$-order term in $\tanh (\beta J)$ has a coefficient of the form $\left(s_{i_{1}} s_{j_{1}}\right) \ldots\left(s_{i_{k}} s_{j_{k}}\right)$, where each pair appears only once. There is a 1:1 mapping by which each $k^{\text {th }}$-order term in $\tanh (\beta \boldsymbol{J})$ is associated with the set of $k$ edges $\left\{\left(i_{1}, j_{1}\right), \ldots,\left(i_{k}, j_{k}\right)\right\}$. We know that

$$
\sum_{s= \pm 1} s^{n}= \begin{cases}2 & \text { if } n \text { is even }  \tag{3.8}\\ 0 & \text { if } n \text { is odd }\end{cases}
$$

Based on (3.8) we can see that only those terms where each $s_{i}$ appears an even number of times will contribute to $Z(\beta)$ in Equation 3.7, while all other terms will vanish. Diagrammatically this corresponds to a set of edges $\left\{\left(i_{1}, j_{1}\right), \ldots,\left(i_{k}, j_{k}\right)\right\}$ in which every node $i$ has an even number $(0,2$ or 4 in case of a rectangular grid) of incident edges. We call such a set of edges (and nodes) an even subgraph:

Definition 22 An even subgraph is a subgraph in which every node has an even degree.
Examples of even subgraphs are shown in Figures 3.1 and 3.2(a). While "even subgraph" is a term that is generally accepted in modern graph theory, many other terms have been used to describe the same concept: closed polygons [31, 56, 59, 60], closed subgraphs [61], quasi-cycles [3], Eulerian subgraphs [37] and loops [126].

To summarize, we can map the Ising partition function on $G$ to a weighted sum over even subgraphs in $G_{0}$. In this case $G_{0}$ is the original graph $G$ whose edge weights are $w_{0}(i, j):=\tanh \left(\beta J_{i j}\right)$ for edge $(i, j) \in \mathcal{E}_{0}$.

## Dual Graph Construction

In the primal graph construction there is no direct correspondence between individual even subgraphs and spin configurations. An alternative construction is to define $G_{0}$ to be the dual graph $G^{*}$ and use unmodified weights, i.e., $w_{0}(i, j):=-\beta J_{i j}$ for edge $(i, j) \in \mathcal{E}_{0}$. Every state $s$ of the Ising model induces a cut in $G$; this cut is an even subgraph of $G^{*}$ :

Definition 23 The cut $C$ of an Ising model $G(\mathcal{V}, \mathcal{E})$ induced by state $s \in\{-1,1\}^{|\mathcal{V}|}$ is the set $\mathcal{C}(\boldsymbol{s})=$ $\left\{(i, j) \in \mathcal{E}: s_{i} \neq s_{j}\right\}$.

Figure 3.1 shows a section of a rectangular lattice $G$ (white circles) with its corresponding dual graph $G^{*}$ (red disks). Edges with opposing spins (disagreement edges) correspond to edges of the cut $C(s)$, which are the edges of the even subgraph in $G^{*}$ (bold red).


Figure 3.1: A section of a rectangular lattice $G$ (white circles) with its corresponding dual graph $G^{*}$ (red disks). The spin configuration at each site induces a cut in $G$, which is an even subgraph in $G^{*}$ (bold edges).

Most work on the Ising partition function before the 1980s was primarily concerned with regular lattices, and it so happens that the dual graph of such lattices is the original lattice. Therefore, it is not clear whether prior work, such as [61], used the dual construction. The first time the dual graph is mentioned in relation to the Ising partition function is in Barahona [2]. However, in that work the correspondence is between sets of unsatisfied edges in $G$ and paths in $G^{*}$ joining pairs of frustrated faces of $G$ (Section 3.3.2). It seems that Globerson and Jaakkola [38] were the first to explicitly use this dual construction for the Ising partition function (Section 4.3.7). We also use this construction in our work (see Chapter 5).

### 3.2.2 Even Subgraphs to Perfect Matchings

The computation on even subgraphs in $G_{0}$ can be mapped to the dimer problem on a decorated graph $G_{d}$. This mapping can be either direct or complementary, depending on whether edges of an even subgraph correspond to matched or unmatched edges of the perfect matching of $G_{d}$, respectively. Past approaches also differ in the construction of the decorated graph $G_{d}$. The decorated graph $G_{d}$ is obtained by replacing every node of $G_{0}$ with a certain subgraph, called a city.

## Direct Mapping

In the direct mapping, edges of an even subgraph in $G_{0}\left(\mathcal{V}_{0}, \mathcal{E}_{0}\right)$ correspond to matched edges of the perfect matching of $G_{d}\left(\mathcal{V}_{d}, \mathcal{E}_{d}\right)$. Recall that an even subgraph has an even number of edges incident on every node of $G_{0}$. This implies that each city in $G_{d}$ must have an even number of nodes. With this in mind, Kasteleyn $[60,61]$ replaces every node $i \in \mathcal{V}_{0}$ with a Kasteleyn city:

Definition 24 A Kasteleyn city of a node $i \in \mathcal{V}_{0}$ with degree $d$ is a d-clique if $d$ is even and $a(d+1)$ clique if $d$ is odd. This clique contains all original edges of $G_{0}$, as well as the added edges that are internal to the clique.

To simplify the description of the mapping, we assume that $G_{0}$ is a 4-connected rectangular lattice. In this case, it is sufficient to replace every node $i \in \mathcal{V}_{0}$ with a 4-clique. The construction of $G_{d}$ can be seen in Figure 3.2(b). We must add extra "dummy" nodes (black disks) to ensure that cities have an even number of nodes; these nodes do not have an incident original edge.

The direct mapping from even subgraphs in $G_{0}$ to dimer coverings in $G_{d}$ is done as follows. With each edge of an even subgraph in $G_{0}$ (Figure 3.2(a)) associate a dimer on the original edge in $G_{d}$ (bold dimers in Figure 3.2(b)). The remaining unmatched nodes in $G_{d}$ come in pairs, as there are 4 nodes per clique with 2 nodes already matched earlier. These pairs of nodes can be matched via dimers on the internal edges of $G_{d}$. If 4 edges of an even subgraph meet (e.g., node 6 in Figure 3.2(a)), then there is only one way of mapping it to a dimer configuration in $G_{\boldsymbol{d}}$ (Figure 3.3(a)). This 1:1 correspondence also holds where 2 edges of an even subgraph meet (Figure 3.3(b)). However, if a node in $G_{0}$ has no incident edges of an even subgraph, such as the isolated nodes 1, 11 and 12 in Figure 3.2(a), then there are 3 ways to map it to a dimer configuration (Figure 3.3(c)).

(a)

(b)

(c)

Figure 3.2: (a) An even subgraph (solid edges) of a rectangular lattice $G_{0}$. (b) The corresponding dimer covering (solid edges) of the decorated graph $G_{d}$ derived from $G_{0}$ with a direct mapping to the even subgraph in (a). Edges of the even subgraph are shown as bold dimers, while dummy nodes are shown as black disks. (c) The corresponding dimer covering (solid edges) of the decorated graph $G_{d}$ with a complementary mapping to the even subgraph in (a).


Figure 3.3: Correspondence between even subgraphs in $G_{0}$ (solid edges, left) and dimer configurations in Kasteleyn's decorated graph $G_{d}$ (right) for nodes incident to (a) 4, (b) 2 and (c) 0 edges of an even subgraph. The correspondence is $1: 1$ for (a) and (b), but 1:3 for (c). Edges added to $G_{d}$ are internal to the cliques, while the remaining edges in $G_{d}$ are the original edges from $G_{0}$.

## Complementary Mapping

The direct mapping requires the addition of dummy nodes, which makes it difficult to generalize to arbitrary graphs. The alternative is to use a complementary mapping, where edges of an even subgraph in $G_{0}\left(\mathcal{V}_{0}, \mathcal{E}_{0}\right)$ correspond to unmatched edges of the perfect matching of $G_{d}\left(\mathcal{V}_{d}, \mathcal{E}_{d}\right)$. One possible $G_{d}$ with a complementary mapping is shown in Figure 3.2(c). Fisher [31] was the first to use a complementary mapping, aiming to simplify Kasteleyn's construction by satisfying the following two conditions:

- All even subgraphs of $G_{0}$ are in 1:1 correspondence with dimer coverings of $G_{d}$.
- $G_{d}$ is a planar graph, so it has a simple Pfaffian orientation described in Section 2.3.4.

Fisher [31] transforms $G_{0}\left(\mathcal{V}_{0}, \mathcal{E}_{0}\right)$ into a decorated graph $G_{d}\left(\mathcal{V}_{d}, \mathcal{E}_{d}\right)$ by replacing every node $i \in \mathcal{V}_{0}$ with a Fisher city. The construction is the following:

1. If $i \in \mathcal{V}_{0}$ has degree $d>3$ then replace it with a chain of $(d-2)$ vertices of degree 3 , as shown in Figure 3.4(b); otherwise $(d \leq 3)$ do nothing.
2. Replace every node with a $d$-clique, where $d$ is the degree of the node, as shown in Figure 3.4(c).

Fisher's city contains all the original edges of $G_{0}$, as well as added edges that are internal to the cliques. This construction plane triangulates $G_{0}$, which we use in our construction (see Section 5.3.1). The complementary $1: 1$ correspondence is constructed as follows: with the presence of an even subgraph edge in $G_{0}$, associate the absence of a dimer on the corresponding original edge of $G_{d}$. Similarly, with the absence of an even subgraph edge associate the presence of a dimer. Figure 3.5 shows an example
of the correspondence for a vertex with degree $d=4$ (for vertices with $d<4$, see [31]). Notice that $G_{d}$ is planar and that the allowed dimer configurations on the internal bonds of $G_{d}$ are always unique, in contrast to Kasteleyn's construction [60].


Figure 3.4: Fisher's construction of the decorated graph $G_{d}$ (c) from the original graph $G_{0}$ (a). We show the intermediate step (b) for illustrative purposes only.

## Construction of the Kasteleyn Matrix

We now describe how to construct the Kasteleyn matrix to solve the dimer problem on $G_{d}$. Let $w(i, j)$ be the weight of edge $(i, j) \in \mathcal{E}_{d}$ if it is an original edge, otherwise (it is an internal edge) set its weight to $w(i, j):=0$. We now give $\mathcal{E}_{d}$ a Pfaffian orientation $\mathcal{E}_{d}^{\prime}$ and use it to obtain a skew-symmetric Kasteleyn matrix $\boldsymbol{K}$, whose Pfaffian is equal to the Ising partition function of $G$ [61]. $\boldsymbol{K}$ is defined as:

$$
\boldsymbol{K}_{i j}:= \begin{cases}+e^{w(i, j)} & \text { if }(i, j) \in \mathcal{E}_{d}^{\prime}  \tag{3.9}\\ -e^{w(i, j)} & \text { if }(j, i) \in \mathcal{E}_{d}^{\prime} \\ 0 & \text { otherwise }\end{cases}
$$

If $G_{d}$ is constructed using Kasteleyn cities then the correspondence between even subgraphs in $G_{0}$ and perfect matchings in $G_{d}$ is a mixture of $1: 1$ and 1:3 (see Figure 3.3). Furthermore, $G_{d}$ is not planar, meaning that we cannot solve the dimer problem on $G_{d}$ directly via the Pfaffian method (Section 2.3.4). Surprisingly however, the combination of non-planarity and lack of $1: 1$ correspondence for isolated nodes in $G_{0}$ does allow us to apply the Pfaffian method successfully. To obtain a Pfaffian orientation of $G_{d}$ we give a clockwise-odd orientation to the faces of $G_{d}$ that contain original edges. We then orient the remaining internal edges such that the 3 possible dimer configurations in Figure 3.3(c) are counted correctly as a single configuration (see [31] for more details). The oriented graph for a rectangular grid is shown in Figure 3.6(b).

### 3.2.3 Non-Planar Ising Models

Kasteleyn [59] also applied his technique to non-planar graphs, specifically a torus modeled as a rectangular lattice with periodic boundary conditions. The key problem with such graphs is that $|\boldsymbol{K}|$ attaches a








Figure 3.5: Correspondence between even subgraphs (solid edges) at a vertex of degree 4 in $G_{0}$ (top) and dimer configurations (bold edges) in Fisher's decorated graph $G_{d}$ (bottom) (adapted from [31]).


Figure 3.6: (a) A rectangular lattice $G_{0}$. (b) Pfaffian orientation of the corresponding Kasteleyn's decorated graph $G_{d}$ (adapted from [61]).
negative sign to even subgraphs that wind around the torus (i.e., cross its periodic boundary, as the blue and red cycles do in Figure 7.2) an odd number of times. This means that the corresponding perfect matchings are counted incorrectly. Kasteleyn addressed this problem for a toroidal lattice by calculating
the partition function as

$$
\begin{equation*}
Z=\frac{1}{2}[-\operatorname{Pf} \boldsymbol{K}+\operatorname{Pf} h(\boldsymbol{K})+\operatorname{Pf} v(\boldsymbol{K})+\operatorname{Pf} h(v(\boldsymbol{K}))], \tag{3.10}
\end{equation*}
$$

where $h(\cdot)$ and $v(\cdot)$ switch the sign of entries in $\boldsymbol{K}$, corresponding to edges crossing the horizontal and vertical periodic boundary, respectively [59, 102]. Kasteleyn realized that Equation 3.10 could be generalized to graphs of arbitrary genus $g$, but deemed this "practically useless" due to the need for evaluating $4^{g}$ determinants [61, p. 98]. This must be seen in light of the lack of computing power in the 1960s, and a wish to calculate partition functions for infinite lattices composed of non-planar units, for which $g$ is unbounded.

Note that $h(\cdot)$ and $v(\cdot)$ preserve the clockwise odd orientation of any underlying graph, as Lemma 18 requires. This raises a troubling question: what distinguishes $\boldsymbol{K}$, with its negative sign in (3.10), from $h(\boldsymbol{K})$ or $v(\boldsymbol{K})$, which could have arisen from the construction of a different clockwise odd orientation?

Indeed (3.10) is necessary but not sufficient to correctly calculate the partition function for toroidal lattices. What is missing is a lateral parity condition. The missing condition holds tautologically for Kasteleyn's particular construction of a toroidal lattice. A correct generalisation to arbitrary non-planar graphs was recently provided by Galluccio and Loebl [36]. Fast algorithms for the computation of the partition function in toroidal lattices and in arbitrary non-planar graphs (with bounded genus), were given in [107] and [37], respectively.

### 3.3 Previous Work on the Ground States of the Ising Model

Most previous methods compute the ground state of the Ising model by finding the minimum-weight perfect matching on a related dual graph. For a long time the computation of the ground state and the computation of the partition function were treated as two separate problems; Barahona [2] apparently was the first to use a single construction for both.

### 3.3.1 Work by Bieche et al

Bieche et al. [9] present an algorithm for the rectangular lattice; here we extend their method to general planar graphs. Consider a graph $G(\mathcal{V}, \mathcal{E})$ where each vertex $i$ is assigned a spin $s_{i}= \pm 1$ and each edge $e=(i, j)$ is assigned a weight $w(e):=-J_{i j} s_{i} s_{j}$, where $J_{i j}$ is a real number representing the intensity of the interaction. An edge $e$ is said to be satisfied if it attains its minimal possible weight, i.e., when $w(e)=-\left|J_{i j}\right|$. Based on $J_{i j}$ one can define the set of frustrated cycles:

Definition $25 A$ frustrated cycle $O \subseteq \mathcal{E}$ of a $\operatorname{graph} G(\mathcal{V}, \mathcal{E})$ with edge weights $w$ is a simple cycle (Definition 2) whose product of edge weights is negative, i.e., $\prod_{(i, j) \in O} w(i, j)<0$.

In other words, a cycle is frustrated if it has no edges with zero weight and an odd number of edges with a negative weight. This means that there is no spin configuration that will satisfy all its edges. From this notion we can also define a frustrated face (called frustrated plaquette in [9]) to be a frustrated cycle that is also a face in a given embedding. It seems that the term "frustated face" was introduced by Toulouse [129]

The method of Bieche et al. [9] computes the ground state by pairing up the frustrated faces of $G$, in such a way that the sum of edge weights along the paths connecting each pair is minimal. This method was first used by Vannimenus and Toulouse [133] who carried out the computation manually. Let $s_{i j} \in\{-1,1\}$ be the state of edge $(i, j) \in \mathcal{E}$. We begin by initializing $s_{i j}:=1$ (agreement edge) if $J_{i j} \geq 0$ and $s_{i j}:=-1$ (disagreement edge) otherwise. If there are no frustrated cycles then every edge will be satisfied and we have the ground state, as shown in Figure 3.7. If this is not the case then we must resolve all frustrated cycles.


Figure 3.7: A rectangular lattice with no frustrated faces. Edges with non-negative interaction strength are solid, while those with negative interaction strength are dashed. The spin configuration of the ground state is shown for each site.

For convenience we describe the algorithm via the dual graph $G^{*}\left(\mathcal{F}, \mathcal{E}^{*}\right)$. To resolve a frustrated face $f$, we need to invert the state of one of its incident edges $e=\left(f, f^{\prime}\right)$, which changes $\mathcal{H}(s)$ by $2 J_{e} s_{e}$. However by doing so, we also change the frustration status of face $f^{\prime}$. If $f^{\prime}$ was already frustrated then we have resolved both $f$ and $f^{\prime}$ in one shot. Otherwise we have to keep inverting edge states of $f^{\prime}$ until we find another frustrated face to match it up with. We are thus looking for a path of minimum energy between two frustrated faces. Let $\Pi\left(f_{i}, f_{j}\right) \subseteq \mathcal{E}^{*}$ be such a path for two frustrated faces $f_{i}$ and $f_{j}$. The weight of this path is

$$
\begin{equation*}
w^{*}\left(\Pi\left(f_{i}, f_{j}\right)\right):=\sum_{e \in \Pi\left(f_{i}, f_{j}\right)}-|w(e)| \tag{3.11}
\end{equation*}
$$

Bieche et al. [9] solve this by finding the minimum perfect matching $\mathcal{M}$ (Definition 14) of the complete graph over frustrated faces $G_{f}\left(\mathcal{F}_{f}, \mathcal{E}_{f}\right)$. $\mathcal{F}_{f}$ is the set of frustrated faces in $\mathcal{F}$, and $\mathcal{E}_{f}=\mathcal{F}_{f} \times \mathcal{F}_{f}$; the weight of edge $\left(f_{i}, f_{j}\right) \in \mathcal{E}_{f}$ is $w^{*}\left(\Pi\left(f_{i}, f_{j}\right)\right)$. Note that this perfect matching will always exist, since the number of frustrated faces will be even:

## Lemma 26 The number of frustrated faces in any planar Ising model is even.

Proof Suppose we have a planar Ising model with $s_{i j}:=1 \forall(i, j) \in \mathcal{E}$. This model has 0 frustrated faces. Inverting the state of an edge negates that edge's weight, which changes the frustration status of exactly two faces, since in a planar graph each edge belongs to two faces. From the given edge state we can reach any other edge state via a sequence of edge state inversions.

To obtain the ground state, for each edge $\left(f_{i}, f_{j}\right) \in \mathcal{M}$ we invert the state of edges in $G$ crossed by the path $\Pi\left(f_{i}, f_{j}\right)$. The energy of the ground state is given by

$$
\begin{equation*}
\mathcal{H}(s)=-\sum_{(i, j) \in \mathcal{E}}\left|J_{i j}\right|+2 w(\mathcal{M}) \tag{3.12}
\end{equation*}
$$

where $w(\mathcal{M})$ is the weight of the perfect matching $\mathcal{M}$. For a complete example consider Figure 3.8. The edges whose state is inverted are those crossed by the bold dual edges in Figure 3.8.


Figure 3.8: Finding the ground state of a planar graph with frustrated faces. Model graph $G$ with white nodes, where dashed edges have disagreement cost -2 and solid edges have cost +1 . Dual frustration graph $G_{f}$ with blue nodes, with path weights indicated and the minimum-weight perfect matching $M$ is shown in bold.

The approach of Bieche et al. [9] suffers from two major drawbacks. Firstly, the computation of all paths of shortest weight in $G^{*}$ can be prohibitively expensive. Secondly, $G_{f}$ can be expensive to store as it is a fully connected graph. Currently treatable rectangular lattices are limited to around $300 \times 300$ nodes.

### 3.3.2 Work by Barahona

Barahona [2] constructs the dual graph $G^{*}\left(\mathcal{F}, \mathcal{E}^{*}\right)$ whose edges have weight $w^{*}(e):=-|w(e)|$. Since there is a bijection between $\mathcal{E}$ and $\mathcal{E}^{*}$ we extend the notion of satisfiability: an edge $e^{*} \in \mathcal{E}^{*}$ is satisfied if its corresponding edge in $\mathcal{E}$ is satisfied, otherwise it is unsatisfied. A node $f_{i} \in \mathcal{F}$ is called odd if $f_{i}$ represents a frustrated face, otherwise it is called even. Odd (even) nodes must be adjacent to an odd (even) number of unsatisfied edges.

Barahona [2] then constructs a graph $\tilde{G}$ such that there is a $1: 1$ correspondence between configurations of unsatisfied edges in $G^{*}$ and perfect matchings in $\tilde{G}$. To obtain $\tilde{G}$ we first expand $G^{*}$ as shown in Figure 3.4(b). The next stage is to transform all odd vertices as shown in Figure 3.9 (top) and all even vertices as shown in Figure 3.9 (bottom). All the original edges of $G^{*}$ retain their weight, while the added edges are given zero weight.

Now each configuration of unsatisfied edges in $G^{*}$ corresponds to a perfect matching of $\tilde{G}$, as shown in Figure 3.9. The edges of $G^{*}$ that are in the minimum-weight perfect matching $\mathcal{M}$ of $\tilde{G}$ correspond to


Figure 3.9: Correspondence between unsatisfied edges (dashed) in $G^{*}$ and perfect matchings (bold) in $\tilde{G}$ for odd vertices (a and b) and even vertices (c and d) (adapted from [2]).
paths of minimum weight between frustrated faces. To obtain the ground state, we follow Bieche et al. [9] and invert the state of edges in $G$ crossed by those paths.

The approach of Barahona [2] is preferable to that of Bieche et al. [9] for larger graphs $G$. This is because the size of $\tilde{G}$ grows more slowly than the size of $G_{f}$, and hence the computation of a minimumweight perfect matching on $\tilde{G}$ is faster.

### 3.3.3 Work by Thomas and Middleton

Thomas and Middleton [126] use Kasteleyn cities (Definition 24) to decorate the dual graph $G^{*}$ of the rectangular lattice $G$. In contrast to Barahona's construction, the topology of the resulting $G_{d}^{*}$ does not depend on the weights of $G$. The original edges of $G$ retain their weight in $G_{d}^{*}$, while newly introduced edges are given a weight of 0 . The minimum-weight perfect matching $\mathcal{M}$ of $G_{d}^{*}$ has a direct mapping (Section 3.2.2) to the even subgraph in $G^{*}$ with the smallest total weight (see Figure 3.2). Edges of the even subgraph in $G^{*}$ correspond to the set of disagreement edges in $G$ (just rotated by $90^{\circ}$ ), i.e., a minimum-weight cut of $G$. We can now use Algorithm 1 to obtain the ground state from this minimumweight cut. In experiments, Thomas and Middleton [126] found this technique to be about 3 times faster than the method of Bieche et al. [9], with running times similar to those of the method of Barahona [2].

## Chapter 4

## Review of Graphical Models

This chapter reviews probabilistic graphical models, which are necessary for understanding the contribution of this thesis. Here we predominantly focus on undirected graphical models. The expression "graphical model" can mean several things, but the meaning that we use here is the one used in the statistical machine learning community. In particular, it is a model defined by a graph whose nodes represent random variables and whose edges represent the possible dependencies between those random variables. In Section 4.3 we describe the current state-of-the-art inference methods for undirected graphical models.

### 4.1 Introduction

In recent years graphical models have become an invaluable tool in machine learning. Graphical models have been used with great success in a number of problem domains, ranging from bioinformatics and natural language processing to computer vision. In computer vision, graphical models have been applied to image segmentation and object identification, boundary detection, image restoration, texture modelling, graph matching, stereo reconstruction, digital photomontage, and video segmentation.

A graphical model defines a family of probability distributions that factorize according to an underlying model graph. The nodes of the graph correspond to random variables, while its edges represent statistical dependencies between these variables. Since the probability distribution can be over a large number of variables, the key idea is to represent it as a product over some local functions (potentials) each of which depends only on a few of these variables. This allows one to efficiently perform inference: compute various statistical quantities such as joint, marginal and conditional probability distributions, as well as maximum a posteriori states. The underlying graph can be directed or undirected. When it is directed, the model is often called a Belief Network or Bayesian Network, otherwise it is generally called a Markov Random Field (MRF). In this thesis we are primarily interested in undirected graphical models.

Graphical models can be parameterised by computing the MRF's potentials as inner product between parameters and features. Optimal parameters can be estimated from data ("learned") according to a number of criteria, such as regularized maximum likelihood or maximum margin. Once the parameters have been learned the MRF can be used to make predictions on unseen data.

### 4.2 Undirected Graphical Models

An undirected graphical model or MRF is defined over an undirected graph $G(\mathcal{V}, \mathcal{E})$ where $\mathcal{V}=$ $\{1,2, \ldots n\}$ is the set of nodes and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges in $G$. The MRF observes a set of real-valued input random variables $\boldsymbol{X}$ (features) with domain $\mathcal{X}$ and models a set of output random variables $\boldsymbol{Y}$ with domain $\boldsymbol{y} . \boldsymbol{y}$ can be either continuous or discrete, although in this thesis we will assume that it is always discrete, so $\mathcal{Y}=\{1, \ldots, m\}$. In particular, to each node $i \in \mathcal{V}$ we associate an output random variable $Y_{i}$, that takes values $y_{i} \in \mathcal{Y}$.

In general, we aim to model the joint probability distribution $\mathbb{P}(\boldsymbol{Y}=\boldsymbol{y}, \boldsymbol{X}=\boldsymbol{x})$, although sometimes it is convenient to model the conditional probability distribution $\mathbb{P}(\boldsymbol{Y}=\boldsymbol{y} \mid \boldsymbol{X}=\boldsymbol{x})$, like in the case of Conditional Random Fields (CRFs)[79]. For simplicity, we assume that the inputs $\boldsymbol{X}$ are always given and so omit them in all subsequent formulae. Furthermore we will write the joint probability distribution $\mathbb{P}(\boldsymbol{Y}=\boldsymbol{y})$ as $\mathbb{P}(\boldsymbol{y})$, and $\mathbb{P}\left(y_{i}\right)$ as a shorthand for $\mathbb{P}\left(Y_{i}=y_{i}\right)$. Let $\boldsymbol{Y}_{A}$ denote the vector of output random variables associated with the set of nodes $A \subseteq \mathcal{V}$ and let $\boldsymbol{y}_{A}$ be a particular realization of $\boldsymbol{Y}_{A}$. The marginal probability distribution for $\boldsymbol{Y}_{A}$ is computed as:

$$
\begin{equation*}
\mathbb{P}\left(\boldsymbol{y}_{A}\right)=\sum_{\boldsymbol{y} \backslash \boldsymbol{y}_{A}} \mathbb{P}(\boldsymbol{y}) . \tag{4.1}
\end{equation*}
$$

The maximum a posteriori (MAP) assignment $y^{*}$ is the state that maximizes the joint probability distribution:

$$
\begin{equation*}
y^{*}:=\underset{y}{\operatorname{argmax}} \mathbb{P}(y) . \tag{4.2}
\end{equation*}
$$

### 4.2.1 Conditional Independence Property

Conditional independence is an important concept for probability distributions over multiple variables, and is the building block for efficient inference in graphical models. Let $A, B$ and $C$ be subsets of nodes in $\mathcal{V}$. A subset of random variables $\boldsymbol{Y}_{A}$ is said to be conditionally independent from the subset $\boldsymbol{Y}_{C}$ given $\boldsymbol{Y}_{B}$, iff given that $\boldsymbol{Y}_{B}$ is known the knowledge of $\boldsymbol{Y}_{C}$ does not add any information to the knowledge of $\boldsymbol{Y}_{A}$. More formally:

Definition $27 \boldsymbol{Y}_{A}$ is conditionally independent of $\boldsymbol{Y}_{C}$ given $\boldsymbol{Y}_{B}$ iff $\mathbb{P}\left(\boldsymbol{y}_{A} \mid \boldsymbol{y}_{B}, \boldsymbol{y}_{C}\right)=\mathbb{P}\left(\boldsymbol{y}_{A} \mid \boldsymbol{y}_{B}\right) \forall \boldsymbol{y}_{A}, \boldsymbol{y}_{B}, \boldsymbol{y}_{C}$. We will write this as $\left(\boldsymbol{Y}_{A} \Perp \boldsymbol{Y}_{C}\right) \mid \boldsymbol{Y}_{B}$.

As an example, consider a football tournament. Given that we know the current league table, the knowledge of previous match outcomes does not help us in determining the current ranking of a team. So we have (team ranking $\Perp$ match outcomes) | league table. If $\left(\boldsymbol{Y}_{A} \Perp \boldsymbol{Y}_{C}\right) \mid \boldsymbol{Y}_{B}$ then we have $\mathbb{P}\left(\boldsymbol{y}_{A}, \boldsymbol{y}_{C} \mid \boldsymbol{y}_{B}\right)=\mathbb{P}\left(\boldsymbol{y}_{A} \mid \boldsymbol{y}_{B}\right) \mathbb{P}\left(\boldsymbol{y}_{C} \mid \boldsymbol{y}_{B}\right)$.

An important and elegant feature of graphical models is that conditional independence properties of the joint probability distribution can be read directly from the underlying graph. In directed graphical models these properties can be deduced by using the concept of d-separation [99]. In undirected graphical models, $\left(\boldsymbol{Y}_{A} \Perp \boldsymbol{Y}_{C}\right) \mid \boldsymbol{Y}_{B}$ is true iff every path from any node in $A$ to any node in $C$ passes through at least one node in $B$; all such paths are then said to be blocked by $B$.

### 4.2.2 Factorization Property

Let $\mathcal{K}$ be the set of maximal cliques (Definition 1) of the model graph G. A potential function $\psi_{C}$ of a clique $C \in \mathcal{K}$ is a function that associates a positive real number to every assignment of labels of nodes in $C$. The joint probability distribution can be written as a product of potential functions $\psi_{C}\left(\boldsymbol{y}_{C}\right)$ over the maximal cliques:

$$
\begin{equation*}
\mathbb{P}(\boldsymbol{y})=\frac{1}{Z} \prod_{C \in \mathcal{K}} \psi_{C}\left(\boldsymbol{y}_{C}\right), \quad \text { where } \quad Z:=\sum_{\boldsymbol{y}} \prod_{C \in \mathcal{K}} \psi_{C}\left(\boldsymbol{y}_{C}\right) \tag{4.3}
\end{equation*}
$$

is the partition function which ensures that $\mathbb{P}(\boldsymbol{y})$ is correctly normalized. The presence of this normalization constant is one of the major limitations of undirected graphical models, because its naive evaluation involves summing over $|\boldsymbol{y}|^{|\mathcal{V}|}$ states. The partition function is needed for maximum-likelihood parameter estimation, but not for computing the MAP states or maximum margin parameter estimation. The positivity of $\psi_{C}\left(\boldsymbol{y}_{C}\right)$ ensures that $\mathbb{P}(\boldsymbol{y})$ is non-negative. To make $\psi_{C}\left(\boldsymbol{y}_{C}\right)$ strictly positive it is often convenient to express it as an exponential function:

$$
\begin{equation*}
\psi_{C}\left(\boldsymbol{y}_{C}\right):=\exp \left(-E\left(\boldsymbol{y}_{C}\right)\right) \tag{4.4}
\end{equation*}
$$

where $E\left(\boldsymbol{y}_{C}\right)$ is the energy function of the clique $C$. The energy function of the joint probability distribution can be written as $E(\boldsymbol{y})=\sum_{C \in \mathcal{K}} E\left(\boldsymbol{y}_{C}\right)$, which allows us to conveniently rewrite Equation 4.3 as

$$
\begin{equation*}
\mathbb{P}(\boldsymbol{y})=\frac{1}{Z} \exp \left(-\sum_{C \in \mathcal{K}} E\left(\boldsymbol{y}_{C}\right)\right)=\frac{1}{Z} e^{-E(\boldsymbol{y})} \tag{4.5}
\end{equation*}
$$

Using the fact that $\mathbb{P}(\boldsymbol{y}) \propto \exp (-E(\boldsymbol{y}))$, we can rewrite the MAP definition (4.2), so that it matches the definition (3.6) of a ground state of an Ising model:

$$
\begin{equation*}
\boldsymbol{y}^{*}=\underset{\boldsymbol{y}}{\operatorname{argmax}} e^{-E(\boldsymbol{y})}=\underset{\boldsymbol{y}}{\operatorname{argmin}} E(\boldsymbol{y}) . \tag{4.6}
\end{equation*}
$$

The true power of graphical models comes from the fact that the conditional independence property implies the factorization property and vice-versa, as stated by the Hammersley-Clifford theorem [7, 44]:

Theorem 28 (Hammersley-Clifford Theorem) Any strictly positive probability distribution which is consistent with the conditional independence properties imposed by a graph $G$ also factorizes with respect to $G$.

This theorem tells us that any strictly positive probability distribution that respects the conditional independence properties of some graph $G$ can be factorized as a product over functions $\psi$, which are local in that each only involves the nodes of one maximal clique of $G$.

### 4.3 Inference Algorithms

For a general undirected graphical model the inference task is NP-hard. However, efficient algorithms exist for certain types of models. The complexity of inference is governed by the following factors: graph structure, form of the energy function and the number of output labels. In this section we will present the most popular algorithms used for inference in undirected graphical models. Variable elimination (Section 4.3.1) is an exact algorithm for small model graphs. Loopy Belief Propagation (Section 4.3.3) is a commonly-used approximate inference method, which is exact for trees. Its cousin, the Junction Tree algorithm (Section 4.3.4), is an exact inference method for "thin" graphs or graphs with small treewidth. Graph cuts (Section 4.3.5) are an exact inference method for models with submodular energy functions (Definition 31) and binary labels. Tree Reweighting (Section 4.3.6) is an approximate inference method that often outperforms standard Loopy Belief Propagation and has better convergence guarantees.

### 4.3.1 Variable Elimination

Suppose we want to evaluate the marginal probability distribution $\mathbb{P}\left(y_{i}\right)$ on a graph with $n$ nodes. We can find $\mathbb{P}\left(y_{i}\right)$ by marginalizing out all the random variables in the joint probability distribution $\mathbb{P}(\boldsymbol{y})$ that are not $y_{i}$ :

$$
\begin{align*}
\mathbb{P}\left(y_{i}\right) & =\frac{1}{Z} \sum_{y \backslash\left\{y_{i}\right\}} \mathbb{P}(\boldsymbol{y})  \tag{4.7}\\
& =\frac{1}{Z} \sum_{y_{1}} \cdots \sum_{y_{i-1}} \sum_{y_{i+1}} \cdots \sum_{y_{n}} \mathbb{P}\left(Y_{1}=y_{1}, \ldots, Y_{n}=y_{n}\right) . \tag{4.8}
\end{align*}
$$

Naively, we could fist evaluate the joint probability distribution for all possible configurations and then perform all the summations. If each variable has $m$ states then we will need $O\left(m^{n}\right)$ space to store the joint probability distribution, followed by $m^{n}$ summations.

We can obtain a much more efficient algorithm by exploiting the conditional independence properties of the graphical model. If we substitute Equation 4.3 for $\mathbb{P}(\boldsymbol{y})$, then we can reorder the summations and multiplications to minimize the total number of computations. The method is best demonstrated with an example. Consider the tree in Figure 4.1(a), for which we want to compute $\mathbb{P}\left(y_{5}\right)$. Based on the factorization properties we can write the computation as

$$
\begin{equation*}
\mathbb{P}\left(y_{5}\right)=\frac{1}{Z} \sum_{y_{1}} \sum_{y_{2}} \sum_{y_{3}} \sum_{y_{4}} \psi_{12}\left(y_{12}\right) \psi_{23}\left(y_{23}\right) \psi_{34}\left(y_{34}\right) \psi_{35}\left(y_{35}\right) . \tag{4.9}
\end{equation*}
$$

We notice that $y_{1}$ participates in just one term, so it makes sense to sum over $y_{1}$ first. After we have summed over (or eliminated) $y_{1}$, we can sum over the remaining variables. Again we will choose to eliminate a variable that occurs in the least number of terms. We continue this process until all variables have been eliminated. We can now rewrite Equation 4.9:

$$
\begin{equation*}
\mathbb{P}\left(y_{5}\right)=\frac{1}{Z} \sum_{y_{3}} \psi_{35}\left(y_{35}\right) \sum_{y_{4}} \psi_{34}\left(y_{34}\right) \sum_{y_{2}} \psi_{23}\left(y_{23}\right) \sum_{y_{1}} \psi_{12}\left(y_{12}\right) . \tag{4.10}
\end{equation*}
$$

In the above we are exploiting the fact that multiplication is distributive over addition, so in

$$
\begin{equation*}
a b+a c=a(b+c) \tag{4.11}
\end{equation*}
$$

the right-hand-side is preferable as it requires only 2 operations, compared to the 3 operations ( 2 multiplications and one addition) required by the left-hand-side. The resulting term of $\sum_{y_{i}} \psi_{i j}\left(y_{i j}\right)$ is a function of $j$, which for convenience we will write as $m_{i j}\left(y_{j}\right)$. The index $i$ refers to the variable being summed over, while the index $j$ refers to other variables appearing in the summand. Note that for trees, there is never more than two variables in the summand, since trees have a maximum clique size of 2 . We continue the derivation using the new notation:

$$
\begin{align*}
\mathbb{P}\left(y_{5}\right) & =\frac{1}{Z} \sum_{y_{3}} \psi_{35}\left(y_{35}\right) \sum_{y_{4}} \psi_{34}\left(y_{34}\right) \sum_{y_{2}} \psi_{23}\left(y_{23}\right) m_{12}\left(y_{2}\right)  \tag{4.12}\\
& =\frac{1}{Z} \sum_{y_{3}} \psi_{35}\left(y_{35}\right) \sum_{y_{4}} \psi_{34}\left(y_{34}\right) m_{23}\left(y_{3}\right) \\
& =\frac{1}{Z} \sum_{y_{3}} \psi_{35}\left(y_{35}\right) m_{23}\left(y_{3}\right) \sum_{y_{4}} \psi_{34}\left(y_{34}\right) \\
& =\frac{1}{Z} \sum_{y_{3}} \psi_{35}\left(y_{35}\right) m_{23}\left(y_{3}\right) m_{43}\left(y_{3}\right) \\
& =\frac{1}{Z} m_{35}\left(y_{5}\right) .
\end{align*}
$$

The final expression is a function of $y_{5}$ only, and thus is the desired marginal. This algorithm is usually called variable elimination as the variables are eliminated one by one.

### 4.3.2 Belief Propagation

In many problems we would like to compute more than a single marginal probability. For example in Figure 4.1(a) we might want to obtain both $\mathbb{P}\left(y_{4}\right)$ and $\mathbb{P}\left(y_{5}\right)$. We could compute each marginal separately using variable elimination, but wouldend up computing same $m_{i j}\left(y_{j}\right)$ twice, such as $m_{12}\left(y_{2}\right)$ and $m_{23}\left(y_{3}\right)$.

Belief Propagation (aka the sum-product algorithm) [99] is a method that efficiently computes all the possible $m_{i j}\left(y_{j}\right)$, thus allowing reuse of intermediate results for multiple computations. The intermediate term $m_{i j}\left(y_{j}\right)$ can be viewed as a "message" that is passed from node $i$ to node $j$ about what state node $j$ should be in, written as:

$$
\begin{equation*}
m_{i j}\left(y_{j}\right):=\sum_{Y_{i}} \psi_{i j}\left(y_{i j}\right) \prod_{k \in N(i) \backslash j} m_{k i}\left(y_{i}\right), \tag{4.13}
\end{equation*}
$$

where $N(i)$ is the set of neighbours of node $i$. Node $i$ can send a message to a neighbouring node $j$ once it has received all the incoming messages from its neighbours (except from node $j$ ). This set of incoming messages is computed as $\prod_{k \in N(i) \backslash j} m_{k i}\left(y_{i}\right)$ in Equation 4.13. Since leaf nodes have only one neighbour, they can start sending messages straight away. Belief propagation proceeds by propagating messages from the leaves to the root. The algorithm terminates once a message has been sent along each edge in both directions. It can be shown that belief propagation always converges to the optimum solution for


Figure 4.1: An undirected graphical model whose underlying graph is a tree. (a) Shows the intermediate messages that arise during variable elimination with elimination order: $y_{1}, y_{2}, y_{4}, y_{3}$. (b) Shows the set of all messages computed by the belief propagation algorithm.
trees. Figure 4.1(b) shows the set of messages needed to compute every individual marginal probability. Ignoring normalization terms, the final marginal probability at a node $i$ is given by the product of all its incoming messages:

$$
\begin{equation*}
\mathbb{P}\left(y_{i}\right) \propto \prod_{k \in N(i)} m_{k i}\left(y_{i}\right) . \tag{4.14}
\end{equation*}
$$

The sum-product algorithm can be easily modified to compute the probability of the MAP state. The idea is to use Equation 4.13 , replacing every instance of $\sum$ operator with the max operator. This works, since maximization commutes with products, just as summation does, i.e., $\max (a b, a c)=a \max (b, c)$. The resulting algorithm is called the max-product algorithm. In practice, products of many small probabilities can lead to numerical underflow problems. These problems can be avoided if one computes the logarithm of the joint probability distribution. Since the logarithm is a monotonic function, we can write $\operatorname{argmax}_{\boldsymbol{y}} \log (\mathbb{P}(\boldsymbol{y}))=\operatorname{argmax}_{\boldsymbol{y}} \mathbb{P}(\boldsymbol{y})$. This has the effect of replacing the products in the max-product algorithm with sums. This is called the max-sum algorithm.

### 4.3.3 Loopy Belief Propagation

It can be shown that belief propagation obtains the optimum solution for graphs containing a single loop. For general graphs, however, belief propagation is not guaranteed to converge to the optimum
solution. Loosely speaking, this is because general graphs may contain numerous loops and thus the same evidence can be passed around multiple times and mistaken for new evidence. Despite the lack of formal convergence guarantees, belief propagation is still widely used for general graphs, such as grids. In this case the method is called Loopy Belief Propagation [34, 137, 143].

From Section 4.3.2, we have seen that a node can pass messages once it has received messages from all its neighbours. Since there can be loops in the graph, we need to decide how to initiate the message passing algorithm. One possibility is to send an initial message given by the unit function across every edge in every direction. Now each node is in a position to start sending messages. We also have that $m_{i j}\left(y_{j}\right)$ can be computed multiple times for the same $i$ and $j$. Let us assume that $m_{i j}\left(y_{j}\right)$ sent at time $t$ replaces any previously sent $m_{i j}\left(y_{j}\right)$.

There are numerous ways to define a message passing schedule: e.g., flooding schedule passes a message across every edge in both directions at every time step, while a serial schedule passes one message at a time. Kschischang et al. [75] describes one such serial schedule. Node $i$ has a message pending to node $j$ if node $i$ has received any message since the last time it sent anything to node $j$. Only pending messages need to be transmitted, because all other messages would duplicate previously sent messages. The algorithm proceeds by sending all pending messages one by one. For trees this will terminate once a message has passed in each direction of each edge. For graphs with loops this might never terminate. Once the algorithm has terminated (due to convergence or otherwise), the final marginal of node $i$ is computed as a product of the most recently received messages to node $i$. Note that this marginal is only an approximation of the true marginal.

### 4.3.4 Junction Tree Algorithm

Belief propagation can be generalised to general graphs, giving an exact inference method known as the Junction Tree algorithm [81]. The idea is to construct a tree-based data structure, called a junction tree, whose nodes are cliques of the original graph. Belief propagation can then be used to compute exact results on this tree. There are several inference algorithms available [83]: Hugin, Shenoy-Shafer and Lauritzen-Spiegelhalter. However, all these algorithms perform exact inference and only differ in computational speed. Below we will describe the Shenoy-Shafer algorithm [114, 115, 116]. If the original graph is directed then we must first make it undirected via a method called moralization. The first step in constructing the junction tree involves transforming the model graph $G(\mathcal{V}, \mathcal{E})$ into a chordal $\operatorname{graph} G_{c}\left(\mathcal{V}, \mathcal{E}_{c}\right)$ :

Definition 29 A graph is chordal if each of its cycles of four or more nodes has a chord. A chord in a cycle is an edge joining two nodes that are not adjacent in that cycle.

We can construct $G_{c}$ by adding edges to $G$. An example of a chordal graph is shown in Figure 5.6(a). We can now construct the junction tree:

Definition 30 The junction tree $\mathcal{T}\left(\mathcal{K}, \mathcal{E}^{\prime}\right)$ of an undirected chordal graph $G_{c}$ is a tree, whose every node $i \in \mathcal{K}$ corresponds to a maximal clique of $G_{c}$. The edges $\mathcal{E}^{\prime}$ satisfy the running intersection property: for any edge $(i, k)=\left(C_{i}, C_{k}\right) \in \mathcal{E}^{\prime}$, the variables in the intersection $i \cap k$ are contained in every node of the tree on the unique path between $i$ and $k$.

The junction tree is maximal in the sense that the number of variables shared by its two connected nodes is maximal. Each maximal clique in $G_{c}$ is associated with exactly one node in $\mathcal{T}$. We set the potential of a clique $C$ to the product of potentials of its sub-cliques:

$$
\begin{equation*}
\psi_{C}\left(\boldsymbol{y}_{C}\right):=\prod_{C_{i} \subset C} \psi_{C_{i}}\left(\boldsymbol{y}_{C_{i}}\right) \tag{4.15}
\end{equation*}
$$

Instead of sending messages between nodes (4.13), we now send messages between cliques. A message from clique $C_{i}$ to $C_{j}$ is given as

$$
\begin{equation*}
m_{C_{i} C_{j}}\left(\boldsymbol{y}_{C_{i j}}\right):=\sum_{C_{i} \backslash C_{i j}} \psi_{C_{i}}\left(\boldsymbol{y}_{C_{i}}\right) \prod_{C_{k} \in N\left(C_{i}\right) \backslash C_{j}} m_{C_{k} C_{i}}\left(y_{C_{k i}}\right), \tag{4.16}
\end{equation*}
$$

where $C_{i j}:=C_{i} \cap C_{j}$ and $N\left(C_{i}\right)$ are the neighbours of clique $C_{i}$ in the junction tree. Similar to Equation 4.14, we can compute the marginal probability of a clique $C_{i}$ as a product of all its incoming messages:

$$
\begin{equation*}
\mathbb{P}\left(\boldsymbol{y}_{C_{i}}\right) \propto \prod_{C_{k} \in N\left(C_{i}\right)} m_{C_{k} C_{i}}\left(\boldsymbol{y}_{C_{k i}}\right) \tag{4.17}
\end{equation*}
$$

Finally, we can obtain the marginal probability for an individual node $y_{i}$ in clique $C$ :

$$
\begin{equation*}
\mathbb{P}\left(y_{i}\right)=\sum_{C \backslash i} \mathbb{P}\left(\boldsymbol{y}_{C}\right) \tag{4.18}
\end{equation*}
$$

Once again the problem of computing the probability of the MAP state can be solved by replacing $\Sigma$ with max. Since the junction tree algorithm works with joint probability distributions of nodes in each clique, its computational complexity is exponential in the number of nodes in its largest clique. This number is closely related to the treewidth of a graph $G$, which is defined as one less than the size of the maximal clique in any chordal graph containing $G$. Thus the complexity of the junction tree algorithm is exponential in the treewidth of its underlying graph. The treewidth of a tree is 1 , while the treewidth of an $n \times n$ grid is $n$.

### 4.3.5 Graph Cuts

The widespread use of graph cuts to find lowest-energy configurations, particularly in computer vision, was pioneered by Greig et al. [41]. Given an undirected graphical model with graph $G(\mathcal{V}, \mathcal{E})$, the algorithm aims to minimize the following real-valued energy function

$$
\begin{equation*}
E(\boldsymbol{y}):=\sum_{i \in \mathcal{V}} E_{i}\left(y_{i}\right)+\sum_{(i, j) \in \mathcal{E}} E_{i j}\left(y_{i}, y_{j}\right) \tag{4.19}
\end{equation*}
$$

where $\boldsymbol{y}=\left\{y_{1}, \ldots, y_{|\mathcal{V}|}\right\}$ is the set of node labels, $E_{i}\left(y_{i}\right)$ is the cost of assigning node $i$ to label $y_{i}$, and $E_{i j}\left(y_{i}, y_{j}\right)$ is the cost of assigning the adjacent nodes $i$ and $j$ to the labels $y_{i}$ and $y_{j}$, respectively. In general this problem is NP-hard, but is tractable for binary-labeled models with a submodular energy function $E$ :

Definition 31 A function $E$ of the form (4.19) is submodular iff each term $E_{i j}$ satisfies the inequality:
$E_{i j}(0,0)+E_{i j}(1,1) \leq E_{i j}(0,1)+E_{i j}(1,0)$.
For the purpose of the following discussion, we assume that the above conditions hold. The graph cut algorithm constructs a directed graph $G^{\prime}\left(\mathcal{V}^{\prime}, \mathcal{E}^{\prime}\right)$ such that the weight of its minimum s-t cut corresponds to the global minimum of $E[70]$.

Definition 32 An s-t cut $\mathcal{C}(S, T)$ of a directed graph $G^{\prime}\left(\mathcal{V}^{\prime}, \mathcal{E}^{\prime}\right)$ is a partition of vertices $\mathcal{V}^{\prime}$ into two disjoint sets $S$ and $T$, such that $s \in S$ and $t \in T$. The weight of the $s$ - $t$ cut $w(C(S, T))$ is the sum of weights of boundary edges $(p, q) \in \mathcal{E}^{\prime}$, where $p \in S$ and $q \in T$.

The minimum s-t cut is an s-t cut with the smallest weight. Every s-t cut partitions the nodes into two sets $S$ and $T$. Those nodes that are in $S$ are labeled 0 , while the remaining nodes are labeled 1 . Due to the theorem of Ford and Fulkerson [32], the minimum s-t cut is equivalent to the maximum flow from $s$ to $t$. By construction, all edges of $\mathcal{E}^{\prime}$ are non-negative, thus the maximum flow of $G^{\prime}$ can be computed in polynomial time, and hence also its minimum s-t cut. $G^{\prime}$ is constructed from $G$ as follows:

1. Add 2 extra terminal vertices $s$ (source) and $t\left(\right.$ sink). Thus we have $\mathcal{V}^{\prime}:=\{s, t, 1, \ldots,|\mathcal{V}|\}$.
2. For each node cost $E_{i}\left(y_{i}\right)$ we add an edge from node $i$ to one of the terminal nodes. If $E_{i}(1)>E_{i}(0)$ then we add the edge $(s, i)$ with the weight $E_{i}(1)-E_{i}(0)$. Otherwise, we add the edge $(i, t)$ with the weight $E_{i}(0)-E_{i}(1)$ (Figure 4.2(a)).
3. For each edge cost $E_{i j}\left(y_{i}, y_{j}\right)$ we add an edge $(i, j)$ with weight $E_{i j}(0,1)+E_{i j}(1,0)-E_{i j}(0,0)-$ $E_{i j}(1,1) \geq 0$. If $E_{i j}(1,0)>E_{i j}(0,0)$ then we add a weight of $E_{i j}(1,0)-E_{i j}(0,0)$ to edge $(s, i)$, otherwise we add a weight of $E_{i j}(0,0)-E_{i j}(1,0)$ to edge $(i, t)$. If $E_{i j}(1,0)>E_{i j}(1,1)$ then we add a weight of $E_{i j}(1,0)-E_{i j}(1,1)$ to edge $(j, t)$, otherwise we add a weight of $E_{i j}(1,1)-E_{i j}(1,0)$ to edge $(s, j)$. Figure 4.2(b) shows the graph for $E_{i j}\left(y_{i}, y_{j}\right)$ for the case when $E_{i j}(1,0)>E_{i j}(0,0)$ and $E_{i j}(1,0)>E_{i j}(1,1)$.

A more elaborate construction can give partial answers for non-submodular energy functions, i.e., graphs with some negative edge weights [43, 68, 105]. Furthermore, a sequence of expansion moves (energy minimizations in binary graphs) can efficiently yield an approximate answer for graphs with discrete but non-binary node labels [12].

### 4.3.6 Tree Reweighting

Given an undirected graphical model with graph $G(\mathcal{V}, \mathcal{E})$, Wainwright et al. [136] compute the MAP energy by maximizing a concave lower bound derived from a convex combination of tree-structured distributions. Let us assume that every $E\left(\boldsymbol{y}_{C}\right)$ decomposes into a product of parameters $\boldsymbol{\theta}_{C}$ and sufficient statistics $\phi_{C}\left(\boldsymbol{y}_{C}\right)$. We can now write $E(\boldsymbol{y}, \boldsymbol{\theta})=\langle\boldsymbol{\theta}, \phi(\boldsymbol{y})\rangle$, where $\boldsymbol{\theta}$ is the set of all parameters and $\phi(\boldsymbol{y})$ is the set of all sufficient statistics. The energy of the optimal configuration $\boldsymbol{y}^{*}$ can be written as $E\left(\boldsymbol{y}^{*}, \boldsymbol{\theta}\right)=\min _{\boldsymbol{\theta}} E(\boldsymbol{y}, \boldsymbol{\theta})$.

Let $\mathcal{T}$ be the set of spanning trees of $G$, such that each edge appears in at least one tree. Let $T \in \mathcal{T}$ be one of the spanning trees and $\theta(T)$ be its set of parameters. Let $\tilde{\boldsymbol{\theta}}$ be a concatenation of all such parameter sets $\boldsymbol{\theta}(T)$. We associate a weight $p(T)$ with every tree $T$, such that $p(T) \geq 0$ and


Figure 4.2: Graph cut construction of $G^{\prime}$ for node $i(\mathrm{a})$ and edge $(i, j)$ (b).
$\sum_{T \in \mathcal{T}} p(T)=1$. If we set $\boldsymbol{\theta}(T)$ such that $\sum_{T} p(T) \boldsymbol{\theta}(T)=\boldsymbol{\theta}$ then applying Jensen's inequality yields the lower bound

$$
\begin{equation*}
E\left(\boldsymbol{y}^{*}, \boldsymbol{\theta}\right) \geq \sum_{T \in \mathcal{T}} p(T) E\left(\boldsymbol{y}^{*}, \boldsymbol{\theta}(T)\right) \tag{4.20}
\end{equation*}
$$

This means that we can compute a lower bound on $E\left(\boldsymbol{y}^{*}, \boldsymbol{\theta}\right)$ via a convex sum of ground energies of individual trees, which can be found exactly. It turns out that equality in Equation 4.20 holds if the collection of trees share a common optimum. This is known as the tree agreement property. Wainwright et al. [136] aim to find the tightest lower bound of the form (4.20). The problem is first posed as a linear programming (LP) relaxation in the primal:

$$
\begin{array}{r}
\max _{\tilde{\boldsymbol{\theta}}} \sum_{T \in \mathcal{T}} p(T) E\left(\boldsymbol{y}^{*}, \boldsymbol{\theta}(T)\right)  \tag{4.21}\\
\text { such that } \sum_{T \in \mathcal{T}} p(T) \boldsymbol{\theta}(T)=\boldsymbol{\theta} .
\end{array}
$$

Specialized versions of this LP relaxation were formulated earlier in [16, 73, 108]. For a general distribution of trees, the primal problem (4.21) entails maximizing a sum of functions over all spanning trees of the graph, which can be a prohibitively large collection. Although this LP relaxation can be solved in polynomial time using interior-point methods, due to their large memory requirements the current state of the art software can only handle instances of a few hundred variables. For example, Meltzer et al. [90] report that the largest grid they could solve with such a method is only $39 \times 39$. Luckily the primal can be solved via its Lagrangian dual. This involves iterating over a relatively simple local consistency polytope $L O C A L(G)$. In particular, $\operatorname{LOCAL}(G)$ can be characterized by $O(m|\mathcal{V}|+$
$m^{2}|\mathcal{E}|$ ) constraints, where $m$ is the number of possible label assignments per node. A nice consequence of this duality is the fact that the optimal solution does not depend on the choice of trees and their probabilities, provided that each edge appears in $\mathcal{T}$.

For solving the dual problem on $\operatorname{LOCAL}(G)$ Wainwright et al. [136] propose two max-product treereweighted message passing algorithms (TRW): edge-based updates (TRW-E) and tree-based updates (TRW-T). TRW-T iterates between two phases: a) running max-product for all trees and b) performing averaging operations for all nodes and edges. TRW-E is similar, except that in phase a) it performs one parallel message update for all trees. The algorithms reduce to the ordinary max-product algorithm when applied to trees. Kolmogorov [66] shows that TRW-T and TRW-E can sometimes increase the lower bound and so do not guarantee convergence. He introduces a sequential update algorithm TRW$S$ which guarantees that the value of the lower bound does not decrease and converges to a solution that satisfies the weak tree agreement criterion (WTA) [66]. Although his experiments suggest that this can produce high-energy solutions for non-submodular energy functions. Kolmogorov [66] also introduces BP-S - ordinary max-product belief propagation algorithm with the same sequential schedule of updating messages as TRW-S.

Kolmogorov and Wainwright [69] show that tree-reweighted message passing is guaranteed to obtain the global optimum for submodular energy functions with binary variables.

### 4.3.7 Work by Globerson and Jaakkola

Globerson and Jaakkola [38] follow TRW's approach and compute a lower bound of the partition function by decomposing the model graph into disjoint planar subgraphs. They adopt and simplify the classical constructions of Kasteleyn [59, 60, 61] and Fisher [30, 31] to perform exact computation of the partition function (and marginals) for each planar subgraph. The model graph is first plane triangulated (Definition 40) and then used to construct the dual graph $G^{*}$. Since $G$ is plane triangulated, all vertices in $G^{*}$ have degree 3 , which is convenient for the next step of constructing the terminal graph $G^{T *}$ of $G^{*}$. The original edges of $G^{*}$ (and $G$ ) retain their weight in $G^{T *}$, while the introduced edges are given a weight of $e^{0}=1$. As shown by Fisher [31], there is now a $1: 1$ correspondence between perfect matchings in $G^{T *}$ and configurations of even subgraphs in $G$. Now the partition function of $G$ can be computed as a sum over the weights of perfect matchings in $G^{T *}$ (Section 2.3.4). $G^{T *}$ is given a Pfaffian orientation, from which the Kasteleyn matrix is constructed. The partition function of the model graph is then computed via the determinant of the Kasteleyn matrix. If $\mathcal{R}$ is the set of disjoint planar subgraphs of $G$ then the lower bound of the log-partition function is given as

$$
\begin{equation*}
\log Z(\boldsymbol{\theta}) \leq \sum_{R \in \mathcal{R}} p(R) \log Z(\boldsymbol{\theta}(R)) \tag{4.22}
\end{equation*}
$$

Our work is based on [38], although it simplifies it in a number of important ways. Our method does not require the construction of the expanded dual graph as everything is done in the model graph. Also, we break the construction of the Kasteleyn matrix into two phases, the first of which is invariant with respect to the model's disagreement costs. Further details are discussed in the next chapter.

## Chapter 5

## Planar Ising Graphical Models: Inference

In this chapter we introduce the Ising graphical model. This is an undirected graphical model that obeys the constraints of the Ising model, i.e., a binary-labeled energy function that is a sum of edge disagreement costs. Here we focus on the planar Ising graphical model, while in Chapter 7 we will look at its extensions to non-planar graphs. For the planar model we give polynomial-time algorithms for the exact computation of MAP (ground) states, log partition function and marginal edge probabilities. Our approach provides an interesting alternative to the well-known graph cut paradigm in that it does not impose any submodularity constraints. Instead, we require planarity to establish a correspondence with perfect matchings in an expanded dual graph. We describe a unified framework, while delegating complex but well-understood subproblems (planar embedding, maximum-weight perfect matching) to established algorithms, for which efficient implementations are freely available.

In Section 5.1 we show that an Ising model with an additional bias node can express any arbitrary binary energy function. In Section 5.1.2 we explain the planarity requirement and how it is affected by this additional bias node. In Section 5.1.4 we show that inference in a general planar graph can be broken down into inference in each of its biconnected components. Section 5.2 describes the expanded dual graph (Section 5.2.1), and how it is used for computing the MAP state‘ of the model graph. In Section 5.2 .2 we give the precise complementary connection between graph cuts of the model graph and perfect matchings of the expanded dual. We explain the role of planarity (resp. triangulation) in making that relation a surjection (resp. bijection). In Section 5.3 we deal with the calculation of the partition function and marginal probabilities. By using a particular indexing scheme (Section 5.3.3) we construct a correspondence between the edges of the model graph and those of its expanded dual. This is simpler than earlier work, as it eliminates the need to construct the expanded dual graph. In Section 5.3.3 we introduce the "proto-Kasteleyn" matrix $\boldsymbol{H}$, which can be efficiently stored as a prefactored bit matrix and reused in the parameter estimation loop. We show how to prefactor Kasteleyn matrices (Section 5.3.4), which reduces storage and speeds up computation of marginal probabilities. In Chapter 6 we apply all of the above algorithms to real examples of machine learning problems.


Figure 5.1: Constructive proof of Theorem 33: conversion of general energy (5.3) into Ising energy (5.1) via addition of a bias node. (a) shows the change in node energy $E_{i}^{\prime}$. (b) shows the change in edge energy $E_{i j}^{\prime}$. (c) our construction of an equivalent submodular model for $E_{i j}>0, E_{0 i}>0$ and $E_{0 j}<0$. (d) equivalent directed model of Kolmogorov and Zabih [70, Figure 2d].

### 5.1 Introduction

The Ising Graphical Model is a binary graphical model defined on an undirected graph $G(\mathcal{V}, \mathcal{E})$ whose energy function takes the form $E:\{0,1\}^{|\mathcal{E}|} \rightarrow \mathbb{R}$ with

$$
\begin{equation*}
E(\boldsymbol{y}):=\sum_{(i, j) \in \mathcal{E}} \llbracket y_{i} \neq y_{j} \rrbracket E_{i j} \tag{5.1}
\end{equation*}
$$

where $\llbracket \rrbracket \rrbracket$ denotes the indicator function, i.e., the disagreement cost $E_{i j}$ is incurred only in those states where $y_{i}$ and $y_{j}$ disagree. The joint probability distribution of an Ising graphical model is the same as that of a MRF (4.5):

$$
\begin{equation*}
\mathbb{P}(\boldsymbol{y})=\frac{1}{Z} e^{-E(\boldsymbol{y})}, \quad \text { where } Z:=\sum_{\boldsymbol{y}} e^{-E(\boldsymbol{y})} \tag{5.2}
\end{equation*}
$$

is the partition function. Recall that the energy function (4.19) of a general graphical model has the form

$$
\begin{equation*}
E^{\prime}(\boldsymbol{y}):=\sum_{i \in \mathcal{V}} E_{i}^{\prime}\left(y_{i}\right)+\sum_{(i, j) \in \mathcal{E}} E_{i j}^{\prime}\left(y_{i}, y_{j}\right) \tag{5.3}
\end{equation*}
$$

Compared to this general energy function, (5.1) with binary node labels imposes two additional restrictions: zero node energies, and edge energies in the form of disagreement costs. At first glance these constraints look severe. For example, such systems must obey the label symmetry $E(\boldsymbol{y})=E(\neg \boldsymbol{y})$, where $\neg$ denotes Boolean negation (ones' complement). However, Globerson and Jaakkola [38] suggested that adding a single node makes the Ising model (5.1) as expressive as the general model (5.3) for binary variables. Here we provide a formal proof:

Theorem 33 Every energy function of the form (5.3) over $n$ binary variables is equivalent to an Ising energy function of the form (5.1) over $n+1$ variables, with the additional variable held constant.

Proof We will prove this theorem by construction. Two energy functions are equivalent if they differ only by a constant. Without loss of generality, we add an extra bias variable $y_{0}$ and hold it constant at $y_{0}:=0$. Given an energy function of the form (5.3), we construct an Ising model with disagreement costs as follows:

1. For each node energy function $E_{i}^{\prime}\left(y_{i}\right)$, add a disagreement cost of $E_{0 i}:=E_{i}^{\prime}(1)-E_{i}^{\prime}(0)$, as shown in Figure 5.1(a). $E_{0 i}$ refers to the disagreement cost of the edge between node $i$ and the bias node. Note that in both states of $y_{i}$, the energy of the resulting Ising model is shifted relative to $E_{i}^{\prime}\left(y_{i}\right)$ by the same constant amount, namely $E_{i}^{\prime}(0)$ :

| $y_{i}$ | General energy (5.3) | Ising energy (5.1) |
| :---: | :---: | ---: |
| 0 | $E_{i}^{\prime}(0)$ | $0=E_{i}^{\prime}(0)-E_{i}^{\prime}(0)$ |
| 1 | $E_{i}^{\prime}(1)$ | $E_{0 i}=E_{i}^{\prime}(1)-E_{i}^{\prime}(0)$ |

2. For each edge energy function $E_{i j}^{\prime}\left(y_{i}, y_{j}\right)$, add the three disagreement cost terms

$$
\begin{align*}
& E_{i j}:=\frac{1}{2}\left(E_{i j}^{\prime}(0,1)+E_{i j}^{\prime}(1,0)-E_{i j}^{\prime}(0,0)-E_{i j}^{\prime}(1,1)\right) \\
& E_{0 i}:=E_{i j}^{\prime}(1,0)-E_{i j}^{\prime}(0,0)-E_{i j}  \tag{5.4}\\
& E_{0 j}:=E_{i j}^{\prime}(0,1)-E_{i j}^{\prime}(0,0)-E_{i j}
\end{align*}
$$

as shown in Figure 5.1(b). Note that for all states of $y_{i}$ and $y_{j}$, the energy of the resulting Ising model is shifted relative to $E_{i}^{\prime}\left(y_{i}\right)$ by the same constant amount, namely $E_{i j}^{\prime}(0,0)$ :

| $y_{i}$ | $y_{j}$ | General energy (5.3) | Ising energy (5.1) |
| :---: | :---: | :---: | ---: |
| 0 | 0 | $E_{i j}^{\prime}(0,0)$ | $0=E_{i j}^{\prime}(0,0)-E_{i j}^{\prime}(0,0)$ |
| 0 | 1 | $E_{i j}^{\prime}(0,1)$ | $E_{0 j}+E_{i j}=E_{i j}^{\prime}(0,1)-E_{i j}^{\prime}(0,0)$ |
| 1 | 0 | $E_{i j}^{\prime}(1,0)$ | $E_{0 i}+E_{i j}=E_{i j}^{\prime}(1,0)-E_{i j}^{\prime}(0,0)$ |
| 1 | 1 | $E_{i j}^{\prime}(1,1)$ | $E_{0 i}+E_{0 j}=E_{i j}^{\prime}(1,1)-E_{i j}^{\prime}(0,0)$ |

Summing the above terms gives us the bias of node i, i.e., its disagreement cost with the bias node

$$
\begin{equation*}
E_{0 i}=E_{i}^{\prime}(1)-E_{i}^{\prime}(0)+\sum_{j:(i, j) \in \mathcal{E}} E_{i j}^{\prime}(1,0)-E_{i j}^{\prime}(0,0)-E_{i j} \tag{5.5}
\end{equation*}
$$

where the sum iterates over all nodes $j$ connected to nodes $i$. This construction defines an Ising model whose energy in every configuration $y$ is shifted, relative to that of the general model we started with, by the same constant amount, namely $E^{\prime}(\mathbf{0})$ :

$$
\begin{align*}
E\left(\left[\boldsymbol{y}, y_{0}\right]\right) & =E^{\prime}(\boldsymbol{y})-\sum_{i \in \mathcal{V}} E_{i}^{\prime}(0)-\sum_{(i, j) \in \mathcal{E}} E_{i j}^{\prime}(0,0) \\
& =E^{\prime}(\boldsymbol{y})-E^{\prime}(\mathbf{0}) \tag{5.6}
\end{align*}
$$

where $\left[\boldsymbol{y}, y_{0}\right]$ denotes $y_{0}$ appended to the state $\boldsymbol{y}$. The energy functions of the two models are therefore equivalent.

Note how in the above construction, the label symmetry $E(\boldsymbol{y})=E(\neg \boldsymbol{y})$ of the Ising energy function (5.1) is conveniently broken by the introduction of a bias node, through the convention that $y_{0}:=0$.

### 5.1.1 Energy Minimization via Cuts

In Section 4.3 .5 we have seen that the minimum s-t cut of a directed graph can be used to compute the MAP state for a binary graphical model with a submodular energy function. We will now show that the MAP state of an Ising graphical model induces the minimum cut of its model graph:

Definition 34 The cut $C$ of a binary undirected graphical model $G(\mathcal{V}, \mathcal{E})$ induced by state $\boldsymbol{y} \in\{0,1\}^{n}$ is the set $\mathcal{C}(\boldsymbol{y})=\left\{(i, j) \in \mathcal{E}: y_{i} \neq y_{j}\right\}$; its weight $w(C(\boldsymbol{y}))$ is the sum of the weights of its edges.

Any given state $\boldsymbol{y}$ partitions the nodes of a binary graphical model into two sets: those labeled ' 0 ', and those labeled ' 1 '. The corresponding graph cut is the set of edges crossing the partition. Since only these edges contribute disagreement costs to the energy of an Ising graphical model (5.1), we have $w(C(\boldsymbol{y}))=E(\boldsymbol{y}) \forall \boldsymbol{y}$. Therefore the lowest-energy state of an Ising graphical model induces its minimum-weight cut. Conversely, the ground state of an Ising graphical model can be determined from its minimum-weight cut via a graph traversal, such as our depth-first-search shown in Algorithm 1.4 Algorithm 1 labels nodes as it encounters them, and checks for consistency on subsequent encounters. The traversal starts by labeling the bias node with its known state $y_{0}=0$.

We note that there is a bijection between states and their induced cuts. In particular, each state $\boldsymbol{y}$ induces a cut $C(\boldsymbol{y})$ and from each cut $C$ we can derive (up to label symmetry) the corresposponding node state $\boldsymbol{y}$. Thus we have that for all $\boldsymbol{y}$ the following holds: $\mathbb{P}(C(\boldsymbol{y}))=\mathbb{P}(\boldsymbol{y})$. Finally, the label symmetry can be broken by conditioning on any node $k$ :

$$
\begin{equation*}
\mathbb{P}(C(\boldsymbol{y}))=\mathbb{P}\left(\boldsymbol{y} \mid y_{k}\right) . \tag{5.7}
\end{equation*}
$$

```
Algorithm 1 Find State from Corresponding Cut
    Input: embedded model graph \(G(\mathcal{V}, \mathcal{E})\), cut \(\mathcal{C}(\boldsymbol{y}) \subseteq \mathcal{E}\)
            1. \(\forall i \in\{0,1,2, \ldots n\}: y_{i}=\) unknown
            2. dfs_state \((0,0)\)
    Output: state vector \(\boldsymbol{y}\)
    procedure dfs_state \((i \in\{0,1,2, \ldots n\}, s \in\{0,1\})\)
        if \(y_{i}:=\) unknown then
            1. \(y_{i}:=s\)
            2. \(\forall(i, j) \in \mathcal{E}_{i}\) : (edges incident to node \(i\) )
                if \((i, j) \in C\) then
                dfs_state \((j, \neg s)\)
            else dfs_state \((j, s)\)
            else if \(y_{i} \neq s\) then throw error
```

[^3]Minimum-weight cuts can be computed in polynomial time in graphs whose edge weights are all non-negative. We can construct an energy function equivalent to (5.1):

- Introduce one more bias node, with the constraint $y_{n+1}:=1$.
- Replace each negatively weighted bias edge $E_{0 i}<0$ by an edge to the new node $n+1$ with the positive weight $E_{i, n+1}:=-E_{0 i}>0$ (Figure 5.1(c)).

This still leaves us with the requirement that all non-bias edges be non-negative. This submodularity constraint (Definition 31) implies that agreement between nodes must be locally preferable to disagreement - a severe limitation.

Our construction (Figure 5.1(c)) differs from that of Kolmogorov and Zabih [70] (Figure 5.1(d)) in that we only use undirected edges. It may thus be possible to use edge direction for other purposes, such as representing a directed graphical model.

### 5.1.2 Planarity Constraint

Unlike graph cut methods, the exact inference algorithms we will describe do not depend on submodularity. Instead they require that the model graph is planar (see Definition 6 and Figure 5.2(b)), and that a planar embedding is provided (Definition 10).

In certain domains (e.g., when working with geographic information) a plane drawing of the graph may be available, from which the corresponding embedding is readily determined. Where it is not, we employ the algorithm of Boyer and Myrvold [11]. Given a connected graph $G$, this linear-time algorithm produces either a planar embedding for $G$ or a proof that $G$ is non-planar. Source code for this step is freely available [11, 139].

In Section 5.1 we have mapped the energy of a general binary graphical model (5.3) to the energy of a Ising graphical model (5.1) via the addition of a bias node. Now that we require that the Ising graphical model is planar, what does that imply for the original general model? If all nodes of the graph are to be connected to the bias node without violating planarity, the graph has to be outerplanar, i.e., have a planar embedding in which all its nodes lie on the external face - a very severe restriction (see Figure 5.2(a)).

The situation improves, however, if we do not insist that all nodes are connected to the bias. If only a subset $\mathcal{B} \subset \mathcal{V}$ of nodes have non-zero bias (5.5), then the graph only needs to be $\mathcal{B}$-outerplanar, i.e., have a planar embedding in which all nodes in $\mathcal{B}$ lie on the same (external) face. Model selection may thus entail the step of picking the face of a suitably embedded planar Ising graphical model whose nodes will be connected to the bias node. In image processing, for instance, where it is common to operate on a square grid of pixels, we can permit bias for all nodes on the perimeter of the grid, which borders the external face. We will refer to this as perimeter bias.

For each node $i$, we can assign a particular weight that measures how "important" it is to include that node's bias (5.5). This weight can be the magnitude or square of $E_{0 i}$. In general, a planar embedding which maximizes a weighted sum over the nodes bordering a given face can be found in linear time [42]; by setting node weights to some measure of their bias, such as the magnitude or square of $E_{0 i}$ (5.5), we can thus efficiently obtain the planar Ising model closest (in that measure) to any given planar binary graphical model.

In contrast to submodularity, $\mathcal{B}$-outerplanarity is a structural constraint. This has the advantage that once a model obeying the constraint is selected, inference (e.g., parameter estimation) can proceed via unconstrained methods (e.g., optimization).

It must be noted that all our exact algorithms can be extended to work exactly for non-planar graphs as well. In that case they take time exponential in the genus of the embedding though still polynomial in the size of the graph (Equation 3.10). Finally, in Chapter 7 we develop efficient approximate inference algorithms for non-planar graphs, such as the one shown in Figure 5.2(c).

(a)

(b)

(c)

Figure 5.2: (a) outerplanar graph. (b) planar graph. (c) non-planar graph.

### 5.1.3 Connectivity

All algorithms in this thesis require that the model graph $G$ is connected (Definition 4). Where this is not the case, we can simply determine the connected components (Definition 3) of $G$ in linear time [47], then invoke the algorithm in question separately on each of them. Since each component is unconditionally independent of all others (as they have no edges between them), the results can be trivially combined:

- $G$ is planar iff all of its connected components are planar. Any concatenation of a planar embedding for each connected component is a planar embedding of $G$.
- Any concatenation of a ground state for each connected component of $G$ is a ground state of $G$.
- The edge marginal probabilities of $G$ are the concatenation of edge marginal probabilities of its connected components.
- The $\log$ partition function of $G$ is the sum of the $\log$ partition functions of its connected components.


### 5.1.4 Biconnectivity

Although the algorithms in this thesis do not require the model graph $G$ to be biconnected (Definition 5), simpler and more efficient alternatives are applicable when this is the case.

As Figure 5.3 illustrates, any planar graph $G$ can be decomposed into a tree of edge-disjoint biconnected components which overlap in the articulation vertices they share (Definitions 3, 5). This


Figure 5.3: Skeletal chemical structure of (a) phosphorus trioxide (b) nitroglycerine and (c) quinine (courtesy of Wikipedia). Biconnected components are shown with shaded ovals. Trivial biconnected components are red, while nontrivial biconnected components are blue. Phosphorous trioxide is biconnected (i.e., all one component). Nitroglycerine is a tree, i.e., has only trivial biconnected components. Quinine is a general graph, i.e., has both trivial and nontrivial biconnected components. (d) shows the decomposition tree of quinine.
decomposition can be performed in linear time [47]. We call this tree of biconnected components the decomposition tree (Figure 5.3(d)). ${ }^{5}$ We show the following key result:

Theorem 35 Let $\mathcal{E}_{1}, \mathcal{E}_{2}, \ldots, \mathcal{E}_{n}$ be the edge sets corresponding to $n$ biconnected components of a graph $G(\mathcal{V}, \mathcal{E})$. The probability of a cut (Definition 34) induced by the states of a Markov Random Field (5.2) on $G$ factors into that of its biconnected components:

$$
\begin{equation*}
\mathbb{P}(C(\boldsymbol{y}))=\mathbb{P}(\boldsymbol{y})=\prod_{k=1}^{n} \mathbb{P}\left(C(\boldsymbol{y}) \cap \mathcal{E}_{k}\right) . \tag{5.8}
\end{equation*}
$$

Proof If $G(\mathcal{V}, \mathcal{E})$ is biconnected, $n=1$ and $\mathcal{E}_{1}=\mathcal{E}$ then Theorem 35 is trivially true. Otherwise split $G$ into a biconnected component $G_{1}\left(\mathcal{V}_{1}, \mathcal{E}_{1}\right)$ which is a leaf of the decomposition tree, and the remainder $G^{\prime}\left(\mathcal{V}^{\prime}, \mathcal{E}^{\prime}\right)$. It is always possible to find such a split, by splitting at the single articulation vertex $i$ connecting $G_{1}$ to $G^{\prime}$. To summarize:

$$
\begin{align*}
& \mathcal{V}_{1} \cup \mathcal{V}^{\prime}=\mathcal{V}, \quad \mathcal{V}_{1} \cap \mathcal{V}^{\prime}=\{i\} \\
& \mathcal{E}_{1} \cup \mathcal{E}^{\prime}=\mathcal{E}, \quad \mathcal{E}_{1} \cap \mathcal{E}^{\prime}=\emptyset, \quad G_{1} \text { is biconnected. } \tag{5.9}
\end{align*}
$$

Let $\boldsymbol{y}_{1}$ and $\boldsymbol{y}^{\prime}$ be the states in $\boldsymbol{y}$ restricted to vertices in $\mathcal{V}_{1}$ and $\mathcal{V}^{\prime}$, respectively. By definition, $\boldsymbol{y}_{1}$ is independent of $y^{\prime}$ when both are conditioned on the state $y_{i}$ of the articulation vertex $i$ connecting them. By using the result in Equation 5.7 we obtain

$$
\begin{align*}
\mathbb{P}(C(\boldsymbol{y})) & =\mathbb{P}\left(\boldsymbol{y} \mid y_{i}\right)=\mathbb{P}\left(\boldsymbol{y}_{1}, \boldsymbol{y}^{\prime} \mid y_{i}\right) \\
& =\mathbb{P}\left(\boldsymbol{y}_{1} \mid y_{i}\right) \mathbb{P}\left(\boldsymbol{y}^{\prime} \mid y_{i}\right)=\mathbb{P}\left(C(\boldsymbol{y}) \cap \mathcal{E}_{1}\right) \mathbb{P}\left(C(\boldsymbol{y}) \cap \mathcal{E}^{\prime}\right) \tag{5.10}
\end{align*}
$$

Recursively applying this argument to $G^{\prime}$ yields Theorem 35.
Even though the node states of two biconnected components do correlate through their common articulation vertex, Theorem 35 shows that their edge states are independent when conditioned on that articulation vertex. By decomposing graphs into their biconnected components, we can perform efficient inference in graphs that are not biconnected themselves.

## Ising Trees

Any undirected graph $G$ can be decomposed into biconnected components. These can be either trivial or nontrivial:

Definition 36 A biconnected component is trivial if it does not contain any cycles, otherwise it is nontrivial. A trivial biconnected component consists of a single edge and the two vertices it connects.

A tree $T$ does not contain any cycles, hence consists entirely of trivial biconnected components (Figure $5.3(\mathrm{~b})$ ). Theorem 35 then implies that each edge can be considered individually, making inference in an Ising tree $T(\mathcal{V}, \mathcal{E})$ simple:

[^4]- $T$ is planar; any embedding of $T$ is a planar embedding.
- The minimum-weight cut of $T$ is the set $\mathcal{E}^{-}=\left\{(i, j) \in \mathcal{E}: E_{i j}<0\right\}$. We can use Algorithm 1 to obtain the corresponding ground state.
- The marginal probability of any edge $(i, j)$ of $T$ is

$$
\begin{equation*}
\mathbb{P}((i, j) \in C)=\frac{\mathbb{P}\left(y_{i} \neq y_{j}\right)}{\mathbb{P}\left(y_{i}=y_{j}\right)+\mathbb{P}\left(y_{i} \neq y_{j}\right)}=\frac{e^{-E_{i j}}}{e^{-0}+e^{-E_{i j}}}=\frac{1}{1+e^{-E_{i j}}} . \tag{5.11}
\end{equation*}
$$

- The log partition function of $T$ is

$$
\begin{equation*}
\ln Z=\ln \prod_{(i, j) \in \mathcal{E}}\left(e^{-0}+e^{-E_{i j}}\right)=\sum_{(i, j) \in \mathcal{E}} \ln \left(1+e^{-E_{i j}}\right) . \tag{5.12}
\end{equation*}
$$

## General Case

The most efficient way to employ the inference algorithms (Sections 5.2 and 5.3 ) on graphs $G$ that are neither biconnected nor trees (e.g., Figure 5.3(c)) is to apply them to each nontrivial biconnected component of $G$ in turn. We can then use Theorem 35 to combine the results, along with the simple solutions given in Section 5.1.4 above for trivial biconnected components, into a result for the full graph. Letting $\mathcal{E}^{T} \subseteq \mathcal{E}$ denote the set of edges that belong to trivial biconnected components of $G$, we have:

- A planar embedding of $G$ is obtained by arbitrarily combining the planar embeddings for each of its nontrivial biconnected components and the edges in $\mathcal{E}^{T}$. The edges in $\mathcal{E}^{T}$ can be embedded arbitrarily.
- A minimum-weight cut of $G$ is the union between the edges in $\mathcal{E}^{-} \cap \mathcal{E}^{T}$ and a minimum-weight cut for each of $G$ 's nontrivial biconnected components. We can use Algorithm 1 to obtain the corresponding ground state.
- The marginal probability of an edge $(i, j) \in \mathcal{E}$ is $\left(1+e^{\left.-E_{i j}\right)^{-1}}\right.$ if $(i, j) \in \mathcal{E}^{T}$, or computed (by the method of Section 5.3) for the biconnected component it belongs to.
- The log partition function of $G$ is the sum of the log partition functions of its nontrivial biconnected components, plus $\sum_{(i, j) \in \mathcal{E}^{\mathcal{T}}} \ln \left(1+e^{-E_{i j}}\right)$ for its trivial biconnected components.

For the rest of this Chapter, we assume that $G$ is connected and planar, and that a plane embedding is provided. We do not require that $G$ is biconnected, though when this is not the case, it is generally more efficient to decompose $G$ into biconnected components as discussed above.

### 5.2 Computing Ground States

Frustration (Definition 25) is the key problem in the computation of ground states. A weighted graph is said to be frustrated if it contains at least one frustrated cycle, otherwise it is unfrustrated. Note that trees can never be frustrated because by their definition they do not contain any cycles.


Figure 5.4: Square face of a plane graph (dashed lines) with its ordinary (a) resp. expanded (b) dual graph (solid lines). (c1)-(c6): Binary node states (open and filled circles) of the graph, induced cuts (bold blue dashes), and complementary perfect matchings (bold red lines) of the expanded dual.

The lowest-energy (ground) state $\boldsymbol{y}^{*}:=\operatorname{argmin}_{y} E(y)$ of an unfrustrated Ising graphical model is found via essentially the same method as in a tree (Section 5.1.4): paint nodes as you traverse the graph, flipping the binary color of your paintbrush whenever you traverse an edge with a negative disagreement cost (as done by Algorithm 1 when invoked on the cut $C=\left\{(i, j) \in \mathcal{E}: E_{i j}<0\right\}$ ). This cannot lead to a contradiction because by Definition 25 you will flip the brush color an even number of times along any cycle in the graph, hence always end a cycle on the same color you started it with.

The presence of frustration unfortunately makes the computation of ground states much harder in fact, it is known to be NP-hard in general [2]. In this section we show that the ground state of a planar Ising graphical model can be computed exactly in polynomial time via maximum-weight perfect matching on the expanded dual of its embedded model graph.

A relationship between the states of a planar Ising model and perfect matchings (or dimer coverings) was first established by Kasteleyn [59, 60, 61] and Fisher [30, 31] (Section 3.2). Perfect matchings in dual graph constructs were used by Bieche et al. [9] (Section 3.3.1) and Barahona [2] (Section 3.3.2) to compute ground states of the Ising model. We generalize a simpler construction for triangulated graphs due to Globerson and Jaakkola [38] (Section 4.3.7). For rectangular lattices our approach reduces to the construction of Thomas and Middleton [126] (Section 3.3.3), though their algorithm to compute ground states is somewhat less straightforward. Pardella and Liers [98] apply this construction to very large square lattices (up to $3000 \times 3000$ ), and find it to be more efficient than earlier methods [2, 9].

### 5.2.1 Expanded Dual Graph

The dual graph for a square face of a plane model graph is shown with solid edges in Figure 5.4(a), and in Figure 5.5(b) for an entire graph. Each edge of the dual graph $G^{*}$ crosses exactly one edge of the original graph. Due to this one-to-one relationship we will consider the dual to have the same set
of edges $\mathcal{E}$ (with the same energies) as the original graph $G$. Since the nodes in the dual are different, however, we will (with some abuse of notation) use a single index for dual edges and their weights, corresponding to the index of the original edge in some arbitrary ordering of $\mathcal{E}$. Thus if $(i, j)$ is the $k^{\text {th }}$ edge in the (ordered) $\mathcal{E}$, its dual will have weight $E_{k}:=E_{i j}$. We now construct the expanded dual graph:

Definition $37 G_{E}^{*}$ is the expanded dual graph of the model graph G. It is constructed from the dual graph $G^{*}$ (Definition 12) by replacing each of its nodes with a $q$-clique, where $q$ is the degree of the node.

The expanded dual graph for a dual node with degree $q=4$ is shown in Figure 5.4(b), and in Figure 5.5 (c) for an entire graph. The additional edges internal to each $q$-clique are given zero energy so as to leave the model's energy unaffected. For large $q$ the introduction of these $q(q-1) / 2$ internal edges slows down subsequent computations (solid line in Figure 5.8(a)). This can be avoided by subdividing the offending $q$-gonal face of the model graph with chords (which are also given zero energy) before constructing the dual. Our implementation performs best when "octangulating" the graph, i.e., splitting octagons off all faces with $q>13$. In our experiments, we found this to be more efficient than a full triangulation (Figure 5.8(a)).

It can be seen that the expanded dual has $2|\mathcal{E}|$ vertices, two for each edge in the original graph. We therefore give the two vertices connected by the dual of the $k^{\text {th }}$ edge in $\mathcal{E}$ the indices $2 k-1$ and $2 k$ (see Section 5.3.3 and Figure 5.7). This consistent indexing scheme allows us to run the inference algorithms described in the remainder of this thesis without explicitly constructing the expanded dual graph data structure.

### 5.2.2 Complementary Perfect Matchings

Figures $5.5(\mathrm{~d})$ and $5.5(\mathrm{e})$ show two perfect matchings (in bold) of the nodes of the expanded dual of an Ising graphical model. We prove that there is a complementary relationship between such perfect matchings and graph cuts in the original Ising graphical model, characterized by the following two theorems. The reader may find it helpful to refer to Figure 5.5 while going through the proofs.

Theorem 38 For every cut $C$ of an embedded $\operatorname{graph} G(\mathcal{V}, \mathcal{E}, \Pi)$ there exists at least one perfect matching $\mathcal{M}$ of its expanded dual complementary to $C$, i.e., $\mathcal{E} \backslash \mathcal{M}=C$. If $G$ is triangulated (Definition 40) then there is exactly one such $\mathcal{M}$.

Proof By construction, the set of edges $\mathcal{E}$ constitutes a perfect matching of the expanded dual. Any subset of $\mathcal{E}$ therefore is a (possibly partial) matching of the expanded dual. The complement $\mathcal{M}^{\prime}:=\mathcal{E} \backslash C$ of a cut of $G$ is a subset of $\mathcal{E}$ and thus a matching of the expanded dual; it obeys $\mathcal{E} \backslash \mathcal{M}^{\prime}=\mathcal{E} \backslash(\mathcal{E} \backslash C)=C$. The nodes that $\mathcal{M}^{\prime}$ leaves unmatched in the expanded dual (bold red in Figures 5.5(d) and 5.5(e)) are those neighboring the edges of the cut $C$.

By definition, $C$ intersects any cycle of $G$, and therefore also the perimeters of $G$ 's faces $\mathcal{F}$, in an even number of edges. In each clique of the expanded dual, $\mathcal{M}^{\prime}$ thus leaves an even number of nodes unmatched. It can therefore be completed into a perfect matching $\mathcal{M} \supseteq \mathcal{M}^{\prime}$ using only edges interior to the cliques to pair up unmatched nodes. These edges have no counterpart in the original graph:


Figure 5.5: (a) Planar Ising graphical model with graph $G$. $G$ has five binary nodes, including a bias node (light blue) fixed to label ' 0 '. (b) The dual graph $G^{*}$ (small red nodes). (c) The expanded dual graph $G_{E}^{*}$. (d) and (e) show two different states of $G$. The graph cut induced by the given state and its complementary perfect matching of the expanded dual are shown as bold blue and bold red edges, respectively. The nodes left unmatched by the complement $\mathcal{M}^{\prime}:=\mathcal{E} \backslash C$ of the cut in the expanded dual are in bold red.
$\left(\mathcal{M} \backslash \mathcal{M}^{\prime}\right) \cap \mathcal{E}=\emptyset$. We thus have

$$
\begin{equation*}
\mathcal{E} \backslash \mathcal{M}=\mathcal{E} \backslash\left[\left(\mathcal{M} \backslash \mathcal{M}^{\prime}\right) \cup \mathcal{M}^{\prime}\right]=\left[\mathcal{E} \backslash\left(\mathcal{M} \backslash \mathcal{M}^{\prime}\right)\right] \backslash \mathcal{M}^{\prime}=\mathcal{E} \backslash \mathcal{M}^{\prime}=C \tag{5.13}
\end{equation*}
$$

In a $K_{3}$ clique of the expanded dual, $\mathcal{M}^{\prime}$ will leave two nodes unmatched or none at all. In either case there is only one way to complete the matching, by adding one edge resp. none at all. By construction, if $G$ is triangulated all cliques in its expanded dual are $K_{3}$ cliques, and so $\mathcal{M}$ is unique.

Theorem 38 shows that there exists a surjection from perfect matchings in $G_{E}^{*}$ to cuts in $G$. Furthermore, since we have given edges interior to the cliques of the expanded dual zero energy, every perfect matching $\mathcal{M}$ complementary to a cut $\mathcal{C}$ of our Ising graphical model (5.1) obeys the relation

$$
\begin{equation*}
w(\mathcal{M})+w(\mathcal{C})=w\left(\mathcal{M}^{\prime}\right)+w(\mathcal{C})=\sum_{(i, j) \in \mathcal{E}} E_{i j}=\text { constant } \tag{5.14}
\end{equation*}
$$

where $w(\mathcal{M})$ and $w(C)$ are the weights of the perfect matching $\mathcal{M}$ and the cut $C$, respectively. This complementary relationship means that instead of a minimum-weight cut in a graph we can look for a maximum-weight perfect matching in its expanded dual. But will that matching always be complementary to a cut? We now prove the following theorem, which shows that this is true for plane graphs:

Theorem 39 Every perfect matching $\mathcal{M}$ of the expanded dual of a plane graph $G(\mathcal{V}, \mathcal{E}, \Pi)$ is complementary to a cut $\mathcal{C}$ of $G$, i.e., $\mathcal{E} \backslash \mathcal{M}=\mathcal{C}$.

Proof By definition, $\mathcal{E} \backslash \mathcal{M}$ is a cut of $G$ iff it intersects every cycle $\mathcal{O} \subseteq \mathcal{E}$ of $G$ an even number of times. This can be shown by induction over the faces of the embedding:

Base case - let $O \subseteq \mathcal{E}$ be the perimeter of a face of the embedding, and consider the corresponding clique of the expanded dual: $\mathcal{M}$ matches an even number of nodes in the clique via interior edges. All other nodes must be matched by edges crossing $O$. The complement of the matching in $G$ thus intersects $O$ an even number of times:

$$
\begin{equation*}
|(\mathcal{E} \backslash \mathcal{M}) \cap O| \equiv 0 \quad(\bmod 2) \tag{5.15}
\end{equation*}
$$

Induction - let $O_{i}, O_{j} \subseteq \mathcal{E}$ be cycles in $G$ obeying (5.15). We define the symmetric difference of two cycles $O_{i}$ and $O_{j}$ as $O_{i} \Delta O_{j}:=\left(O_{i} \cup O_{j}\right) \backslash\left(O_{i} \cap O_{j}\right)$. We now have

$$
\begin{align*}
\left|(\mathcal{E} \backslash \mathcal{M}) \cap\left(O_{i} \Delta O_{j}\right)\right| & =\left|\left[(\mathcal{E} \backslash \mathcal{M}) \cap\left(O_{i} \cup O_{j}\right)\right] \backslash\left(O_{i} \cap O_{j}\right)\right| \\
& =\left|\left[(\mathcal{E} \backslash \mathcal{M}) \cap O_{i}\right] \cup\left[(\mathcal{E} \backslash \mathcal{M}) \cap O_{j}\right]\right|-\left|(\mathcal{E} \backslash \mathcal{M}) \cap\left(O_{i} \cap O_{j}\right)\right| \\
& =\left|(\mathcal{E} \backslash \mathcal{M}) \cap O_{i}\right|+\left|(\mathcal{E} \backslash \mathcal{M}) \cap O_{j}\right|-2\left|(\mathcal{E} \backslash \mathcal{M}) \cap O_{i} \cap O_{j}\right| \\
& \equiv 0+0-2 n \equiv 0 \quad(\bmod 2), \forall n \in \mathbb{N} . \tag{5.16}
\end{align*}
$$

We see that property (5.15) is preserved under composition of cycles via symmetric differences, and thus holds for all cycles that can be composed from face perimeters of the embedding of $G$.

All cycles in a plane graph $G$ are contractible on its embedding surface (a plane or a sphere) because that surface has zero genus, and is therefore simply connected. Intuitively this means that all such
cycles can be "contracted" to a single point, while remaining on the surface of their embedding. All contractible cycles of $G$ can be constructed by composition of face perimeters via symmetric differences, thus intersect $\mathcal{E} \backslash \mathcal{M}$ an even number of times. Therefore $\mathcal{E} \backslash \mathcal{M}$ is a cut.

This is where planarity matters: surfaces of non-zero genus are not simply connected, and thus nonplane graphs may contain non-contractible cycles (e.g., cycles enclosing the hole of a torus, as shown in Figure 7.2). In the presence of frustration, our construction does not guarantee that the complement $\mathcal{E} \backslash \mathcal{M}$ of a perfect matching of the expanded dual contains an even number of edges along such cycles. For planar graphs, however, the above theorems allow us to leverage known polynomial-time algorithms for perfect matchings into inference methods for Ising graphical models. This approach also works for non-planar Ising graphical models that do not contain any frustrated non-conctractible cycles.

We note that if all cliques of the expanded dual are of even size, there is also a direct (noncomplementary) surjection from perfect matchings to cuts in the original graph (see Section 3.2.2). In contrast to our complementary map in Theorem 38, the direct surjection requires the addition of dummy vertices into the expanded dual for faces of $G$ with odd perimeter, so as to make the corresponding cliques even [60, 84].

### 5.2.3 Computing the Ground State

The blossom-shrinking algorithm $[23,24]$ is a sophisticated method to efficiently compute the maximumweight perfect matching of a graph. It can be implemented to run in as little as $O(|\mathcal{E}||\mathcal{V}| \log |\mathcal{V}|)$ time [89]. A detailed description of blossom-shrinking is provided in Section 7.1. In our experiments we used both Blossom IV [17] and the more recent Blossom V code [67]. We can now efficiently compute the lowest-energy state of a planar Ising graphical model as follows:

1. Find a planar embedding of the model graph $G$ (Section 2.2.2).
2. Construct the expanded dual of $G$ (Section 5.2.1).
3. Run the blossom-shrinking algorithm on $G_{E}^{*}$ to compute its maximum-weight perfect matching. Its complement in the original model graph is the minimum-weight graph cut $C$ (Section 5.2.2).
4. Identify the state which induces this cut $C$ via a graph traversal (Algorithm 1).

### 5.3 Computing the Partition Function and Marginal Probabilities

As it involves a summation over exponentially many states $\boldsymbol{y}$, calculating the partition function (5.2) is generally intractable. For planar graphs, however, the generating function for perfect matchings can be calculated in polynomial time via the determinant of a skew-symmetric matrix [30, 31, 59, 60, 61], which we call the Kasteleyn matrix $\boldsymbol{K}$ (Section 2.3.3). Due to the close relationship with graph cuts (Section 5.2.2) we can apply this method to calculate $Z$ in (5.2). We first convert a planar embedding of the Ising model graph into a Boolean "proto-Kasteleyn" matrix $\boldsymbol{H}$ :

1. Plane triangulate (Section 5.3.1) the embedded graph so as to make the relationship between cuts and complementary perfect matchings a bijection (Section 5.2.2).


Figure 5.6: (a) A chordal graph whose external face is not triangulated. (b) a plane triangulated graph that has a cycle of length 4 (bold blue) without a chord. (c) proper and (d) improper plane triangulations (dashed) of the plane graph from Figure 2.2(d).
2. Orient the edges of the graph such that the in-degree of every node is odd (Section 5.3.2).
3. Construct the Boolean proto-Kasteleyn matrix $\boldsymbol{H}$ from the oriented graph (Section 5.3.3).
4. Prefactor the triangulation edges (added in Step 1) out of $\boldsymbol{H}$ (Section 5.3.4).

Our Step 2 simplifies equivalent operations in previous constructions [31, 38, 60, 61]. Step 3 differs in that it only sets unit (i.e., +1) entries in a Boolean matrix. Step 4 can dramatically reduce the size of $\boldsymbol{H}$ for compact storage (as a bit matrix) and faster subsequent computations (Figure 5.8).

Edge disagreement costs do not enter into $\boldsymbol{H}$. They are only taken into account in a second phase, when the full Kasteleyn matrix $\boldsymbol{K}$ is constructed from $\boldsymbol{H}$ (Section 5.3.3). We can then factor $\boldsymbol{K}$ (Section 5.3.4) and compute the partition function from its determinant (Section 5.3.4 and [30, 59]). This factorisation can also be used to invert $\boldsymbol{K}$, which is necessary for obtaining the marginal probabilities of disagreement on the edges of the model graph (Section 5.3.4).

In what follows, we elaborate in turn on the graph operations of plane triangulation (Section 5.3.1), odd edge orientation (Section 5.3.2), construction (Section 5.3.3) and factoring (Section 5.3.4) of the Kasteleyn matrix $\boldsymbol{K}$ resp. $\boldsymbol{H}$.

### 5.3.1 Plane Triangulation

We begin by plane triangulating the model graph:
Definition 40 An embedded graph is plane triangulated iff it is biconnected (Definition 5) and each of its faces (including the external face) is a triangle.

Note that plane triangulation is not equivalent to making a graph chordal (Definition 29), though the latter process is sometimes also called "triangulation". For instance, the graph in Figure 5.6(a) is chordal but not plane triangulated because the external face is not triangular, while that in Figure 5.6(b) is plane triangulated but not chordal because it contains a cycle of length 4 (blue) that has no chord.

We can plane triangulate an embedded graph in linear time by traversing all of its faces and inserting chords as necessary as we go along (Algorithm 2). This may create multiple edges between the same two vertices, such as edge ( 2,4 ) in Figure 5.6(c). Care must be taken when encountering "dangling"
nodes, i.e., nodes in trivial biconnected components that are leaves of the decomposition tree. Such nodes could cause the insertion of a self-loop (edge (4,5) in Figure 5.6(d)). Our Algorithm 2 detects and biconnects such components, as Definition 40 requires. From now on, we will refer to the edges added by Algorithm 2 as the triangulation edges.

The insert_chord $(i, j, k)$ subroutine of Algorithm 2 updates $\mathcal{E}, \pi_{i}$ and $\pi_{k}$ so as to insert the new edge ( $i, k$ ) in its proper place in the rotation system. In order to leave the distribution (5.2) modeled by the graph unchanged, the new edge is given zero energy. Repeated traversals of the same face in Algorithm 2 can be avoided by the use of "done" flags, which have been omitted here for the sake of clarity.

Note that plane triangulation is not strictly necessary for the computation of partition function or marginal probabilities. Plane triangulation ensures that the graph becomes biconnected, so an edge cannot border the same face of the model graph on both sides. Thus the expanded dual is a proper graph, i.e., has no multiple edges and no self loops.

In previous work [31, 38], plane triangulation came at a computational price: the edges added during plane triangulation can make factoring and inversion of $\boldsymbol{K}$ (Section 5.3.4) significantly (up to 20 times) more expensive. We avoid this cost by removing the triangulation edges before constructing $\boldsymbol{K}$ (Section 5.3.4). This is legal, because we are not interested in the marginal probability of triangulation edges.

```
Algorithm 2 Plane Triangulation
    Input: plane graph \(G(\mathcal{V}, \mathcal{E}, \Pi)\) with \(|\mathcal{V}| \geq 3\)
            \(\forall i \in \mathcal{V}\) :
            \(\forall(i, j) \in \mathcal{E}_{i}: \quad\) (edges incident to node \(i\) )
            1. \((j, k):=\pi_{j}(j, i)\)
            2. \((k, l):=\pi_{k}(k, j)\)
            3. while \(l \neq i\) or \(\pi_{l}(l, k) \neq(l, j)\) :
                    (a) if \(i=k\) then (avoid self-loop)
                    \(i:=j, j:=k, k:=l\)
                    \((k, l):=\pi_{k}(k, j)\)
                    (b) insert_chord \((i, j, k)\)
                    (c) \(i:=k, j:=l\)
                    (d) \((j, k):=\pi_{j}(j, i)\)
                    (e) \((k, l):=\pi_{k}(k, j)\)
    Output: plane triangulated graph \(G(\mathcal{V}, \mathcal{E}, \Pi)\)
```

```
procedure insert_chord \((i, j, k \in \mathcal{V}) \quad\) (insert \((k, i)\) between \((k, j)\) and \(\pi_{k}(k, j)\) )
```

procedure insert_chord $(i, j, k \in \mathcal{V}) \quad$ (insert $(k, i)$ between $(k, j)$ and $\pi_{k}(k, j)$ )
1. $\mathcal{E}:=\mathcal{E} \cup\{(i, k)\}$
1. $\mathcal{E}:=\mathcal{E} \cup\{(i, k)\}$
Update $\pi_{k}$
Update $\pi_{k}$
2. $\pi_{k}(k, i):=\pi_{k}(k, j)$
2. $\pi_{k}(k, i):=\pi_{k}(k, j)$
3. $\pi_{k}(k, j):=(k, i)$
3. $\pi_{k}(k, j):=(k, i)$
Update $\pi_{i}$
Update $\pi_{i}$
4. $\pi_{i}\left(\pi_{i}^{-1}(i, j)\right):=(i, k) \quad\left(\pi_{i}^{-1}(i, j)\right.$ is the counter-clockwise edge from $\left.(i, j)\right)$
4. $\pi_{i}\left(\pi_{i}^{-1}(i, j)\right):=(i, k) \quad\left(\pi_{i}^{-1}(i, j)\right.$ is the counter-clockwise edge from $\left.(i, j)\right)$
5. $\pi_{i}(i, k):=(i, j)$
5. $\pi_{i}(i, k):=(i, j)$
6. $E_{i k}:=0$

```
    6. \(E_{i k}:=0\)
```



Figure 5.7: Clockwise odd orientation (Section 5.3.2) and indexing scheme (Section 5.3.3) for the expanded dual graph $G_{E}^{*}$ (red, small nodes) of the model graph $G$ (large nodes).

### 5.3.2 Odd Edge Orientation

To calculate the generating function for perfect matchings, the expanded dual graph $G_{E}^{*}$ must be given a clockwise odd orientation (Section 2.3.4, Lemma 18). Here we will describe an equivalent orientation for the model graph $G$.

We will now explain the orientation process by referring to Figure 5.7. By giving all interior edges of the $K_{3}$ cliques of the expanded dual graph $G_{E}^{*}$ a clockwise orientation (small red arrows), we ensure that

- The interior faces of the $K_{3}$ cliques have a clockwise odd orientation.
- All interior edges of the $K_{3}$ cliques are oriented counter-clockwise wrt. all faces exterior to the $K_{3}$ cliques, hence do not affect the latters' clockwise odd orientation status.

It remains to consider the orientation of edges external to the $K_{3}$ cliques (large red arrows in Figure 5.7). What does a clockwise odd orientation of these edges correspond to in the original model graph $G$ ? To map these edges back into the model graph, rotate them clockwise by $90^{\circ}$ degrees. A face with a clockwise odd orientation of its perimeter in $G_{E}^{*}$ thus maps to a vertex with an odd in-degree, i.e., a vertex with an odd number of incoming edges. This facilitates a drastic simplification of this step in our construction:

Lemma 41 To establish a clockwise odd orientation of the expanded dual graph $G_{E}^{*}$, orient the edges of the model graph $G$ such that all vertices, except possibly one, have an odd in-degree.

Our Algorithm 3 achieves this time linear in $|\mathcal{E}|$ by orienting edges appropriately upon return from a depth-first traversal of the graph. In contrast to earlier constructions [31, 38, 60], it does not require following orbits around faces, and in fact does not refer to an embedding $\Pi$ or dual graph $G^{*}$ at all.

```
Algorithm 3 Construct Odd Edge Orientation
    Input: undirected graph \(G(\mathcal{V}, \mathcal{E})\)
            1. \(\forall v \in \mathcal{V}: v\).visited \(:=\) false
            2. pick arbitrary edge \((r, s) \in \mathcal{E}\)
            3. \(\mathcal{E}^{\prime}:=\{(r, s)\}\)
            4. make_odd \((r, s)\)
    Output: orientation \(\mathcal{E}^{\prime}\) of \(G(\mathcal{V}, \mathcal{E})\) :
            in-degree \((v) \equiv 1(\bmod 2), \forall v \in \mathcal{V} \backslash\{s\}\)
    function make_odd: \((u, v \in \mathcal{V}) \rightarrow\{\) true, false \(\}\)
        1. \(\mathcal{E}:=\mathcal{E} \backslash\{(u, v)\}\)
        2. if \(v\).visited=true then return true
        3. \(v\). visited \(:=\) true
        4. odd := false
        5. \(\forall\{v, w\} \in \mathcal{E}\) :
            if make_odd \((v, w)=\) true then
            (a) \(\mathcal{E}^{\prime}:=\mathcal{E}^{\prime} \cup\{(w, v)\}\)
            (b) odd \(:=\neg\) odd
            else \(\quad \mathcal{E}^{\prime}:=\mathcal{E}^{\prime} \cup\{(v, w)\}\)
    6. return odd
```

In Algorithm 3 any vertex can be chosen to be the exceptional vertex $s$ (Lemma 41), since the choice of the external face of a plane drawing is arbitrary. The external face is an artifact of the drawing, not an intrinsic property of the embedding. A planar graph embedded on a sphere has no external face (see Figure 2.7).

### 5.3.3 Constructing the Kasteleyn Matrix

The Kasteleyn matrix $\boldsymbol{K}$ is a skew-symmetric, $2|\mathcal{E}| \times 2|\mathcal{E}|$ matrix constructed from the model graph of the Ising graphical model. Its determinant is the square of the partition function. Our construction improves upon the work of Globerson and Jaakkola [38] in a number of ways:

- We employ an indexing scheme (Section 5.3.3) that removes any need to refer to the expanded dual graph $G_{E}^{*}$, which we never explicitly construct.
- We break the construction of the Kasteleyn matrix into two phases, the first of which is invariant with respect to the model's disagreement costs. This has several advantages as discussed in Section 5.3.3.
- We make the "proto-Kasteleyn" matrix $\boldsymbol{H}$ computed in the first phase very compact by prefactoring out the triangulation edges (see Section 5.3.4) and storing it as a bit matrix. Empirically this makes $\boldsymbol{H}$ several orders of magnitude smaller than $\boldsymbol{K}$.


Figure 5.8: Cost of our inference methods on a ring graph (i.e., single cycle), plotted against ring size. (a) and (b): CPU time on Apple MacBook with 2.2 GHz Intel Core2 Duo processor, averaged over 100 repetitions. (a) MAP state calculated via Blossom IV [17] on original, triangulated, and octangulated ring. (b) marginal edge probabilities with $v s$. without prefactoring. (c) size of $\boldsymbol{K}$ (double precision, no prefactoring) vs. prefactored bit matrix $\boldsymbol{H}$, in uncompressed (solid lines) vs. compressed form (dashed lines), using compressed row storage for $\boldsymbol{K}$ and bzip2 compression for $\boldsymbol{H}$.

## Indexing Scheme

The expanded dual $G_{E}^{*}$ has $2|\mathcal{E}|$ vertices, one lying to either side of every edge of the model graph $G$. We associate each model edge with an index $k$, so if $e_{k}=(i, j) \in \mathcal{E}$ then $\pi_{i}\left(e_{k}\right)=\pi_{i}(i, j)$ (Definition 7). When viewing the model edge $e_{k}$ along its direction in $\mathcal{E}^{\prime}$, we label the dual node to its right $2 k-1$ and that to its left $2 k$. For example in Figure 5.7 the node to the left of $e_{3}$ is 6 , while the one to its right is 5. One benefit of this scheme is that quantities relating to the edges of the model graph (as opposed to internal edges of the cliques of the expanded dual) will always be found on the superdiagonal of $\boldsymbol{K}$.

## Two-Phase Construction

In the first phase we process the structure of the model graph into a Boolean "proto-Kasteleyn" matrix $\boldsymbol{H}$ which does not yet include disagreement costs. $\boldsymbol{H}$ only has positive entries, and only those corresponding to edges with zero disagreement cost, i.e., those added during plane triangulation or those internal to the cliques of the expanded dual. All such entries have the weight $e^{0}=1$, making $\boldsymbol{H}$ a Boolean matrix, which can be stored compactly as a bit matrix. Using the indexing scheme from Section 5.3.3, our Algorithm 4 constructs $\boldsymbol{H}$ in linear time, cycling once through the edges incident upon each vertex of the model graph.

In the second phase, we use $\boldsymbol{H}$ to construct the conventional, real-valued Kasteleyn matrix $\boldsymbol{K}$ by adding the exponentiated disagreement costs $E_{k}, \forall k=\{1,2, \ldots|\mathcal{E}|\}$ along the superdiagonal and skewsymmetrizing:

1. $K:=H$
2. $\boldsymbol{K}_{2 k-1,2 k}:=\boldsymbol{K}_{2 k-1,2 k}+e^{E_{k}} \quad \forall k \in\{1,2, \ldots|\mathcal{E}|\}$
3. $\boldsymbol{K}:=\boldsymbol{K}-\boldsymbol{K}^{\top}$

This two-phase construction of $\boldsymbol{K}$ has a number of advantages over classical constructions [30, 38, 59]:

- When working with a large number of isomorphic graphs (as we do in Chapter 6), the corresponding proto-Kasteleyn matrix is identical for all of them, hence needs to be constructed just once.
- During maximum likelihood parameter estimation, the partition function and/or marginals have to be recomputed many times for the same graph, with disagreement costs varying due to the ongoing adaptation of the model parameters. $\boldsymbol{H}$ remains valid when disagreement costs change, so we can compute it just once upfront and then reuse it in the parameter estimation loop.
- $\boldsymbol{H}$ can be stored very compactly as a prefactored bit matrix. As Figure 5.8(c) shows, the uncompressed $\boldsymbol{H}$ can be several orders of magnitude smaller than the corresponding Kasteleyn matrix $\boldsymbol{K}$. Since $G_{E}^{*}$ is cubic (i.e., every node has degree 3 ), the Kasteleyn matrix $\boldsymbol{K}$ is sparse with each row and column having exactly 3 non-zero entries. Compressed row storage [1] of $\boldsymbol{K}$ is efficient, but applying the bzip2 compressor (from http://www.bzip.org/) to the prefactored bit matrix $\boldsymbol{H}$ yields by far the most compact storage format we have found. Such memory efficiency becomes very important when working with large data sets of non-isomorphic graphs.

```
Algorithm 4 Construct proto-Kasteleyn Bit Matrix \(H\)
    Input: oriented, embedded and triangulated graph \(G\left(\mathcal{V}, \mathcal{E}^{\prime}, \Pi\right)\)
        1. \(\boldsymbol{H}:=\mathbf{0} \in\{0,1\}^{2\left|\mathcal{E}^{\prime}\right| \times 2\left|\mathcal{E}^{\prime}\right|}\)
        2. \(\forall v \in \mathcal{V}\) :
            (a) \(e_{s}:=\) any edge incident on \(v\)
            (b) if \(e_{s}\) points to \(v\) then \(\alpha:=2 s\)
                else \(\alpha:=2 s-1\)
            (c) \(e_{r}:=\pi_{v}\left(e_{s}\right)\)
            (d) do
                if \(e_{r}\) points to \(v\) then
                    \(H_{2 r-1, \alpha}\) := 1
                    \(\alpha:=2 r\)
                    if \(e_{r}\) was created by plane triangulation (Alg. 2) then
                    \(H_{2 r-1,2 r}:=1\)
                else
                    \(H_{2 r, \alpha}:=1\)
                    \(\alpha:=2 r-1\)
                \(e_{r}:=\pi_{v}\left(e_{r}\right)\)
            while \(e_{r} \neq \pi_{v}\left(e_{s}\right)\)
    Output: proto-Kasteleyn bit matrix \(\boldsymbol{H}\)
```


### 5.3.4 Factoring Kasteleyn Matrices

Standard approaches such as LU-factorization can be used to factor the Kasteleyn matrix $\boldsymbol{K}$, but they do not exploit its skew symmetry and may encounter numerical difficulties. The preferred decompositions for even-sized skew-symmetric matrices are $L D L^{\top}$ (i.e., root-free Cholesky-style) factorizations
operating on $2 \times 2$ matrix blocks [46, Chapter 11]. Here we develop such a decomposition which can be used to factor $\boldsymbol{K}$ as well as to prefactor $\boldsymbol{H}$ (see below). We begin by writing the Kasteleyn matrix as

$$
\boldsymbol{K}=\left[\begin{array}{rr|r}
0 & c & \boldsymbol{a}^{\top}  \tag{5.17}\\
-c & 0 & \boldsymbol{b}^{\top} \\
\hline-\boldsymbol{a} & -\boldsymbol{b} & \boldsymbol{C}
\end{array}\right]
$$

for some scalar $c$, vectors $\boldsymbol{a}$ and $\boldsymbol{b}$, and a matrix $\boldsymbol{C}$ which is either empty or again of the same form. We factor (5.17) into [see 5, 14]

$$
\boldsymbol{K}=\left[\begin{array}{rr|r}
0 & -1 & \mathbf{0}^{\top}  \tag{5.18}\\
1 & 0 & \mathbf{0}^{\top} \\
\hline \boldsymbol{a} / c & \boldsymbol{b} / c & \boldsymbol{I}
\end{array}\right]\left[\begin{array}{rr|r}
0 & c & \mathbf{0}^{\top} \\
-c & 0 & \mathbf{0}^{\top} \\
\hline \mathbf{0} & \mathbf{0} & \boldsymbol{C}^{\prime}
\end{array}\right]\left[\begin{array}{rr|r}
0 & 1 & \boldsymbol{a}^{\top} / c \\
-1 & 0 & \boldsymbol{b}^{\top} / c \\
\hline \mathbf{0} & \mathbf{0} & \boldsymbol{I}
\end{array}\right]
$$

where $\mathbf{0}$ is the zero matrix, $\boldsymbol{I}$ is the identity matrix and $\boldsymbol{C}^{\prime}$ is the Schur complement given as

$$
\begin{equation*}
\boldsymbol{C}^{\prime}:=\boldsymbol{C}+\left(\boldsymbol{b} \boldsymbol{a}^{\top}-\boldsymbol{a} \boldsymbol{b}^{\top}\right) / c \tag{5.19}
\end{equation*}
$$

Iterated application of (5.18) to the Schur complement ultimately yields $\boldsymbol{K}=\boldsymbol{R}^{\top} \boldsymbol{J} \boldsymbol{R}$, where

To prevent small pivots $c_{i}$ from causing numerical instability, pivoting is required. In our experience, partial pivoting suffices here since Kasteleyn matrices are sparse and have at least two entries of unit magnitude (i.e., suitable pivots) in each row and column.

## Partition Function

The partition function for perfect matchings is $\sqrt{|\boldsymbol{K}|}[30,59]$. Our factoring gives $|\boldsymbol{R}|=1$ and $|\boldsymbol{J}|=$ $\prod_{i} c_{i}^{2}$, so we have

$$
\begin{equation*}
\sqrt{|\boldsymbol{K}|}=\sqrt{\left.\mid \boldsymbol{R}^{\eta}\right\rceil|\boldsymbol{J}||\boldsymbol{R}|}=\sqrt{|\boldsymbol{J}|}=\prod_{i=1}^{|\mathcal{E}|}\left|c_{i}\right| \tag{5.21}
\end{equation*}
$$

Calculation of the product in (5.21) is prone to numerical overflow. This can be avoided by working with logarithms. Using the complementary relationship (5.14) with graph cuts in planar Ising graphical
models, we show that the log partition function for the latter is

$$
\begin{align*}
\ln Z & :=\ln \sum_{\boldsymbol{y}} \exp \left(-\sum_{e_{k} \in \mathcal{C}(\boldsymbol{y})} E_{k}\right)=\ln \sum_{\boldsymbol{y}} \exp \left(-\left(\sum_{e_{k} \in \mathcal{E}} E_{k}-\sum_{e_{k} \notin \mathcal{C}(\boldsymbol{y})} E_{k}\right)\right)  \tag{5.22}\\
& =\ln \left(\exp \left(-\sum_{e_{k} \in \mathcal{E}} E_{k}\right) \sum_{\boldsymbol{y}} \exp \left(\sum_{e_{k} \notin C(\boldsymbol{y})} E_{k}\right)\right)=\ln \sqrt{|\boldsymbol{K}|}-\sum_{e_{k} \in \mathcal{E}} E_{k}=\sum_{i=1}^{|\mathcal{E}|}\left(\ln \left|c_{i}\right|-E_{i}\right) .
\end{align*}
$$

## Marginal Probabilities

The marginal probability of disagreement along an edge equals the negative gradient of the log partition function (5.22) with respect to the disagreement costs. Computing this involves the inverse of $\boldsymbol{K}$. If $\mathcal{C}$ is some cut of the model graph, we compute the marginal probability as follows:

$$
\begin{align*}
\mathbb{P}\left(e_{k} \in C\right) & =\sum_{\boldsymbol{y}: e_{k} \in C(\boldsymbol{y})} \mathbb{P}(\boldsymbol{y})=\frac{1}{Z} \sum_{\boldsymbol{y}: e_{k} \in C(\boldsymbol{y})} e^{-E(\boldsymbol{y})}=-\frac{1}{Z} \frac{\partial Z}{\partial E_{k}}=-\frac{\partial \ln Z}{\partial E_{k}} \\
& =1-\frac{1}{2|\boldsymbol{K}|} \frac{\partial|\boldsymbol{K}|}{\partial E_{k}}=1-\frac{1}{2} \operatorname{tr}\left(\boldsymbol{K}^{-1} \frac{\partial \boldsymbol{K}}{\partial E_{k}}\right)=1+\boldsymbol{K}_{2 k-1,2 k}^{-1} \boldsymbol{K}_{2 k-1,2 k} \tag{5.23}
\end{align*}
$$

where tr denotes the matrix trace, and we have used the fact that $\boldsymbol{K}^{-1}$ is also skew-symmetric. To invert $\boldsymbol{K}$, observe from (5.20) that $\boldsymbol{R}$ and $\boldsymbol{J}$ are essentially triangular resp. diagonal (simply swap rows $2 k-1$ and $2 k, k=1,2, \ldots|\mathcal{E}|$, and thus easily inverted. Then use $\boldsymbol{K}^{-1}=\boldsymbol{R}^{-1} \boldsymbol{J}^{-1} \boldsymbol{R}^{-\top}$ to obtain

$$
\begin{align*}
\boldsymbol{K}_{2 k-1,2 k}^{-1} & =\sum_{i=1}^{2|\mathcal{E}|} \sum_{j=1}^{2|\mathcal{E}|} \boldsymbol{R}_{2 k-1, i}^{-1} \boldsymbol{J}_{i, j}^{-1} \boldsymbol{R}_{j, 2 k}^{-\top}=\frac{-1}{c_{k}}+\sum_{i=k+1}^{|\mathcal{E}|} d_{i k},  \tag{5.24}\\
\text { where } d_{i k} & =\frac{\boldsymbol{R}_{2 k-1,2 i}^{-1} \boldsymbol{R}_{2 k, 2 i-1}^{-1}-\boldsymbol{R}_{2 k-1,2 i-1}^{-1} \boldsymbol{R}_{2 k, 2 i}^{-1}}{c_{i}}
\end{align*}
$$

Note in the above $\boldsymbol{A}_{i, k}^{-1}=\left[\boldsymbol{A}^{-1}\right]_{i, k}$ so inversion is performed first.

## Prefactoring

Consider the rows and columns of $\boldsymbol{K}$ corresponding to an edge added during plane triangulation (Section 5.3.1). Reorder $\boldsymbol{K}$ to bring those rows and columns to the top left, so that they form the $\boldsymbol{a}, \boldsymbol{b}$, and $c$ of (5.17). Since the disagreement cost of a triangulation edge is zero, we now have a unity pivot: $c=e^{0}=1$. This has two advantageous consequences:

Size reduction: The unity pivot does not affect the value of the partition function. Since we are not interested in the marginal probability of triangulation edges (which after all are not part of the original model), we do not need $\boldsymbol{a}$ or $\boldsymbol{b}$ either, once we have computed the Schur complement (5.19). We can therefore discard the first two rows and first two columns of $\boldsymbol{K}$ after factoring (5.17). Factoring out all triangulation edges in this fashion reduces the size of $\boldsymbol{K}$ (resp. $\boldsymbol{R}$ and $\boldsymbol{J}$ ) to range only over the edges of the original model graph. This reduces storage requirements and speeds up subsequent computation of the inverse (Figure 5.8(b)).

Boolean closure: The unity pivot eliminates the division from the Schur complement (5.19). In fact we show below that applying (5.18) to prefactor $\boldsymbol{H}-\boldsymbol{H}^{\top}$ yields a Schur complement that can be expressed as $\boldsymbol{H}^{\prime}-\boldsymbol{H}^{\prime}$, where $\boldsymbol{H}^{\prime}$ is again a Boolean matrix. This closure property allows us to simply prefactor triangulation edges directly out of $\boldsymbol{H}$ without explicitly constructing $\boldsymbol{K}$.

Specifically, let $\boldsymbol{K}=\boldsymbol{H}-\boldsymbol{H}^{\top}$ for a proto-Kasteleyn matrix $\boldsymbol{H}$ with elements in $\{0,1\}$. Without loss of generality, assume that $\boldsymbol{H}$ and its transpose are disjoint, i.e., have no non-zero elements in common: $\boldsymbol{H} \odot \boldsymbol{H}^{\top}=\mathbf{0}$, where $\odot$ denotes Hadamard (element-wise) multiplication. Algorithm 4 respects this condition; violations would cancel in the construction of $\boldsymbol{K}$ anyway. Expressing $\boldsymbol{H}$ as

$$
\boldsymbol{H}=\left[\begin{array}{cc|c}
0 & 1 & \boldsymbol{a}_{1}^{\top}  \tag{5.25}\\
0 & 0 & \boldsymbol{b}_{\mathbf{1}}^{\top} \\
\hline \boldsymbol{a}_{\mathbf{2}} & \boldsymbol{b}_{\mathbf{2}} & \boldsymbol{C}_{\mathbf{1}}
\end{array}\right]
$$

we can write $\boldsymbol{K}=\boldsymbol{H}-\boldsymbol{H}^{\top}$ as (5.17) with $\boldsymbol{a}=\boldsymbol{a}_{\mathbf{1}}-\boldsymbol{a}_{\mathbf{2}}, \boldsymbol{b}=\boldsymbol{b}_{\mathbf{1}}-\boldsymbol{b}_{\mathbf{2}}, c=1$, and $\boldsymbol{C}=\boldsymbol{C}_{\mathbf{1}}-\boldsymbol{C}_{\mathbf{1}}^{\top}$. The Schur complement (5.19) then becomes

$$
\begin{align*}
C^{\prime} & =C_{1}-C_{1}^{\top}+\left(b_{1}-b_{2}\right)\left(a_{1}-a_{2}\right)^{\top}-\left(a_{1}-a_{2}\right)\left(b_{1}-b_{2}\right)^{\top} \\
& =\left(C_{1}+b_{1} a_{1}^{\top}+b_{2} a_{2}^{\top}+a_{1} b_{2}^{\top}+a_{2} b_{1}^{\top}\right)-\left(C_{1}^{\top}+a_{1} b_{1}^{\top}+a_{2} b_{2}^{\top}+b_{2} a_{1}^{\top}+b_{1} a_{2}^{\top}\right) \\
& =H^{\prime}-H^{\prime \top} \tag{5.26}
\end{align*}
$$

where

$$
\begin{equation*}
H^{\prime}=C_{1}+b_{1} a_{1}^{\top}+b_{2} a_{2}^{\top}+a_{1} b_{2}^{\top}+a_{2} b_{1}^{\top} \tag{5.27}
\end{equation*}
$$

It remains to show that $\boldsymbol{H}^{\prime}$ is a Boolean matrix, ie all its elements are in $\{0,1\}$. By definition of $\boldsymbol{H}$ (5.25), all elements of $\boldsymbol{C}_{\mathbf{1}}, \boldsymbol{a}_{\mathbf{1}}, \boldsymbol{a}_{\mathbf{2}}, \boldsymbol{b}_{\mathbf{1}}, \boldsymbol{b}_{\mathbf{2}}$ are in $\{0,1\}$, and by closure of multiplication in $\{0,1\}$, so are their products. Thus an element of $\boldsymbol{H}^{\prime}$ will be in $\{0,1\}$ iff it is non-zero in at most one term on the right-hand side of (5.27), or equivalently iff all pairs formed from the five terms in question are disjoint. We show that this is the case:

- Because $\boldsymbol{H} \odot \boldsymbol{H}^{\top}=\mathbf{0}$, we know that neither $\boldsymbol{a}_{\mathbf{1}}$ and $\boldsymbol{a}_{\mathbf{2}}$ nor $\boldsymbol{b}_{\mathbf{1}}$ and $\boldsymbol{b}_{\mathbf{2}}$ can have any non-zero elements in common, so $\boldsymbol{b}_{\mathbf{1}} \boldsymbol{a}_{\mathbf{1}}^{\top}$ and $\boldsymbol{b}_{\mathbf{2}} \boldsymbol{a}_{\mathbf{2}}^{\top}$ are disjoint, as are $\boldsymbol{a}_{\mathbf{1}} \boldsymbol{b}_{\mathbf{2}}^{\top}$ and $\boldsymbol{a}_{\mathbf{2}} \boldsymbol{b}_{\mathbf{1}}^{\top}$.
- By construction of $\boldsymbol{H}$ (Algorithm 4), $\boldsymbol{a}_{\mathbf{1}}$ and $\boldsymbol{b}_{\mathbf{1}}$ (resp. $\boldsymbol{a}_{\mathbf{2}}$ and $\boldsymbol{b}_{\mathbf{2}}$ ) can only have an element in common if the dual nodes on both sides of the corresponding edge are members of the same clique. This cannot happen because we explicitly ensure that the graph becomes biconnected during plane triangulation (Section 5.3.1), so that an edge cannot border the same face of the model graph on both sides. Thus all four outer products in (5.27) are pairwise disjoint.
- Finally, each outer product in (5.27) is disjoint from $\boldsymbol{C}_{\mathbf{1}}$ as long as the edges being factored out do not form a cut of the model graph (i.e., cycle of the dual). We are prefactoring only edges added during triangulation. These edges will form a cut only if the model graph was disconnected prior to triangulation. However this cannot happen, because we deal with (and eliminate) this possibility during earlier preprocessing (Section 5.1.3).

In summary, all five terms on the right-hand side of (5.27) are pairwise disjoint. Thus the Schur complement $\boldsymbol{H}^{\prime}$ is a Boolean matrix as well, and can be computed from $\boldsymbol{H}$ (5.25) very efficiently by replacing the additions and multiplications in (5.27) with bitwise OR and AND operations, respectively. As long as further triangulation edges remain in $\boldsymbol{H}^{\prime}$, we then set $\boldsymbol{H}:=\boldsymbol{H}^{\prime}$ and iteratively apply (5.25) and (5.27) so as to prefactor them out as well.

### 5.4 Conclusion

This chapter described an alternative algorithmic framework for efficient exact inference in binary graphical models, which replaces the submodularity constraint of graph cut methods with a planarity constraint. In contrast to previous approaches, our construction uses the complementary mapping (Section 3.2.2) between cuts in the model graph $G$ (even subgraphs in $G^{*}$ ) and perfect matchings in its expanded dual graph $G_{E}^{*}$. The existence of two distinct tractable frameworks for inference in binary graphical models implies a more powerful hybrid. Consider a graph each of whose biconnected components is either planar or submodular. In its entirety, this graph may be neither planar nor submodular, yet efficient exact inference in it is clearly possible by applying the appropriate framework to each component, then combining the results (Section 5.1.4). Can this hybrid approach be extended to cover less obvious situations?

Although we have only dealt with binary-labeled models, the Ising graphical model can also implement $\alpha$-expansion moves and $\alpha$ - $\beta$-swaps [12], allowing it to deal with multi-labeled models.

Our exact algorithms can all be extended to non-planar graphs, at a cost exponential in the genus of the embedding. These extensions may prove of great practical value for graphs that are "almost" planar. Examples of such graphs include road networks (where edge crossings arise from overpasses without on-ramps) and graphs describing the tertiary structure of proteins [135].

## Chapter 6

## Planar Ising Graphical Models: Experiments

Unlike graph cut methods, the algorithms described in Chapter 5 allow us to perform penalized maximumlikelihood (Section 6.1.1) as well as maximum-margin (Section 6.1.2) parameter estimation in our Ising graphical model. For prediction we can employ marginal posterior probabilities as well as MAP states.

We demonstrate the suitability of our approach to CRF parameter estimation on 3 tasks whose underlying model graph is planar. We begin with two computer vision problems: the synthetic binary image denoising task of Kumar and Hebert [76, 77] (Section 6.2), and boundary detection in noisy masks from the GrabCut Ground Truth image segmentation database [104] (Section 6.3). Finally, we apply Ising CRFs to estimate territory in the game of go by employing graph abstraction and parametrisation techniques (Section 6.4).

### 6.1 Parameter Estimation

We chose to represent the Ising graphical model as a planar Conditional Random Field (CRF). We compute the disagreement costs in Equation 5.1 as $E_{k}:=\boldsymbol{\theta}^{\top} \boldsymbol{x}_{k}$, i.e., as inner products between local features (sufficient statistics) $\boldsymbol{x}_{k}$ of the modeled data at each edge $e_{k}$, and corresponding parameters $\boldsymbol{\theta}$ of the model. In a computer vision task, for example, $\boldsymbol{x}_{k}$ might be the absolute intensity difference between two adjacent pixels. We use a Markov Random Field (5.2) to model the conditional probability distribution $\mathbb{P}(\boldsymbol{y} \mid \boldsymbol{x}, \boldsymbol{\theta})$, where $\boldsymbol{x}$ denotes the union of all local features.

### 6.1.1 Maximum Likelihood

Penalized maximum-likelihood (ML) CRF parameter estimation seeks to minimize wrt. $\boldsymbol{\theta}$ the $L_{2}-$ regularized negative log-likelihood of a given target labeling $\tilde{\boldsymbol{y}}{ }^{6}$ Using (5.2) this negative log-likelihood

[^5]for the Ising graphical model becomes
\[

$$
\begin{align*}
L_{\mathrm{ML}}(\boldsymbol{\theta}) & :=\frac{1}{2} \lambda\|\boldsymbol{\theta}\|^{2}-\ln \mathbb{P}(\tilde{\boldsymbol{y}} \mid \boldsymbol{x}, \boldsymbol{\theta}) \\
& =\frac{1}{2} \lambda\|\boldsymbol{\theta}\|^{2}+E(\tilde{\boldsymbol{y}} \mid \boldsymbol{x}, \boldsymbol{\theta})+\ln Z(\boldsymbol{\theta} \mid \boldsymbol{x}) \tag{6.1}
\end{align*}
$$
\]

where $\lambda$ is the regularization parameter, $E(y)$ is the energy function of the Ising graphical model (5.1) and $\mathbb{P}(\boldsymbol{y})$ is its joint probability distribution (5.2). This is a smooth, convex, non-negative objective that can be optimized via gradient methods such as LBFGS, either in conventional batch mode [86, 95] or online [113]. The gradient of (6.1) with respect to the parameters $\boldsymbol{\theta}$ is given by

$$
\begin{equation*}
\frac{\partial}{\partial \boldsymbol{\theta}} L_{\mathrm{ML}}(\boldsymbol{\theta})=\lambda \boldsymbol{\theta}+\sum_{e_{k} \in \mathcal{E}}\left(\llbracket e_{k} \in C(\tilde{\boldsymbol{y}}) \rrbracket-\mathbb{P}\left(e_{k} \in C(\boldsymbol{y} \mid \boldsymbol{x}, \boldsymbol{\theta})\right)\right) \boldsymbol{x}_{k} \tag{6.2}
\end{equation*}
$$

where $C(\tilde{\boldsymbol{y}})$ is the cut induced by the target state $\tilde{\boldsymbol{y}}$, and $\mathbb{P}\left(e_{k} \in C(\boldsymbol{y} \mid \boldsymbol{x}, \boldsymbol{\theta})\right)$ is the marginal probability of $e_{k}$ being contained in a cut, given $\boldsymbol{x}$ and $\boldsymbol{\theta}$. We compute the latter via the inverse of the Kasteleyn matrix (5.23).

### 6.1.2 Maximum Margin

For maximum-margin (MM) parameter estimation [123] we instead minimize

$$
\begin{align*}
L_{\mathrm{MM}}(\boldsymbol{\theta}) & :=\frac{1}{2} \lambda\|\boldsymbol{\theta}\|^{2}+E(\tilde{\boldsymbol{y}} \mid \boldsymbol{x}, \boldsymbol{\theta})-\min _{\boldsymbol{y}} M(\boldsymbol{y} \mid \tilde{\boldsymbol{y}}, \boldsymbol{x}, \boldsymbol{\theta})  \tag{6.3}\\
& =\frac{1}{2} \lambda\|\boldsymbol{\theta}\|^{2}+E(\tilde{\boldsymbol{y}} \mid \boldsymbol{x}, \boldsymbol{\theta})-E(\hat{\boldsymbol{y}} \mid \boldsymbol{x}, \boldsymbol{\theta})+d(\hat{\boldsymbol{y}} \| \tilde{\boldsymbol{y}})
\end{align*}
$$

where $\hat{\boldsymbol{y}}:=\operatorname{argmin}_{\boldsymbol{y}} M(\boldsymbol{y} \mid \tilde{\boldsymbol{y}}, \boldsymbol{x}, \boldsymbol{\theta})$ is the worst margin violator, i.e., the state that minimizes, relative to a given target state $\tilde{\boldsymbol{y}}$ the margin energy

$$
\begin{equation*}
M(\boldsymbol{y} \mid \tilde{\boldsymbol{y}}):=E(\boldsymbol{y})-d(\boldsymbol{y} \| \tilde{\boldsymbol{y}}) \tag{6.4}
\end{equation*}
$$

where $d(\cdot \| \cdot)$ is some measure of divergence in state space. We choose $d$ to be a weighted Hamming distance between induced cuts:

$$
\begin{equation*}
d(\boldsymbol{y} \| \tilde{\boldsymbol{y}}):=\sum_{e_{k} \in \mathcal{E}} \llbracket \llbracket e_{k} \in C(\boldsymbol{y}) \rrbracket \neq \llbracket e_{k} \in C(\tilde{\boldsymbol{y}}) \rrbracket \rrbracket v_{k} \tag{6.5}
\end{equation*}
$$

where $v_{k}>0$ are constant weighting factors (in the simplest case: all ones) on the edges of $G$. It is now easy to verify that the margin energy (6.4) is implemented (up to a shift that depends only on $\tilde{\boldsymbol{y}}$ ) by an isomorphic Ising graphical model with disagreement costs

$$
\begin{equation*}
\bar{E}_{k}:=E_{k}+\left(2 \llbracket e_{k} \in C(\tilde{y}) \rrbracket-1\right) v_{k} \tag{6.6}
\end{equation*}
$$

Thus we can efficiently find the worst margin violator $\hat{\boldsymbol{y}}$ by computing the ground state (Section 5.2.3) of this isomorphic Ising graphical model. Thomas and Middleton [126] employ a similar approach to obtain the ground state from a given state $\tilde{\boldsymbol{y}}$ by setting up an isomorphic Ising model with disagreement
costs $\bar{E}_{k}:=E_{k}\left(1-2 \llbracket e_{k} \in C(\tilde{\boldsymbol{y}}) \rrbracket\right)$. Using (6.6), the maximum-margin objective (6.3) can be expressed as

$$
\begin{equation*}
L_{\mathrm{MM}}(\boldsymbol{\theta}):=\frac{1}{2} \lambda\|\boldsymbol{\theta}\|^{2}+\bar{E}(\tilde{\boldsymbol{y}} \mid \boldsymbol{x}, \boldsymbol{\theta})-\bar{E}(\hat{\boldsymbol{y}} \mid \boldsymbol{x}, \boldsymbol{\theta}), \tag{6.7}
\end{equation*}
$$

where $\bar{E}(\hat{\boldsymbol{y}} \mid \boldsymbol{x}, \boldsymbol{\theta})$ is the energy of the worst margin violator. This objective is convex but non-smooth; its gradient is

$$
\begin{equation*}
\frac{\partial}{\partial \boldsymbol{\theta}} L_{\mathrm{MM}}(\boldsymbol{\theta})=\lambda \boldsymbol{\theta}+\sum_{e_{k} \in \mathcal{E}}\left(\llbracket e_{k} \in C(\tilde{\boldsymbol{y}}) \rrbracket-\llbracket e_{k} \in C(\hat{\boldsymbol{y}}) \rrbracket\right) \boldsymbol{x}_{k} \tag{6.8}
\end{equation*}
$$

In other words, local features $\boldsymbol{x}_{k}$ are multiplied by one of $\{-1,0,1\}$, depending on the membership of edge $e_{k}$ in the cuts induced by $\tilde{\boldsymbol{y}}$ and $\hat{\boldsymbol{y}}$, respectively. We can minimize (6.7) via bundle methods, such as the bundle trust (BT) algorithm [109], making use of the convenient lower bound: $L_{\mathrm{MM}}(\boldsymbol{\theta}) \geq 0, \forall \boldsymbol{\theta}$. This lower bound holds because in Equation 6.7

$$
\begin{align*}
\bar{E}(\tilde{\boldsymbol{y}} \mid \boldsymbol{x}, \boldsymbol{\theta}) & =\bar{E}(\tilde{\boldsymbol{y}} \mid \boldsymbol{x}, \boldsymbol{\theta})-d(\tilde{\boldsymbol{y}} \mid \tilde{\boldsymbol{y}})  \tag{6.9}\\
& =M(\tilde{\boldsymbol{y}} \mid \tilde{\boldsymbol{y}}, \boldsymbol{x}, \boldsymbol{\theta}) \\
& \geq \min _{\boldsymbol{y}} M(\boldsymbol{y} \mid \tilde{\boldsymbol{y}}, \boldsymbol{x}, \boldsymbol{\theta})=\bar{E}(\hat{\boldsymbol{y}} \mid \boldsymbol{x}, \boldsymbol{\theta})
\end{align*}
$$

### 6.2 Synthetic Binary Image Denoising

Kumar and Hebert [76, 77] developed an image denoising benchmark problem for binary-labeled CRFs based on four hand-drawn $64 \times 64$ pixel images (Figure 6.1, top row). We created 50 instances of each image corrupted with pink noise, produced by convolving a white noise image (all pixels i.i.d. uniformly random) with a Gaussian density of one pixel standard deviation. ${ }^{7}$ Original and pink noise images were linearly mixed using signal-to-noise $(\mathrm{S} / \mathrm{N})$ amplitude ratios of $1: \mathrm{n}$, where $n \in \mathbb{N}$. Figure 6.1 shows samples of the resulting noisy instances for $\mathrm{S} / \mathrm{N}$ ratios of $1: 5$ (row 2) and 1:6 (row 4).

We then employed a grid Ising CRF to denoise the images, with edge disagreement energies set to $E_{i j}:=\left[1,\left|x_{i}-x_{j}\right|\right]^{\top} \boldsymbol{\theta}$, where $x_{i}$ is the pixel intensity at node $i$. The perimeter of the grid was connected to a bias node with constant pixel intensity $x_{0}:=0$ and fixed label $y_{0}:=0$. Edges to the bias had their own parameters, yielding a CRF with four parameters and up to (for a $64 \times 64$ grid) 4097 nodes and 8316 edges.

The CRFs were trained by maximum margin (MM) and maximum likelihood (ML) parameter estimation (Section 6.1) on the 50 noisy instances derived from the first image (Figure 6.1, left column) only. ${ }^{8}$ We assessed the quality of the obtained parameters by determining (via the method of Section 5.2.3) the maximum a posteriori (MAP) states

$$
\begin{equation*}
\boldsymbol{y}^{*}:=\underset{y}{\operatorname{argmax}} \mathbb{P}(\boldsymbol{y} \mid \boldsymbol{x}, \boldsymbol{\theta})=\underset{\boldsymbol{y}}{\operatorname{argmin}} E(\boldsymbol{y} \mid \boldsymbol{x}, \boldsymbol{\theta}) \tag{6.10}
\end{equation*}
$$

of the trained CRF for all 150 noisy instances of the other three images. Interpreting these MAP states

[^6]

Figure 6.1: Denoising of binary images by maximum-margin training of grid Ising CRFs. Row 1: original images [76, 77]. Rows 2 and 3: examples of images mixed with pink noise in a 1:5 ratio and their reconstruction via MAP state of $64 \times 64$ Ising grid CRF. Rows 4 and 5: examples of images mixed with pink noise in a 1:6 ratio and their reconstruction via MAP state of $64 \times 64$ Ising grid CRF. Only the left-most image was used for training (via max-margin CRF parameter estimation, $\lambda=2$ ); the other 3 images were used for reconstruction.

| Train Method | Patch Size | Train Time (s) | Edge Error (\%) | Node Error (\%) |
| :---: | :---: | :---: | :---: | :---: |
| MM | $64 \times 64$ | 490.4 | 1.15 | 2.10 |
|  | $32 \times 32$ | 174.7 | 1.16 | 2.15 |
|  | $16 \times 16$ | 91.4 | 1.12 | 1.98 |
|  | $8 \times 8$ | $\mathbf{7 8 . 1}$ | $\mathbf{1 . 0 9}$ | $\mathbf{1 . 8 3}$ |
|  |  | 5468.2 | 1.11 | 1.92 |

Table 6.1: Performance comparison of parameter estimation methods on the denoising task with image reconstruction via MAP on the full $(64 \times 64)$ model and images. Patch Size is the size of patches used during training. Train Time is the time required to train on 50 images. The minimum in each result column is boldfaced.
as attempted reconstructions of the original images, we then calculated the reconstruction error rates for both nodes and edges of the model.

All experiments in Section 6.2 used Blossom IV [17] and were carried out on a Linux PC with 3 GB RAM and dual Intel Pentium 4 processors running at 3.6 GHz , each with 2 MB of level 2 cache.

### 6.2.1 Noise Level

We first explore the limit of the ability of a full-size $(64 \times 64)$ MM-trained Ising grid CRF to reconstruct the test images as the noise level increases. Rows 3 and 5 of Figure 6.1 show sample reconstructions obtained from the noisy instances shown in rows 2 and 4, respectively. At low noise levels $(n<5)$ we obtain perfect reconstructions of the original images. At an $\mathrm{S} / \mathrm{N}$ ratio of $1: 5$ the first subtle errors do creep in (Figure 6.1, row 3), though less than $0.5 \%$ of the nodes and $0.3 \%$ of the edges are predicted incorrectly. At the $1: 6 \mathrm{~S} / \mathrm{N}$ ratio, these figures increase to $2.1 \%$ for nodes and $1.15 \%$ for edges, and the errors become far more noticeable (Figure 6.1, row 5). For higher noise levels ( $n>6$ ) the reconstructions rapidly deteriorate as the noise finally overwhelms the signal. It must be noted that at these noise levels our human visual system is also unable to accurately reconstruct the images.

### 6.2.2 Parameter Estimation

Next we compared MM and ML parameter estimation at the $\mathrm{S} / \mathrm{N}$ ratio of 1:6 (Figure 6.1, row 4), where reconstruction begins to break down and any differences in performance should be evident. To make ML training computationally feasible, we subdivided each training image into $648 \times 8$ patches, then trained an $8 \times 8$ grid CRF on those patches. For MM training we used the full $(64 \times 64)$ images and model, as well as $32 \times 32,16 \times 16$, and $8 \times 8$ patches, so as to assess how this subdividision impacts the quality of the model. Testing always employed the MAP state of the full model on the full images (i.e., $64 \times 64$ ).

Table 6.1 reports the edge and node errors obtained under each experimental condition. To assess statistical significance, we performed binomial pairwise comparison tests at a $95 \%$ confidence level against the null hypothesis that each of the two algorithms being compared has an equal (50\%) chance of outperforming the other on a given test image.

We found no statistically significant difference here between $8 \times 8$ CRFs trained by MM vs. ML. However, ML training took 70 times as long to achieve this, so MM training is much more preferable on computational grounds.

Counter to our expectations, the node and edge errors suggest that MM training actually works better on small ( $8 \times 8$ and $16 \times 16$ ) image patches. We believe that this is because small patches have a relatively larger perimeter, leading to better training of the bias edges. Pairwise comparison tests, however, only found the node error for the $32 \times 32$ patch-trained CRF to be significantly worse than for the smaller patches; all other differences were below the significance threshold. We can confidently state that subdividing the images into small patches did not hurt performance, and yielded much shorter training times.

The reduced performance of the $32 \times 32$ model could be an artifact of the training image we are using (first image in Figure 6.1, row 4 ). The $32 \times 32$ patches are generated by cutting the image in half both vertically and horizontally. The resulting four $32 \times 32$ patches each have a black region on two edges of their boundaries. Since the model connects the entire perimeter to a bias node, the strong presence of these black regions weakens the learned boundary parameters, which reduces the prediction accuracy.

### 6.2.3 Reconstruction

Fox and Nicholls [33] argued that the MAP state does not summarize well the information in the posterior probability distribution of an Ising model of noisy binary images, and proposed reconstruction via the marginal posterior mode (MPM) instead. For binary labels, the MPM is simply obtained by thresholding the marginal posterior node probabilities: $y_{i}:=\llbracket \mathbb{P}\left(y_{i}=1 \mid \boldsymbol{x}, \boldsymbol{\theta}\right)>0.5 \rrbracket$. However, in our Ising graphical model we have marginal posterior probabilities for edges (Section 5.3.4), and infer node states from graph cuts (Algorithm 1). Here implementing the MPM runs into a difficulty, since the edge set

$$
\begin{equation*}
\left\{e_{k} \in \mathcal{E}: \mathbb{P}\left(e_{k} \in \mathcal{C}(\boldsymbol{y} \mid \boldsymbol{x}, \boldsymbol{\theta})\right)>0.5\right\} \tag{6.11}
\end{equation*}
$$

may not be a cut of the model graph, hence may not unambiguously induce a node state. What we really need is the cut closest (in some given sense) to (6.11). For this purpose we formulate the state $\boldsymbol{y}^{\boldsymbol{+}}$ with the maximal minimum marginal posterior $\left(\mathrm{M}^{3} \mathrm{P}\right)$ :

$$
\boldsymbol{y}^{+}:=\underset{\boldsymbol{y}^{\prime}}{\operatorname{argmax}} \min _{e_{k} \in \mathcal{E}}\left\{\begin{align*}
\mathbb{P}\left(e_{k} \in \mathcal{C}(\boldsymbol{y} \mid \boldsymbol{x}, \boldsymbol{\theta})\right) & \text { if } e_{k} \in C\left(\boldsymbol{y}^{\prime}\right),  \tag{6.12}\\
1-\mathbb{P}\left(e_{k} \in C(\boldsymbol{y} \mid \boldsymbol{x}, \boldsymbol{\theta})\right) & \text { otherwise. }
\end{align*}\right.
$$

In other words, the $\mathrm{M}^{3} \mathrm{P}$ state (6.12) is induced by the cut whose edges (and those of its complement) have the largest minimum marginal probability. We can efficiently compute $\boldsymbol{y}^{+}$as follows:

1. Find the maximum-weight spanning tree $T\left(\mathcal{V}, \mathcal{E}^{+}\right)$of the model graph $G(\mathcal{V}, \mathcal{E})$ with edge weights $\left|\mathbb{P}\left(e_{k} \in C(\boldsymbol{y} \mid \boldsymbol{x}, \boldsymbol{\theta})\right)-0.5\right|$. This can be done in $O(|\mathcal{E}| \log |\mathcal{E}|)$ time.
2. Run Algorithm 1 on $T\left(\mathcal{V}, \mathcal{E}^{+}\right)$to find $\boldsymbol{y}^{+}$as the state induced by the edge set (6.11).

Since $T\left(\mathcal{V}, \mathcal{E}^{+}\right)$is a tree, it contains no cycles, so Algorithm 1 will unambiguously identify the $\mathrm{M}^{3} \mathrm{P}$ state.


Figure 6.2: Image reconstructions on the denoising task with the Ising CRF trained via MM on $8 \times 8$ patches. (a) original images [76, 77]. (b) examples of images mixed with pink noise in a 1:6 ratio. (c) examples of MAP reconstruction of the full $64 \times 64$ image. (d) examples MAP reconstruction of $8 \times 8$ patches. (e) examples $\mathrm{M}^{3} \mathrm{P}$ reconstruction of $8 \times 8$ patches. (f) examples QPBO reconstruction of $8 \times 8$ patches, where gray regions could not be reconstructed due to unlabeled nodes.

| Test Method | Patch Size | Test Time (s) | Edge Error (\%) | Unl. Edges (\%) | Node Error (\%) | Unl. Nodes (\%) |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| MAP | $8 \times 8$ | 3.3 | 1.96 | - | 3.31 | - |
| $\mathrm{M}^{3} \mathrm{P}$ |  | 397.5 | 1.95 | - | 3.32 | - |
| QPBO |  | 475.3 | 2.70 | 1.99 | 3.75 | 1.94 |
| QPBOP |  | 450.4 | 1.96 | $<0.01$ | 3.31 | $<0.01$ |


| Test Method | Patch Size | Test Time (s) | Edge Error (\%) | Unl. Edges (\%) | Node Error (\%) | Unl. Nodes (\%) |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| MAP | $16 \times 16$ | 3.7 | 1.37 | - | 2.33 | - |
| QPBO |  | 154.7 | 3.66 | 5.18 | 4.21 | 5.09 |
| QPBOP |  | 159.5 | 1.41 | 0.08 | 2.35 | 0.08 |
| MAP | $32 \times 32$ | 4.2 | 1.14 | - | 1.89 | - |
| QPBO |  | 51.1 | 21.20 | 41.52 | 21.40 | 41.44 |
| QPBOP |  | 988.4 | 7.12 | 12.37 | 7.77 | 12.35 |
| MAP | $64 \times 64$ | 5.2 | 1.09 | - | 1.83 | - |
| QPBO |  | 24.7 | 49.83 | 99.66 | 49.83 | 99.63 |
| QPBOP |  | 24031.5 | 43.94 | 87.66 | 44.60 | 87.51 |

Table 6.2: Comparison of reconstruction methods on the denoising task, using the parameters of an MM-trained $8 \times 8$ CRF. Patch Size is the size of patches used during testing. Test Time is the time required to reconstruct all 150 images. Unlabeled nodes and edges count as half errors. Unlabeled percentages are not applicable to MAP and $\mathrm{M}^{3} \mathrm{P}$, since these methods provide a full labeling. The minimum in each result column is boldfaced.

## Algorithm Comparison

Table 6.2 lists the reconstruction errors achieved on the denoising task using the parameters of the MMtrained $8 \times 8$ CRF which gave the best performance in Section 6.2.2. Note that we obtained comparable results for ML training on $8 \times 8$ patches and MM training on the full images.

We were able to compare MAP $v s . \mathrm{M}^{3} \mathrm{P}$ states only when reconstructing $8 \times 8$ patches (Table 6.2 top). Since the $M^{3} P$ state (6.12) requires calculation of the edge marginals, it was not feasible to compute it for larger patches. To highlight the difficulty of this task, we also compared to quadratic pseudo-boolean optimization (QPBO) and its variant QPBOP (with probing) [43, 68, 105]. These methods extend the classical graph cut algorithm (Section 4.3.5) by providing a part of an optimal node state for graphical models with non-submodular energy functions. Columns 5 and 7 of Table 6.2 show the percentage of unlabeled edges and nodes, respectively. To be fair on QPBO and QPBOP we count unlabeled nodes and edges as half errors.

The binomial pairwise comparison test at a $95 \%$ confidence level reveals no statistically significant difference between MAP and $\mathrm{M}^{3} \mathrm{P}$ states. Thus in our framework we are not able to reproduce the advantage that Fox and Nicholls [33] report for MPM. Furthermore, the MAP state on the entire image clearly outperforms any reconstruction from $8 \times 8$ patches in terms of both edge and node error (Table 6.2 and Figures 6.2 (c) and 6.2(d)). The impressive scalability of the MAP state computation via blossom-shrinking (Section 5.2.3) thus in practice overrules any possible theoretical advantage that reconstructions via marginal probabilities may have.

The binomial pairwise comparison test at a $95 \%$ confidence level also reveals no statistically significant difference between MAP and QPBOP states. QPBOP manages to label nearly all the nodes on this task, thus its reconstruction is close to optimal. However, QPBO cannot label $1.94 \%$ of nodes (Figure 6.2(f)) making it significantly worse than QPBOP and MAP.

## Scalability

When reconstructing on patches larger than $8 \times 8$, the MAP state significantly outperforms QPBO and QPBOP in both edge and node error (Table 6.2 bottom). As the patch size increases, the reconstruction quality of QPBO and QPBOP rapidly decreases, since the percentage of unlabeled edges and nodes becomes high.

MAP is orders of magnitude faster than all of the control methods. Its advantage is most evident when reconstructing the full images. Interestingly, QPBO becomes relatively faster as the patch size increases. This is because in larger patches QPBO has less chance of resolving unlabeled nodes, therefore it "gives up" more quickly.

### 6.3 Boundary Detection

We designed a boundary detection task based on images from the GrabCut Ground Truth image segmentation database [104]. We took $100 \times 100$ pixel subregions of images that depicted a segmentation boundary, and corrupted the segmentation mask with pink noise, produced by convolving a white noise image (all pixels i.i.d. uniformly random) with a Gaussian density with one pixel standard deviation.

We then employed a planar Ising graphical model to recover the original boundary. We used a $100 \times 100$ square grid with one additional edge fixed to a high energy, encoding prior knowledge that the bottom left and top right corners of the image depict different regions. We set the energy of the other edges to $E_{i j}:=\left[1,\left|x_{i}-x_{j}\right|\right]^{\top} \boldsymbol{\theta}$, where $x_{i}$ is the pixel intensity at node $i$. We train unsupervised on the noisy mask (Figure 6.3(b)), then use the trained CRF's MAP state for prediction (Figure 6.3(d)). ${ }^{9}$ We did not employ a bias node for this task, and simply set the regularization constant to $\lambda=1$. All experiments were carried out on an Apple MacBook laptop with 2.2 GHz Intel Core2 Duo processor and 4 GB RAM.

Note that this is a large model with 10000 nodes and 19801 edges. Due to the sparsity of the associated Kasteleyn matrix, we can nonetheless compute its log partition function in less than half a CPU second, using the SuperLU sparse matrix factorisation package [19]. However it then takes over 200 CPU seconds to solve for the marginals, making optimization of the ML objective (6.1) computationally unattractive. By contrast, we can compute the ground state of this model with Blossom V [67] via the algorithm described in Section 5.2 in less than 100 CPU milliseconds. We can therefore efficiently minimize the MM objective (6.3).

We estimated the smallest $\mathrm{S} / \mathrm{N}$ ratio of the form 1:n for which we obtained a good segmentation. Depending on the image, this occurred for $n=7$ or 8 . Figure 6.3 (right column) shows that at these noise levels our approach is capable of recovering the original segmentation boundary quite well, with

[^7]

Figure 6.3: Boundary detection by maximum-margin training of grid Ising CRFs. (a) original image. (b) noisy mask for $\mathrm{S} / \mathrm{N}$ ratio of 1:8 (rows 1 and 2) resp. 1:7 (rows 3 and 4). (c) ground truth segmentation. (d) MAP segmentation obtained from the Ising grid CRF trained on the noisy mask.
less than $1 \%$ of nodes mislabeled. For $\mathrm{S} / \mathrm{N}$ ratios of $1: 9$ and lower the system is unable to locate the boundary; for $\mathrm{S} / \mathrm{N}$ ratios of 1:6 and higher we obtain perfect reconstruction. Again this corresponds closely to our human ability to visually locate the segmentation boundary accurately.

In further scalability tests we were able to use Blossom V to calculate the MAP state for a random $1500 \times 1500$ Ising grid with perimeter bias (a graph with 2.25 million nodes and 4.5 million edges) in 200 CPU seconds. It is worth mentioning that the approach of Pardella and Liers [98] is also able to compute the MAP state of such problem instances. Despite the similarity, there are a few key differences between the two approaches:

- Our method is a generalisation that can handle arbitrary planar graphs.
- Our construction uses the complementary mapping (Section 3.2.2) between even subgraphs in $G_{0}$ and perfect matchings in the decorated graph $G_{d}$, instead of the direct mapping used in [98].
- We use Blossom V instead of Blossom IV. Although both implementations have the same worstcase asymptotic time complexity, Blossom V is usually considered faster in practice [67].

We were also able to use Blossom V to compute the log partition function for a $600 \times 600$ image (a graph with 0.36 million nodes and 0.72 million edges) in 100 CPU seconds. A vector of edge marginals for the latter, however, would take over 100 CPU hours to compute.

### 6.4 Territory Prediction in Go

We apply planar Ising CRFs to territory prediction in the game of Go. We propose a two-stage graph reduction of the position on the Go board grid, with the first stage used for parameter estimation and the second for inference. We use our polynomial-time algorithms for both model parameter estimation and computation of the MAP state for territory prediction.

### 6.4.1 The Game of Go in Machine Learning

Go, known as wei-qi in China and baduk in Korea, is a board game that originated in China over 4000 years ago. It is played on a $19 \times 19$ grid, though smaller boards are commonly used by beginners and computers. 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 or killed) if all its empty neighbours (liberties) are occupied by opponent stones. Neighboring stones of the same color form a contiguous block. Note that a block can contain a single stone. Stones in a block are surrounded and captured as one. Players aim to create alive blocks, i.e., blocks that cannot be captured. The area occupied by these alive blocks counts as their final territory. The winner of the game is the player who controls the most territory at the end.

## Computer Go

From these simple rules emerges a game of greater subtlety and strategic complexity than Chess. It is not surprising then that there is great interest in creating computer programs that can play go. Unlike


Figure 6.4: (a) A typical $9 \times 9$ endgame board position corresponding to a grid graph $G(\mathcal{V}, \mathcal{E})$. The player's final territory is his alive stones plus the area shaded with his color. Stones (6), (12) and (1) are dead; all other stones are alive. (b) Corresponding common fate graph $G_{f}\left(\mathcal{V}_{f}, \mathcal{E}_{f}\right)$. (c) Corresponding block graph $G_{b}\left(\mathcal{V}_{b}, \mathcal{E}_{b}\right)$. ○ represent stones, $\square$ represent surrounds, and $\diamond$ are neutral. Node size reflects the number of points in the block. Dashed red lines indicate nodes of the group graph $G_{g}\left(\mathcal{V}_{g}, \mathcal{E}_{g}\right)$.

Chess, however, go is not amenable to conventional game tree search. This is because its branching factor of about 200 does not permit search tree expansion to a depth anywhere near that employed by human players, who routinely look ahead dozens of moves. Static position evaluation is an even greater challenge, since (again in contrast to Chess) individual stones carry little information. It is the details of the shape and arrangement of entire groups of blocks of stones that determine their fate. At the same time, that fate can depend on the precise location of a faraway individual stone, so it does not suffice to simply combine the results of local searches.

Due to these difficulties, go programs of the past took an intensive knowledge-based approach, combining many local searches with pattern matching on configurations of stones. The patterns were typically handcrafted or harvested from search techniques. Recently, Monte-Carlo game-tree searches have shown great promise on the small $(9 \times 9)$ go board, and considerable progress has been made on $19 \times 19$ boards. These Monte Carlo go programs critically depend on very fast evaluation of their leaf nodes.

Go is the second-most researched board game after Chess [91]. Despite all this activity, go remains a paradigmatic unsolved AI problem, with the best computer programs playing at the level of a strong human amateur.

## Machine Learning and Go

Go is a very interesting domain for machine learning: it is a finite, deterministic, discrete, zero-sum, perfect-information game, and thus represents the closest we can come to a fully observable real-world domain. Yet the domain is extremely complex — even its subproblems (e.g., ladders and endgame) are known to be in PSPACE [18, 140]. It is generally acknowledged that the traditional knowledge engineering paradigm to computer $g o$ is at a dead end, and future advances are expected to come from machine learning and Monte Carlo techniques. Millions of recorded games between highly proficient human players, including professionals, are available on the Internet. Now the challenge is to develop techniques that effectively use all this data.

## Territory Prediction

An important subproblem in go is territory prediction: given a board position, determine which player controls each intersection. A player controls an empty intersection if his opponent cannot place stones at that intersection without them being eventually captured ${ }^{10}$ (shaded empty intersections in Figure 6.4(a)). A player controls a stone if that stone cannot be captured, making it alive ${ }^{10}$ (unshaded stones in Figure 6.4(a)); otherwise it is $d e a d^{10}$ (shaded stones in Figure 6.4(a)).

Since the game is won by whoever controls more territory at the end, territory prediction amounts to static evaluation and is therefore inherently difficult. In fact, a game of go ends as soon as both players agree on their territory prediction - so by definition they disagree on it throughout the game.

Even the most sophisticated exact methods cannot exhaustively solve board positions that contain more than 18 enclosed empty points (this is the case in $41 \%$ of the games in our $9 \times 9$ training set), and

[^8]take up to 200 seconds for a game they can solve [92, 93]. Therefore a heuristic approach is called for. Schraudolph et al. [112] use temporal difference learning in convolutional neural networks to predict territory; Wu and Baldi [142] employ a recursive neural network architecture. van der Werf et al. build a series of classifiers that learn to estimate potential territory [2004], score final positions [2005], and predict life and death [2005].

### 6.4.2 Go Positions as Graph Abstraction Hierarchies

Go positions are most naturally represented as graphs, suggesting the use of graphical models. These graphs can have various levels of abstraction, forming an abstraction hierarchy [35]. We will use different levels of that hierarchy for territory modelling and prediction.

## Grid Graph

The go board defines a square grid graph $G(\mathcal{V}, \mathcal{E})$ whose intersections can be in one of three states: black, white, or empty (Figure 6.4(a)). Stern et al. [119] model the territory distribution with an MRF defined on $G$. Their system uses sampling and Loopy Belief Propagation (LBP) to learn just 6 parameters for node and edge potentials. We believe this is insufficient to model the complex interaction between stones in $g o$.
van der Werf et al. [130, 131, 132] obtain better prediction accuracy by avoiding graphical models altogether. However, this forces them to describe each node of the grid with a plethora of hand-crafted, domain-specific features, which are prone to cause overfitting.

## 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 the grid graph does. Worse, it is counterproductive: interactions between neighboring blocks, which are essential to the game, may appear as hard-to-model long-range interactions at the grid level. It thus makes sense to merge all stones in a block into a single node, and move information such as the size and shape of the block - which would otherwise be lost —into the block's node features (Figure 6.4(b)). We will refer to such blocks as stone blocks. Graepel et al. [40] use this common fate $\operatorname{graph} G_{f}\left(\mathcal{V}_{f}, \mathcal{E}_{f}\right)$ to train SVMs to discriminate between good and bad moves. One of the strongest learning-based go program uses a neural network that also learns from a common fate graph representation [26, 27].

## Block Graph

The common fate graph still represents empty grid intersections individually. Regions of empty intersections do not have the common fate property; they can be (and usually are) divided up between the players. Therefore collapsing them as well as stone blocks results in too impoverished a representation. On the other hand, not collapsing them produces unwieldy, large graphs, especially early in the game. We compromise by grouping empty intersections into three types before collapsing regions of the same type. Specifically, we construct our block graph as follows:

Classify the intersections of the grid graph $G(\mathcal{Y}, \mathcal{E})$ (Figure 6.4(a)) into black stones $\bullet$, white stones $\bigcirc$, and three types of empty intersections: black surround $\mathbf{\square}$, white surround $\square$ and neutral $\diamond$. To determine the type of all empty intersections we use a flood fill algorithm to find the Manhattan distances from each intersection to the nearest black resp. white stone. Each intersection is then colored $\mathbf{\square}, \stackrel{\otimes}{ }$, or $\square$ depending on whether the distance to the nearest black stone is less than, equal to, or greater than the distance to the nearest white stone. For instance, intersection 19 on the grid graph has neutral type since its closest white stone (9) and closest black stones (12) and (1) all lie a Manhattan distance of 2 away (Figure 6.4(a)). Finally, we collapse all contiguous regions of intersections of the same type to obtain the block graph $G_{b}\left(\mathcal{V}_{b}, \mathcal{E}_{b}\right)$ (Figure 6.4(c)). From now on, blocks that are composed entirely of empty intersections will be called empty blocks.

Our block graph provides a more succinct representation than the common fate graph, yet by classifying empty blocks into three types preserves the kind of information needed for predicting territory. For instance, consider Figure 6.4(c): (12 and (1) are dead because they are in the opponent's territory. However, if there were more friendly (black) stones in the neighbourhood then they could be alive. It is important to encode this potential for a status change (aji in go terminology), as it could affect our prediction for neighbouring opponent blocks. We encode this by having black surround blocks (blocks 16 and 18 in Figure 6.4(c)). The block graph also concisely encodes the notion of eyes - small empty regions surrounded by stones of a single color. Eyes are a literally vital concept in $g o$, because a stone block with two or more eyes cannot be captured.

## Group Graph

Stone blocks of one color that share the same surround are very unlikely to end up with different labels - in fact we have not found a single instance of such an occurrence in our set of $180009 \times 9$ games. Since such groups of blocks often share the same fate (e.g., all live or all die), we can group them together (dashed lines in Figure 6.4(c)). We use the resulting group graph $G_{g}\left(\mathcal{V}_{g}, \mathcal{E}_{g}\right)$ for prediction; for parameter estimation we use the block graph, because the fate of a group very much depends on the shape of its constituent blocks, albeit shared amongst them.

The grid $G$, block $G_{b}$, and group $G_{g}$ graphs are the first three levels in a 5 -stage graph abstraction hierarchy of go positions introduced by Friedenbach, Jr. [35], who also presents an algorithm for incrementally updating such a hierarchy. This hierarchical graph representation is intended to model how proficient go players evaluate positions. Integrating yet higher levels of Friedenbach's abstraction hierarchy into a go learning program could be a promising direction for future work.

We note that the idea of graph abstraction hierarchies is also found in other fields, such as computer vision. For example, an image can be decomposed into homogenous regions (e.g., sky, grass, building etc.), while each region can be decomposed into separate objects and finally each object is composed of individual pixels.

## Planarity

We note that the go grid $G$, which is drawn on a plane board, is by definition planar. Since common fate graph, block graph, and group graph are all derived from the grid graph via edge contractions (which preserve planarity) they are also planar. This allows us to employ our exact polynomial-time algorithms
from Chapter 5 for both model parameter estimation on the block graph and territory prediction on the group graph.

### 6.4.3 Conditional Random Field Model

We will use the block graph $G_{b}$ described in Section 6.4.2 to encode the conditional independence assumptions of our Ising CRF model. Following (6.1) the $L_{2}$-regularized negative $\log$-likelihood $L_{\mathrm{ML}}(\boldsymbol{\theta})$ of the data under our model parameterized by $\boldsymbol{\theta}$ can be written as

$$
\begin{align*}
& L_{\mathrm{ML}}(\boldsymbol{\theta}):=\frac{1}{2} \lambda\|\boldsymbol{\theta}\|^{2}+\ln Z(\boldsymbol{\theta})+\sum_{(i, j) \in \mathcal{E}_{b}} \llbracket y_{i}^{b} \neq y_{j}^{b} \rrbracket E_{i j}^{b},  \tag{6.13}\\
& \text { where } \quad Z(\boldsymbol{\theta}):=\sum_{y} \exp \left(\sum_{(i, j) \in \mathcal{E}_{b}} \llbracket y_{i}^{b} \neq y_{j}^{b} \rrbracket E_{i j}^{b}\right) \tag{6.14}
\end{align*}
$$

is the partition function, and $\boldsymbol{y}^{b}$ the binary labels specifying territory (black or white) of the nodes (blocks) in $G_{b}$. Note that the original go positions can also have "neutral" territory, which does not belong to any player. We deal with such cases separately in Section 6.4.4. $E_{i j}^{b}:=\boldsymbol{\theta}^{\top} \boldsymbol{\phi}_{i j}^{b}$ is the disagreement cost of edge $(i, j) \in \mathcal{E}_{b}$. We now consider how to compute the sufficient statistics $\boldsymbol{\phi}_{i j}^{b}$ for every edge $(i, j)$ of our block graph.

(a) Nodes

(b) Edges

Figure 6.5: Computation of node and edge features of $G_{b}$. (a) Stone block, resulting in node feature vector $x_{\circ}=$ [ $2,4,2,1,9]$. For example, the $4^{\text {th }}$ element is 1 since there is only one (4), i.e., an intersection with 4 neighbours. (b) Two neighbouring stone blocks, resulting in edge feature vectors $x_{\bullet}=[3,3,1,0,7]$ and $x_{0} \bullet[6,3,0,0,9]$.

## Node Features

The fate of a block is largely determined by its shape and the arrangement of its nearby blocks. Therefore we believe that it is crucial to accurately capture the shape of a block. Neighbour classification of Vilà and Cazenave [134] provides a powerful way to summarise a block's shape, which is particularly important for empty blocks. Vilà and Cazenave [134] showed that in go, empty blocks with identical neighbour classification have identical eye-forming potential.

For our node features we use an extension of neighbour classification. We let $\boldsymbol{x}_{i}$ be the vector of raw node features for block $i \in \mathcal{V}_{b}$. For each intersection in a block (stone block or empty block) we compute the number of adjacent points that are in the same block (Figure 6.5(a)). A block's shape feature is a vector of length five whose $k^{\text {th }}$ element counts the number of points with exactly $k$ neighbours

Table 6．3：Correspondence between node（left）resp．edge（center）types and parameters．Right：computation of disagreement cost $E_{i j}$ for two sample edge types． $\boldsymbol{\theta}_{t(i)}$ denotes the parameter vector for a node of type $t(i)$ ，while $\boldsymbol{\theta}_{t(i, j)}$ denotes the parameter vector for an edge of type $t(i, j)$ ．

| Node <br> Params． | Node <br> Types |
| :---: | :---: |
| $\boldsymbol{\theta}_{\diamond}$ | $\diamond$ |
| $\boldsymbol{\theta}_{\bigcirc}$ | $\bullet^{\circ}$ \＆$\bigcirc$ |
| $\boldsymbol{\theta}_{\square}$ | $\boldsymbol{⿴} \& \square$ |


| Edge Params． | Edge <br> Types |
| :---: | :---: |
| $\boldsymbol{\theta}_{\bigcirc \bigcirc}$ | $\bullet \rightarrow$ O \＆O $\rightarrow$－ |
| $\boldsymbol{\theta}_{\square \square}$ | $\square \rightarrow \square \& \square \rightarrow \square$ |
| $\boldsymbol{\theta}_{\text {○口 }}$ | $\bullet \rightarrow \square \& \bigcirc \rightarrow \square$ |
| $\theta_{\square}$ | $\square \rightarrow$ \＆$\rightarrow$－ |
| $\boldsymbol{\theta}_{\bigcirc \bigcirc}$ | $\bullet \rightarrow \diamond$－$\rightarrow \diamond$ |
| $\boldsymbol{\theta}_{\diamond \bigcirc}$ | $\diamond \rightarrow \bullet \& \diamond \bigcirc$ |
| $\boldsymbol{\theta}_{\square}$ | $\square \rightarrow \diamond$ ¢ $\rightarrow \diamond$ |
| $\boldsymbol{\theta}_{\diamond \square}$ | $\diamond \rightarrow$ ■ $\& \diamond \rightarrow \square$ |


| Edge（ $i, j$ ）with type ©■ |  |  |  |
| :---: | :---: | :---: | :---: |
| $\boldsymbol{x}_{i}$ | $\boldsymbol{x}_{i \rightarrow j}$ | $\boldsymbol{x}_{j \rightarrow i}$ | $\boldsymbol{x}_{j}$ |
| $\boldsymbol{\theta}_{\circ}$ | $\boldsymbol{\theta}_{\text {Oロ }}$ | $\boldsymbol{\theta}_{\square}$ | $\boldsymbol{\theta}_{\square}$ |
| $\boldsymbol{\theta}_{\bigcirc}^{\top} \boldsymbol{x}_{i}+\boldsymbol{\theta}_{\circ \square}^{\top} \boldsymbol{x}_{i \rightarrow j}+\boldsymbol{\theta}_{\square}^{\top} \boldsymbol{x}_{j \rightarrow i}+\boldsymbol{\theta}_{\square}^{\top} \boldsymbol{x}_{j}$ |  |  |  |
| Edge（ $i, j$ ）with type ©0 |  |  |  |
| $\boldsymbol{x}_{i}$ | $\boldsymbol{x}_{i \rightarrow j}$ | $\boldsymbol{x}_{j \rightarrow i}$ | $\boldsymbol{x}_{j}$ |
| $\boldsymbol{\theta}_{\circ}$ | $\boldsymbol{\theta}_{\bigcirc \bigcirc}$ | $\boldsymbol{\theta}_{\bigcirc}$ | $\boldsymbol{\theta}^{\circ}$ |
| $\boldsymbol{\theta}_{\bigcirc}^{\top} \boldsymbol{x}_{i}+\boldsymbol{\theta}_{\circ \square}^{\top} \boldsymbol{x}_{i \rightarrow j}+\boldsymbol{\theta}_{\square \bigcirc}^{\top} \boldsymbol{x}_{j \rightarrow i}+\boldsymbol{\theta}_{\square}^{\top} \boldsymbol{x}_{j}$ |  |  |  |

in the same block．For completeness，the $5^{\text {th }}$ element of $\boldsymbol{x}_{i}$ stores the the total number of intersections in the block．

## Edge Features

Nearby blocks can interact with each other in a number of ways，for example：
－Two stone blocks can merge into one by connecting with each other．
－A stone block can become alive by forming two eyes in the nearby empty space．
－A stone block can surround an opponent block by reducing its liberties．
The edges of the block graph must capture and describe such interactions．We believe that the shape of the boundary between two neighbouring blocks contains important information about their interaction．

For this purpose we define our edge features using a variant of neighbour classification［134］．Let $\boldsymbol{x}_{i \rightarrow j}$ be the vector of raw edge features for the edge $(i, j) \in \mathcal{E}_{b}$ as viewed from block $i$ ．For each point in block $i$ we compute the number of adjacent points that are in block $j$ ．We now let $\boldsymbol{x}_{i \rightarrow j}$ be a vector of length five whose $k^{\text {th }}$ element counts the number of points in block $i$ with exactly $k$ neighbours in block $j$ ．For completeness，the $5^{\text {th }}$ element of $\boldsymbol{x}_{i \rightarrow j}$ stores the length of the boundary between $i$ and $j$ ．Vector $\boldsymbol{x}_{j \rightarrow i}$ is computed similarly，but with the roles of $i$ and $j$ reversed（Figure 6．5（b））．Note that the elements of $\boldsymbol{x}_{i \rightarrow j}$ and $\boldsymbol{x}_{j \rightarrow i}$ can be different．It is also worth noting that our edge features provide additional information that is not conveyed by the node features alone．This contrasts with the common approach of creating edge features that are simply some arithmetic combination（e．g．，dot－product，magnitude of difference）of node features．

## Feature Processing

Since（6．13）provides for edge disagreement costs only，we move our node features into the features of each adjoining edge．Because our edge features are non－commutative，we provide shape vectors for both orientations．This means that the full feature vector for an edge $(i, j)$ comprises of four parts，two node feature vectors and two edge feature vectors： $\boldsymbol{x}_{i j}:=\left[\boldsymbol{x}_{i}, \boldsymbol{x}_{i \rightarrow j}, \boldsymbol{x}_{j \rightarrow i}, \boldsymbol{x}_{j}\right]$（Table 6．3，right）．We

|  | Current Edge |  | Neighbour Edges |  |
| :---: | :---: | :---: | :---: | :---: |
|  | Parameters | Features | Parameters | Features |
| Nodes | $\boldsymbol{\theta}_{\circ}$ | $\boldsymbol{x}_{i}$ | $\tilde{\boldsymbol{\theta}}_{\diamond}$ | $\boldsymbol{x}_{k}$ |
|  | $\theta_{\square}$ | $\boldsymbol{x}_{j}$ | $\tilde{\boldsymbol{\theta}}_{\bigcirc}$ | $\boldsymbol{x}_{m}$ |
|  |  |  | $\tilde{\boldsymbol{\theta}}_{\square}$ | $\boldsymbol{x}_{n}$ |
| Edges | $\theta_{\text {○口 }}$ | $\boldsymbol{x}_{i \rightarrow j}$ | $\tilde{\boldsymbol{\theta}}_{\diamond \bigcirc}$ | $\boldsymbol{x}_{k \rightarrow i}$ |
|  | $\boldsymbol{\theta}_{\square}$ | $\boldsymbol{x}_{j \Rightarrow i}$ | $\tilde{\boldsymbol{\theta}}_{\bigcirc \odot}$ | $\boldsymbol{x}_{i \rightarrow k}$ |
|  |  |  | $\tilde{\boldsymbol{\theta}}_{\bigcirc}$ | $\begin{aligned} & \hline \boldsymbol{x}_{m \rightarrow i} \\ & \boldsymbol{x}_{i \rightarrow m} \end{aligned}$ |
|  |  |  | $\tilde{\boldsymbol{\theta}}_{\square \square}$ | $\begin{aligned} & \boldsymbol{x}_{n \rightarrow j} \\ & \boldsymbol{x}_{j \rightarrow n} \\ & \hline \end{aligned}$ |



Table 6.4: Parameters and features (left) used to compute the disagreement cost of edge ( $i, j$ ) in a small block graph (right). Neighbouring edges are shown in green.
perform normalisation and power expansion, replacing each feature value $x \in \mathbb{N}$ with $1 /(x+1), x / 20$, and $x^{2} / 400$, and provide a constant bias feature set to 1 . Such an expansion allows the Ising CRF to capture non-linear interactions between the features.

## Parameter Sharing

For each type of node and edge, we provide a single parameter vector that is shared between all nodes resp. edges of that kind. There are three types of node parameters, one for each node type: stone $\left(\boldsymbol{\theta}_{\circ}\right)$, surround $\left(\boldsymbol{\theta}_{\square}\right)$, and neutral $\left(\boldsymbol{\theta}_{\diamond}\right)$ (Table 6.3, left). There are eight types of edge parameters resulting from all possible types of node pairings (Table 6.3, center). We let $\boldsymbol{\theta}_{t(i)}$ and $\boldsymbol{\theta}_{t(i, j)}$ denote the parameter vectors for a node of type $t(i)$ and edge of type $t(i, j)$, respectively. We aggregate these 11 parameter vectors into a single vector $\boldsymbol{\theta}$, and match input features with the appropriate parameters via a routing matrix $R$ that depends on the kind of edge present: $\boldsymbol{\phi}_{i j}:=\boldsymbol{R}_{i j} \boldsymbol{x}_{i j}$.

## Neighbour Features

To assist the propagation of go knowledge along the graph, for each edge we also include the features, weighted by $\tilde{\boldsymbol{\theta}}$, of its neighbouring nodes and edges. Table 6.4 shows the resulting features and parameters for one particular edge (double blue line). With this new model, the final disagreement cost $E_{i j}$ for edge ( $i, j$ ) becomes

$$
\begin{align*}
E_{i j} & =\boldsymbol{\theta}_{t(i)}^{\top} \boldsymbol{x}_{i}+\boldsymbol{\theta}_{t(i, j)}^{\top} \boldsymbol{x}_{i \rightarrow j}+\boldsymbol{\theta}_{t(j, i)}^{\top} \boldsymbol{x}_{j \rightarrow i}+\boldsymbol{\theta}_{t(j)}^{\top} \boldsymbol{x}_{j} \\
& +\sum_{k \sim i ; k \neq j}\left(\tilde{\boldsymbol{\theta}}_{t(k)}^{\top} \boldsymbol{x}_{k}+\tilde{\boldsymbol{\theta}}_{t(k, i)}^{\top} \boldsymbol{x}_{k \rightarrow i}+\tilde{\boldsymbol{\theta}}_{t(i, k)}^{\top} \boldsymbol{x}_{i \rightarrow k}\right)  \tag{6.15}\\
& +\sum_{k \sim j ; k \neq i}\left(\tilde{\boldsymbol{\theta}}_{t(k)}^{\top} \boldsymbol{x}_{k}+\tilde{\boldsymbol{\theta}}_{t(k, j)}^{\top} \boldsymbol{x}_{k \rightarrow j}+\tilde{\boldsymbol{\theta}}_{t(j, k)}^{\top} \boldsymbol{x}_{j \rightarrow k}\right)
\end{align*}
$$

where $\sim$ denotes the adjacency relation between nodes, i.e., $k \sim i$ is a node $k$ that is adjacent to node $i$.

## Parameter Estimation

We train our Ising CRF model on the block graph $G_{b}$ by penalised maximum likelihood (ML) parameter estimation, which involves minimising the negative log-likelihood (6.13). Since this is a convex optimisation problem, efficient quasi-Newton solvers such as LBFGS can be employed to good effect [86, 95]. These solvers require the gradient of the log-likelihood

$$
\begin{equation*}
\frac{\partial}{\partial \boldsymbol{\theta}} L_{\mathrm{ML}}(\boldsymbol{\theta})=\lambda \boldsymbol{\theta}+\sum_{(i, j) \in \mathcal{E}_{b}}\left(\llbracket y_{i}^{b} \neq y_{j}^{b} \rrbracket-\frac{\partial \ln Z(\boldsymbol{\theta})}{\partial E_{i j}^{b}}\right) \boldsymbol{\phi}_{i j}^{b}, \tag{6.16}
\end{equation*}
$$

where we compute $\partial \ln Z(\boldsymbol{\theta}) / \partial E_{i j}^{b}$ via the inverse of the Kasteleyn matrix (5.23).

## Prediction

Recall that we train on the block graph $G_{b}$, but predict on the group graph $G_{g}$. Parameter estimation on the block graph provides us with $\boldsymbol{\theta}^{*}:=\operatorname{argmin}_{\boldsymbol{\theta}} L_{M L}(\boldsymbol{\theta})$. We use $\boldsymbol{\theta}^{*}$ to compute the disagreement costs for the block graph $E_{i j}^{b}:=\boldsymbol{\theta}^{* \top} \boldsymbol{\phi}_{i j}^{b}$. These allow us to compute the disagreement costs of the group graph:

$$
\begin{equation*}
E_{g(i), g(j)}^{g}:=\sum_{(i, j) \in \mathcal{E}_{b}} E_{i j}^{b} \tag{6.17}
\end{equation*}
$$

where $(g(i), g(j)) \in \mathcal{E}_{g}$ and $g(i)$ is the group of block $i \in \mathcal{V}_{b}$. See Figure 6.6 for an example. We can


Figure 6.6: Left: Block graph $G_{b}$ with edge disagreement costs $a, b, c$. Right: Corresponding group graph $G_{g}$.
now use blossom-shrinking to efficiently compute the MAP state of the group graph:

$$
\begin{equation*}
\boldsymbol{y}^{g^{*}}:=\underset{\boldsymbol{y}^{g}}{\operatorname{argmax}} \mathbb{P}\left(\boldsymbol{y}^{g} \mid \boldsymbol{x} ; \boldsymbol{\theta}^{*}\right) . \tag{6.18}
\end{equation*}
$$

We set the state of each block $\boldsymbol{y}_{i}^{b}$ to the state of its parent group $\boldsymbol{y}_{g(i)}^{g^{*}}$. Similarly, we set the state of each intersection $\boldsymbol{y}_{i}$ to the state of its parent block. Since $G_{b}$ is generally not outerplanar (Section 5.1.2), our parameter estimation model does not use a constant bias node, as was done in Section 5.1. For the same reason, our prediction model does not use a constant bias node, and thus due to label symmetry there are two possible MAP states. We resolve this ambiguity by a domain-specific heuristic: there are usually (e.g., in $99.4 \%$ of our $9 \times 9$ games) more alive than dead stones. Of the two MAP states $\boldsymbol{y}^{g *}$ we therefore pick the one which maximises the number of alive stones, resolving ties randomly.

### 6.4.4 Experiments

We now evaluate our Ising CRF model and inference algorithms in terms of their territory prediction accuracy and speed, on several sets of final go positions at different board sizes.

## Datasets

$\mathbf{9} \times \mathbf{9}$ games. van der Werf et al. [130, 131, 132] have created a collection of about $180009 \times 9$ games that were played between 1995 and 2002 on the No Name Go Server (NNGS), whose territory has been determined using a combination of manual labeling, GNU Go 3.6 [39], and their own classifiers. We use a subset of this collection of games for our experiments: 1000 games played between 1996 and 2002 for training and validation, and 906 games played in 1995 for testing. The average rank of ranked players is 17 kyu in the training set and 12 kyu in the test set.
$19 \times 19$ games. We had great difficulty in finding a sufficiently large collection of $19 \times 19$ games whose labels have been independently verified. Thus we developed our own heuristic scoring method to label a set of games from the Kiseido Go Server (KGS), and picked only games labeled identically by our scorer and the players themselves for the dataset. This heuristic scorer is completely independent from our Ising CRF classifier and thus does not cause any bias. The training and test set consist of 1000 games each, played between January 2005 and January 2003, respectively. The average rank of ranked players is 11 kyu for both training and test sets.
Oversize games. We manually scored 22 games that were played on KGS in December 2004 with extra-large board sizes ranging from $21 \times 21$ to $38 \times 38$. Compared to the $9 \times 9$ and $19 \times 19$ datasets, these games had a higher average number of dead blocks and stones, making them a greater challenge to score. The average rank of ranked players in these games was 15 kyu.

## Territory Prediction Accuracy

Task. Given an endgame board position $G$ we want to predict the label (black, neutral or white) of each intersection in $\mathcal{V}$. However, our classifier can only make binary predictions. This is not a problem though: by predicting black and white we obtain the location of all dead stones; once those are removed identifying neutral intersections becomes simple. We use four measures of test error:

- Stone error is the percentage of misclassified stones in $\mathcal{V}$.
- Block error is the percentage of misclassified stone blocks in $\mathcal{V}_{b}$.
- Game error is the percentage of games that have at least one misclassified stone block in $\mathcal{V}_{b}$.
- Winner error is the percentage of games whose winner (using Chinese scoring) is predicted incorrectly.

Method. We trained our Ising CRF model on 1000 block graphs until convergence, using LBFGS as the optimizer. The regularizer $\lambda$ was tuned for best performance, which was found at $\lambda=3 / 4$. Predicted territory was determined from the MAP state of the group graphs on the test set. We compared our approach to four control methods:

- The naive method of assuming that all stones are alive, i.e., always labelling $\bullet$ as black and $\bigcirc$ as white.
- GNU Go 3.6, an open-source go program that scores endgame positions by combining go-specific knowledge with results from many local searches.
- A neural network (NN) that uses a total of 63 go -specific features of graphs at several levels of abstraction. This is the best classifier from the collection built by van der Werf et al. [131] to predict the final territory of $9 \times 9$ endgame positions.
- A simple MRF (just 6 parameters) trained by Stern et al. [119] on the grid graphs of $29019 \times 19$ expert games. Inference is performed via 50 iterations of loopy belief propagation (LBP) to compute approximate marginal expectations at each intersection, which are then thresholded to yield a prediction. In contrast to our approach of using the MAP state, marginals can lead to inconsistent labelings (for example see Figure 6.7, left). Since this method has no notion of blocks and would therefore do poorly on the block error, we give it a helping hand in our comparison by setting each block's label to the label of the majority of its stones.

Results. Table 6.5 compares our approach with the controls on various board sizes. Our method (Ising_9 and Ising_19) significantly outperforms MRF, the only other learning method based on graphical models, on all board sizes. On $9 \times 9$ it appears inferior to approaches that heavily rely on domain knowledge (GnuGo and NN). However, it must be noted that this dataset was labelled using a combination of GnuGo and NN classifiers in the first place [131], hence is quite likely biased towards these methods.

It is interesting to note that MRF is relatively better at stone error than block error. This is due to the fact that MRF has no concept of blocks and treats all stones as individuals. Thus, unlike our method, an error in one stone does not cause errors in other stones of the same block.

On $19 \times 19$ our method is clearly worse than GnuGo. To put this in perspective though, consider that GnuGo takes about four orders of magnitude as long to make a prediction here (Section 6.4.4). Our model trained on $9 \times 9$ positions (Ising_9) does not perform much worse here than the natively trained Ising_19, indicating that we are able to transfer knowledge across board sizes. Figure 6.7, left shows a sample position, with the errors made by each method marked.

To investigate their scalability, we also tested the graphical models on our small set of oversize games. Even though players of differing skill levels on boards of differing sizes tend to reach different types of final positions, our architecture manages a fair amount of information transfer; Figure 6.7, right gives an example where our model made no error.

## Territory Prediction Speed

Task. In this set of experiments, we compare the CPU times taken by various territory prediction methods as a function of $\left|\mathcal{V}_{g}\right|$. All experiments were performed on an Intel Pentium 4 running at 3 GHz with 1 GB RAM and 512 KB level 2 cache. We compare the following five methods:

- GNU Go 3.6 scoring the position in aftermath mode.
- The approach of Stern et al. [119]: 50 iterations of loopy belief propagation (LBP) to calculate approximate marginal expectations on the grid graph.

| Size | Algorithm | Error (\%) |  |  |  |
| :---: | :---: | ---: | ---: | ---: | ---: |
|  |  | Stone | Block | Game | Winner |
| $9 \times 9$ | Naive | 8.80 | 17.57 | 75.70 | 30.79 |
|  | MRF | 5.97 | 8.19 | 38.41 | 13.80 |
|  | Ising_9 | 2.32 | 2.57 | 9.05 | 2.32 |
|  | GnuGo* $^{*}$ | $\mathbf{0 . 0 5}$ | - | 1.32 | - |
|  | NN* | 0.19 | $\leq \mathbf{1 . 0 0}$ | $\mathbf{1 . 1 0}$ | $\mathbf{0 . 5 0}$ |
| $19 \times 19$ | Naive | 6.96 | 16.52 | 98.30 | 32.60 |
|  | MRF | 3.80 | 4.91 | 63.90 | 20.50 |
|  | Ising_9 | 3.91 | 5.22 | 61.90 | 14.30 |
|  | Ising_19 | 3.81 | 3.93 | 43.40 | 9.30 |
|  | GnuGo | $\mathbf{0 . 1 1}$ | - | $\mathbf{5 . 1 0}$ | - |
| greater <br> than <br> $19 \times 19$ | Naive | 10.25 | 19.64 | 100.00 | 31.81 |
|  | MRF | 6.83 | 7.80 | 100.00 | 22.73 |
|  | Ising_9 | 6.64 | 6.95 | 95.45 | 13.64 |
|  | Ising_19 | $\mathbf{5 . 0 2}$ | $\mathbf{4 . 5 2}$ | $\mathbf{8 1 . 8 2}$ | $\mathbf{9 . 0 9}$ |

*Bias alert: dataset creation involved these methods (see text for details).
Table 6.5: Prediction error of different methods for various board sizes. Naive is the naive error, MRF is the work of Stern et al. [119], GnuGo is GNU Go 3.6 [39], NN is the work of [131], Ising_9 and Ising_19 is our method trained on $9 \times 9$ and $19 \times 19$ games, respectively. We could not easily obtain the block and winner errors for GnuGo. The minimum in each result column is boldfaced.


Figure 6.7: Left: a $19 \times 19$ endgame position showing the errors made by different methods. Stones marked with $O$ are misclassified by Naive, MRF and Ising CRF; $\square$ by Naive and MRF; $\Delta$ by Naive only. GnuGo made no errors in this game. Notice how MRF misclassifies stones at R12 and S12 and yet manages to get the rest of the block labelled correctly. Right: A $25 \times 25$ endgame position, whose territory assignment is predicted correctly by our system trained on $19 \times 19$ boards. By comparison, MRF on the same position predicts 33 stones (in 4 blocks) incorrectly.

- "Brute force": MAP state calculation using variable elimination with arbitrary elimination ordering, on the group graph.
- MAP state calculation using variable elimination with elimination ordering determined by the min-fill heuristic [64], on the group graph.
- Our approach: MAP state calculation by running Blossom IV [17] on the expanded dual of the group graph.

Results. Figure 6.8 shows that our approach (blossom) is about three orders of magnitude faster than popular inference methods for graphical models, both exact (variable elimination) and approximate (LBP). The knowledge-intensive GnuGo is an order of magnitude slower still, though admittedly far more accurate. Only the primitive brute force method can compete in speed with the blossom-shrinking approach on small graphs, as encountered in $9 \times 9$ go. However, its exponential complexity means that it is useless for $19 \times 19$ go positions, let alone larger problems of any kind. Blossom-shrinking, by contrast, scales as $O\left(\left|\mathcal{E}_{g}\right|\left|\mathcal{V}_{g}\right|^{3}\right)$ in the worst case.


Figure 6.8: Average CPU time taken for territory prediction methods as a function of $\left|\mathcal{V}_{g}\right|$. Grey histogram indicates the number of graphs of a given size: the two broad peaks correspond to board sizes $9 \times 9$ and $19 \times 19$.

### 6.4.5 Outlook and Discussion

We presented an exact and efficient method for territory prediction in go. Rather than the naive grid graph, our method uses higher-order representations, namely the block and group graphs, to encode dependencies. Our inference algorithm is highly efficient and scalable, yielding predictions orders of magnitude faster than conventional approximate and exact graphical inference techniques. As a result, we were able to predict territory on $g o$ boards as large as $38 \times 38$.

Our Ising CRF model could be improved in terms of its prediction accuracy. We suspect that our generic shape-based features do not capture all the intricacies of life and death in go. Some domainspecific feature engineering could greatly boost performance in that regard.

On the other hand, we note that Monte-Carlo go programs, which currently dominate computer go, rely on fast rather than accurate static evaluation of their leaf nodes. While we have applied our model to territory prediction, it is generic enough to be adapted to a wide range of computer go subproblems.

The speed of inference in our model makes it suitable for leaf evaluation in game-tree searches and Monte-Carlo go.

### 6.5 Conclusion

In this chapter we have demonstrated the suitability of our planar Ising CRF on three different tasks. Although these are relatively simple tasks, they are sufficiently large to show that our approach is correct and efficient. For parameter estimation we have used both penalized maximum likelihood and maximum margin methods. We have shown that maximum margin is orders of magnitude faster than penalized maximum likelihood. Furthermore, maximum margin converges to parameters that are superior for prediction.

In terms of prediction, we have shown that the computation of MAP states via blossom-shrinking is orders of magnitude faster than the computation of $\mathrm{M}^{3} \mathrm{P}$ states. In particular, we were able to obtain the MAP state for a random $1500 \times 1500$ Ising grid in 200 CPU seconds. We have found reconstruction via MAP to be similar to that of $\mathrm{M}^{3} \mathrm{P}$, despite the latter's theoretical advantages [33].

## Chapter 7

## Non-Planar Ising Graphical Models

The algorithms described in Chapter 5 provide a foundation for approximate inference algorithms for non-planar Ising graphical models. Importantly the method for calculating the ground state (Section 5.2) also works exactly for non-planar graphs whose ground state does not contain frustrated noncontractible cycles. In this chapter we discuss approximative algorithms for Ising graphical models whose ground state does contain frustrated non-contractible cycles.

### 7.1 Edmond's Blossom-Shrinking Algorithm

In Chapter 5 we have used the blossom-shrinking algorithm [23, 24] for computing the maximumweight perfect matching $\mathcal{M}$ (Definition 14) of our expanded dual graph $G_{E}^{*}$. We then used $\mathcal{M}$ to obtain the exact ground state of a planar model graph $G$. To understand the process for non-planar graphs $G$, we begin by reviewing the blossom-shrinking algorithm, closely following the description of Kolmogorov [67].

Let $G(\mathcal{V}, \mathcal{E})$ be an undirected weighted graph. Each edge $e \in \mathcal{E}$ has a corresponding weight $c_{e}$. Our goal is to compute a perfect matching $\mathcal{M}^{*} \subseteq \mathcal{E}$ with the maximum weight $\sum_{e \in \mathcal{M}^{*}} c_{e}$. We can represent a perfect matching $\mathcal{M} \subseteq \mathcal{E}$ with an incidence vector $\boldsymbol{x} \in\{0,1\}^{|\mathcal{E}|}$, where $x_{e}=\llbracket e \in \mathcal{M} \rrbracket$. For a subset $\mathcal{S} \subseteq \mathcal{V}$, let $\delta(\mathcal{S})$ be the set of boundary edges of $\mathcal{S}: \delta(\mathcal{S}):=\{(i, j) \in \mathcal{E}: i \in \mathcal{S}, j \in \mathcal{V} \backslash \mathcal{S}\}$. Let $O$ be the set of all subsets of $\mathcal{V}$ that have an odd number of nodes greater than one. The blossom-shrinking algorithm efficiently solves the following integer linear program:

$$
\begin{align*}
\underset{x}{\operatorname{maximize}} & \sum_{e \in \mathcal{E}} c_{e} x_{e}  \tag{7.1}\\
\text { subject to } \quad \sum_{e \in \delta((i\})} x_{e}=1 & \forall i \in \mathcal{V},  \tag{7.2}\\
\sum_{e \in \delta(\mathcal{S})} x_{e} \geq 1 & \forall \mathcal{S} \in \mathcal{O},  \tag{7.3}\\
x_{e} \geq 0 & \forall e \in \mathcal{E} . \tag{7.4}
\end{align*}
$$

(7.2) ensures that there is at most one edge in $\mathcal{M}^{*}$ incident upon any node $i \in \mathcal{V}$. If we select any subset $\mathcal{S}$ with an odd number of nodes then at least one of those nodes must be matched to a node not in $\mathcal{S}$,
hence (7.3). Let $\boldsymbol{y} \in \mathbb{R}^{|\mathcal{V}|+|O|}$ be a feasible dual vector, then the dual program is given as:

$$
\begin{array}{r}
\underset{y}{\operatorname{minimize}} \quad \sum_{i \in \mathcal{V}} y_{i}+\sum_{\mathcal{S} \in O} y_{\mathcal{S}} \\
\text { subject to } \quad \operatorname{slack}(e) \geq 0 \quad \forall e \in \mathcal{E}, \\
y_{\mathcal{S}} \geq 0 \quad \forall \mathcal{S} \in O \tag{7.7}
\end{array}
$$

where $\operatorname{slack}(e)$ in Equation 7.6 denotes the reduced cost of edge $e=(i, j)$ :

$$
\begin{equation*}
\operatorname{slack}(e):=c_{e}-y_{i}-y_{j}-\sum_{\mathcal{S} \in O}: e \in \delta(\mathcal{S}), \tag{7.8}
\end{equation*}
$$

An edge $e$ is said to be tight if $\operatorname{slack}(e)=0$. A set $\mathcal{S} \in O$ is called full if $\sum_{e \in \delta(\mathcal{S})} x_{e}=1$. Blossomshrinking is a primal-dual algorithm that maintains both an infeasible primal solution $\boldsymbol{x}$ (i.e., a nonperfect matching) and a feasible dual solution $\boldsymbol{y}$. The algorithm alternates dual updates that create new tight edges with primal updates that increase the size of the proposed matching. Primal feasibility is achieved when the matching becomes perfect, i.e., when $\sum_{e \in \mathcal{E}} x_{e}=|\mathcal{V}| / 2$. At convergence, $\boldsymbol{x}$ corresponds to a perfect matching of maximal cost and the complementary slackness conditions are satisfied:

$$
\begin{align*}
\operatorname{slack}(e)>0 & \Rightarrow x_{e}=0 \quad \text { and }  \tag{7.9}\\
y_{\mathcal{S}}>0 & \Rightarrow \sum_{e \in \delta(\mathcal{S})} x_{e}=1 . \tag{7.10}
\end{align*}
$$

(7.9) states that edges in the matching must be tight. One potential concern is that the dual problem (7.5) has an exponential number of variables $y_{\mathcal{S}}$. The algorithm's efficiency is due to there being at most $O(|\mathcal{V}|)$ subsets $\mathcal{S} \in \mathcal{O}$ with a non-zero variable $y_{\mathcal{S}}$ at any given moment. These subsets are called blossoms:

Definition 42 A blossom of a graph $G(\mathcal{V}, \mathcal{E})$ is a subgraph $G_{b}\left(\mathcal{V}_{b}, \mathcal{E}_{b}\right)$ such that the removal of any single node $i \in \mathcal{V}_{b}$ (and its incident edges) leaves $G_{b}$ with at least one perfect matching.

A blossom can also be defined recursively: it is a cycle containing an odd number of "pseudonodes," where a pseudonode is either a node in $\mathcal{V}$ or another blossom (dashed ovals in Figure 7.1(a)). Two adjacent pseudonodes in a blossom are connected via a blossom-forming edge (red edges in Figure 7.1). The key idea is that if we can match an edge coming out of a node inside a blossom, then we can recursively match the remaining even number of nodes using only blossom-forming edges (Figures 7.1(b) and 7.1(c)). Since blossom-forming edges are tight, such a completion of the matching is guaranteed to be optimal.

### 7.1.1 Application to Non-Planar Graphs

For convenience we define an edge state of an Ising graphical model:
Definition 43 An edge state of an Ising graphical model defined on graph $G(\mathcal{V}, \mathcal{E})$ is a binary state $\boldsymbol{y}^{e} \in\{-1,1\}^{\mid \mathcal{E |}}$ of its edges; its energy is given by $E\left(\boldsymbol{y}^{e}\right):=\frac{1}{2} \sum_{(i, j) \in \mathcal{E}}\left(1-y_{i j}^{e}\right) E_{i j}$.


Figure 7.1: (a) A blossom containing another blossom as a pseudonode (dashed ovals). Blossom nodes and blossom-forming edges are red, while all other nodes and edges are black. (b) and (c): optimal completion of the matching (bold edges) when a node in a blossom is matched to an exterior node.

Edge states are an extension of node states: given any node state $\boldsymbol{y}$ with energy $E(\boldsymbol{y})$, we can construct an edge state $\boldsymbol{y}^{e}$ with identical energy $E\left(\boldsymbol{y}^{e}\right)=E(\boldsymbol{y})$ by assigning $y_{i j}^{e}:=2 \llbracket y_{i}=y_{j} \rrbracket-1, \forall(i, j) \in \mathcal{E}$. However, the converse is not necessarily true, since there are edge states with no corresponding node state. An edge state is globally consistent if it has a corresponding node state:

Definition 44 An edge state $\boldsymbol{y}^{e}$ of an Ising graphical model defined on graph $G(\mathcal{V}, \mathcal{E})$ is globally consistent iff $\prod_{(i, j) \in C} y_{i j}^{e}=1$ for all cycles $C \subseteq \mathcal{E}$. An edge state $\boldsymbol{y}^{e}$ of an Ising graphical model with an embedded graph $G(\mathcal{V}, \mathcal{E}, \Pi)$ is locally consistent iff $\prod_{(i, j) \in C_{\Pi}} y_{i j}^{e}=1$ for all those cycles $C_{\Pi} \subseteq \mathcal{E}$ that are contractible in $\Pi$.

In other words, if an edge state $\boldsymbol{y}^{e}$ is locally consistent then every contractible cycle in $G$ (e.g., green line in Figure 7.2) will have an even number of disagreement edges. However, the definition of local consistency does not guarantee any consistency for edges on non-contractible cycles ${ }^{11}$ of $G$ (e.g., blue and red lines in Figure 7.2): if we label the nodes by walking around the hole of the torus, the label of the final node may end up in contradiction to its starting label.

As described in Chapter 5, the maximum-weight perfect matching $\mathcal{M}$ of our expanded dual graph $G_{E}^{*}$ can be used to obtain the exact ground state of a planar model graph $G$. When $G$ is non-planar, however, blossom-shrinking only recovers the extended ground state of $G$ :

Definition 45 An extended ground state of an Ising graphical model with an embedded graph $G(\mathcal{V}, \mathcal{E}, \Pi)$ is a locally consistent edge state with minimum energy.

### 7.2 Heuristics

For non-planar graphs blossom-shrinking only recovers the extended ground state, which in general does not uniquely determine a ground state. Since the computation of the ground state of a non-planar

[^9]

Figure 7.2: A contractible (green) and two non-contractible (blue and red) cycles on the surface of a torus. (a) 3D view of the torus (courtesy of Wikipedia). (b) the torus represented as a regular grid with periodic boundary conditions: opposite boundary edges are identified with each other.

Ising graphical model is NP-complete [2], we must resort to approximative algorithms. In this section we propose heuristic methods for obtaining ground states from the extended ground state of a non-planar Ising graphical model.

### 7.2.1 Low-Genus Embedding

Non-planar graphs cannot be embedded in the plane, but every graph can be drawn on some orientable surface such that no edges cross and each face is topologically a 2-disk, i.e., has no holes [138].

To reduce the number of non-contractible cycles in $G$, we would like to obtain an embedding of as low a genus as possible. Unfortunately finding the lowest-genus embedding of a graph is NP-complete [128]. However, for regular grids we can use a simple method for constructing a (not necessarily optimal) low-genus embedding:

Consider an undirected binary graphical model with energy function (5.3) defined on a $6 \times 6$ grid, as shown in Figure 7.3(a). To construct an equivalent Ising graphical model we must connect every node to a bias node (see Section 5.1). We connect all the nodes on the perimeter to the bias node by routing the edges around the surface, as shown in Figure 7.3(b). We connect the remaining nodes through the holes in the surface, as shown in Figure 7.3(c). Each hole in Figure 7.3(c) can route up to 4 edges (from 4 nodes) without any crossings (see Figure 7.3(b)). With this method an $N \times M$ grid can be embedded on a surface with genus $\lceil N / 2-1\rceil\lceil M / 2-1\rceil$.

### 7.2.2 Minimax Spanning Tree

In our Ising graphical model the extended ground state is an edge state $\boldsymbol{y}^{e} \in\{-1,1\}^{|\mathcal{E}|}$, such that the state of edge $(i, j)$ is $y_{i j}^{e}:=1-2 \llbracket(i, j) \in \mathcal{E} \backslash \mathcal{M} \rrbracket$. Since the model graph is not planar, the complement $\mathcal{E} \backslash \mathcal{M}$ may not be a cut. To disambiguate the node state we use a spanning tree, as in Section 6.2.3, to obtain a minimax solution. We construct a maximum-weight spanning tree $T\left(\mathcal{V}, \mathcal{E}^{\prime}\right)$ of the model graph $G(\mathcal{V}, \mathcal{E})$ whose edge weights are set to $-E_{i j} y_{i j}^{e}$, which is the local cost of disobeying the edge state $\boldsymbol{y}^{e}$.

(a)

(b)

(c)

Figure 7.3: Construction of a low-genus embedding for a regular $6 \times 6$ grid (a). (b) $6 \times 6$ grid with every node (white circles) connected to the bias node (represented by multiple black disks). (c) The graph in (b) (omitted for clarity) lies on top of the embedding surface (represented as a cube). Arrows indicate the routing of connections to the bias node (yellow ball) through the 4 holes.

As before, we run Algorithm 1 on $T$ (bold edges in Figure 7.5) to find the ground state $\boldsymbol{y}$ of the model graph $G$.

We can also take advantage of the dual slack information provided by Blossom to construct a minimax spanning tree. For each edge $(i, j)$ of the expanded dual graph $G_{E}^{*}$ the cost of disobeying Blossom's matching is the dual $\operatorname{slack}(i, j)(7.8)$. Consider the matching of $G_{E}^{*}$ shown in Figure 7.4(a). If we remove edge $(i, j)$ from the matching $\mathcal{M}$, then we must add two of its neighbouring edges to $\mathcal{M}$ - one from each side (one possible matching is shown in Figure 7.4(b)). Although we do not know which of the neighbouring edges of $(i, j)$ will be added to $\mathcal{M}$, we can compute a lower bound on their cost. Hence we can set the weight of edge $(i, j) \in \mathcal{E}^{\prime}$ to

$$
\begin{equation*}
\operatorname{slack}(i, j)+\min _{k \in N(i) \backslash\{j\}} \operatorname{slack}(i, k)+\min _{k \in N(j) \backslash i\}} \operatorname{slack}(j, k), \tag{7.11}
\end{equation*}
$$

where $N(i) \subseteq \mathcal{V}$ is the set of neighbours of node $i$. Since $\operatorname{slack}(i, j)$ is always non-negative, (7.11) gives a lower bound on the cost of changing the state of $(i, j)$, i.e., it is a min-marginal for $(i, j)$ [65]. This is in contrast to using edge weights of the model graph $G$, which can be either positive or negative and hence does not provide any bound.

### 7.2.3 Node Quenching with Implicit Lookahead

We propose a method for improving an approximate ground state via small, local moves which we call node quenching. This method is based on Iterated Conditional Modes (ICM), which we describe below.

## Iterated Conditional Modes

ICM [8] is a method that finds an approximate MAP state of a MRF. The algorithm begins with an initial node state $\hat{\boldsymbol{y}}$. During each iteration, for each node ICM selects the state that produces the largest decrease in energy. This process is repeated until convergence, which is guaranteed to occur, and is usually rapid in practice. The order of node updates can be arbitrary; Besag [8] proposes a raster scan


Figure 7.4: (a) Blossom's matching $\mathcal{M}$ (bold edges) of the expanded dual graph $G_{E}^{*}$ (red). (b) a possible matching of $G_{E}^{*}$ when the edge $(i, j)$ is removed from $\mathcal{M}$.
whose direction is varied from one iteration to the next, thus avoiding possible directional effects. ICM is sensitive to the initial estimate $\hat{\boldsymbol{y}}$, especially in high-dimensional spaces where there are many local minima. Besag [8] initialises $\hat{\boldsymbol{y}}$ to the state which is best locally:

$$
\begin{equation*}
\hat{\boldsymbol{y}}_{i}=\underset{y_{i} \in \mathcal{Y}}{\operatorname{argmax}} \mathbb{P}\left(Y_{i}=y_{i} \mid \boldsymbol{x}, \boldsymbol{\theta}\right) . \tag{7.12}
\end{equation*}
$$

We modify ICM as follows:

1. We initialize $\hat{\boldsymbol{y}}$ to the state given by the minimax spanning tree, which we know to be a good starting point;
2. We prioritise the order in which nodes are considered;
3. We include an implicit one-step lookahead, which allows us to make certain uphill moves.

## Prioritisation

For each node $i \in \mathcal{V}$ we compute the change in energy $f_{i}$ incurred by fipping that node's state, i.e., changing it to the opposite state (we assume binary node states):

$$
\begin{equation*}
f_{i}:=\sum_{(i, j) \in \mathcal{E}} E_{i j} y_{i j}^{e} \tag{7.13}
\end{equation*}
$$

We control the order of the node flips by using a priority queue $q=\left(f_{i_{1}}, \ldots, f_{i_{n}}\right)$, where $f_{i_{1}}$ is at the head of the queue and node $i_{1}$ will be considered next. This queue prioritises nodes whose flipping produces the greatest decrease in energy, i.e., $f_{i_{k}} \leq f_{i_{j}} \forall k<j$. The use of a priority queue ensures that promising states are explored first, so that the method is less likely to get stuck in local minima. Once node $i$ is flipped the global energy changes by $f_{i}$ and now flipping node $i$ will incur a cost of $-f_{i}$. We also need to update the change in energy of its neighbouring nodes, i.e., $f_{k} \forall(i, k) \in \mathcal{E}$; the cost of flipping all other nodes is unaffected.

## Implicit One-Step Lookahead

Now suppose we make an uphill move $f_{i}>0$ that leads to a larger downhill move $f_{k}<-f_{i}$, where node $k$ is the best node to flip after flipping node $i$. However, if an uphill move $f_{i}>0$ does not immediately lead to a larger downhill move, then $-f_{i}$ will remain at the head of the queue and we will have a neverending 2-cycle. We detect such cycles when we see node $i$ twice in a row, and break them by setting $f_{i}:=\infty$ and $y_{i}:=-y_{i}$. This method is equivalent to a one-step lookahead, but without the cost of checking every pair of nodes. We terminate the method when $f_{i_{1}}=\infty$, i.e., when there is no node or pair of nodes left whose flipping decreases the energy.

### 7.2.4 Tree Quenching

Node quenching makes small changes to the node state, as it only considers flipping individual or pairs of nodes. We now introduce tree quenching, which is able to make large, global changes to the node state. During each iteration, tree quenching flips the state of an edge of the minimax spanning tree $T\left(\mathcal{V}, \mathcal{E}^{\prime}\right)$ (Section 7.2.2) that produces the greatest decrease in energy. We now show how to compute the potential change in energy $f_{e_{k}}$ if the state of edge $e_{k}=(i, j) \in \mathcal{E}^{\prime}$ is flipped. Split $T$ into edge $(i, j)$ and the two disjoint subtrees it connects: $T_{1}\left(\mathcal{V}_{1}, \mathcal{E}_{1}^{\prime}\right)$ and $T_{2}\left(\mathcal{V}_{2}, \mathcal{E}_{2}^{\prime}\right)$, as shown in Figure 7.5. To summarize:

$$
\begin{array}{r}
i \in \mathcal{V}_{1}, \quad j \in \mathcal{V}_{2}, \\
\mathcal{V}_{1} \cup \mathcal{V}_{2}=\mathcal{V}, \quad \mathcal{V}_{1} \cap \mathcal{V}_{2}=\emptyset \\
\mathcal{E}_{1}^{\prime} \cup\{(i, j)\} \cup \mathcal{E}_{2}^{\prime}=\mathcal{E}^{\prime}, \quad \mathcal{E}_{1}^{\prime} \cap \mathcal{E}_{2}^{\prime}=\emptyset \tag{7.14}
\end{array}
$$

We want to flip the state of $e_{k}$ while keeping the state of all other edges in $\mathcal{E}^{\prime}$ unchanged. This corresponds to flipping the state of all the nodes of one of the subtrees. For efficiency, we flip the nodes of the smaller subtree. The edges of $G$ whose state will change are those that connect the two subtrees, i.e., the edges in the set $\mathcal{E}_{f}=\left\{(i, j) \in \mathcal{E}: i \in \mathcal{V}_{1}, j \in \mathcal{V}_{2}\right\}$, which is a cut of $G$ (dashed line in Figure 7.5). The change in energy incurred by flipping the state of edge $e_{k}$ is

$$
\begin{equation*}
f_{e_{k}}:=\sum_{(i, j) \in \mathcal{E}_{f}} E_{i j} y_{i j}^{e} \tag{7.15}
\end{equation*}
$$

Once we have flipped the state of edges in $\mathcal{E}_{f}$ we must update $f_{e_{k}} \forall e_{k} \in \mathcal{E}^{\prime}$. This can be done efficiently by noting that flipping the state of a non-tree edge $(i, j) \in \mathcal{E} \backslash \mathcal{E}^{\prime}$ only affects $f_{e_{k}}$ for edges $e_{k}$ that lie on the path $\Pi(i, j) \subseteq \mathcal{E}^{\prime}$, which can be precomputed.

As in node quenching we use a priority queue $q=\left(f_{e_{1}}, \ldots, f_{e_{n}}\right)$, where $f_{e_{1}}$ is at the head of the queue and edge $e_{1}$ will be considered next. The tree quenching algorithm is shown in Algorithm 5. After every iteration of the main loop Algorithm 5 needs to recompute the cost of many edges in $T$, making it considerably slower than node quenching. Hence we propose two versions of tree quenching, which differ only in their termination criterion:

1. Fast: terminate when there is no edge whose flipping decreases the energy, i.e., when $f_{e_{1}}>0$.
2. Slow: terminate when there are no edge or pair of edges whose flipping decreases the energy, i.e., when $f_{e_{1}}=\infty$.

Note that slow tree quenching can be up to a magnitude slower than fast tree quenching (see right column in Figure 7.9).


Figure 7.5: $5 \times 5$ model grid graph $G$ with edges of the minimax spanning tree $T$ shown in bold. The two subtrees $T_{1}$ and $T_{2}$ and the edge to be flipped are shown in blue, red and black respectively. The cut of $G$ (dashed line) crosses the edges whose state will change if the state of the bold black edge is flipped.

```
Algorithm 5 Tree Quenching
    Input: model graph \(G(\mathcal{V}, \mathcal{E})\), minimax spanning tree \(T\left(\mathcal{V}, \mathcal{E}^{\prime}\right)\)
        1. \(\forall(i, j) \in \mathcal{E} \backslash \mathcal{E}^{\prime}\) :
            Compute and store the path \(\Pi(i, j) \subseteq \mathcal{E}^{\prime}\)
    \(\forall k \in \Pi(i, j): \quad\) (initialize edge priorities in \(q\) )
            \(f_{e_{k}}:=f_{e_{k}}+E_{i j} y_{i j}^{e}\)
2. Main Loop:
\((i, j):=q \cdot \mathrm{pop}()\)
Divide \(T\) into \(T_{1}\) and \(T_{2}\) as shown in (7.14)
Compute the cut \(\mathcal{E}_{f}=\left\{(i, j) \in \mathcal{E}: i \in \mathcal{V}_{1}, j \in \mathcal{V}_{2}\right\}\)
\(\forall(p, q) \in \mathcal{E}_{f}\) :
\(\forall k \in \Pi(p, q)\) : (update edge priorities in \(q\) )
\(f_{e_{k}}:=f_{e_{k}}-2 E_{p q} y_{p q}^{e}\)
\(y_{p q}^{e}:=-y_{p q}^{e} \quad\) (flip the state of edges along the cut)
```


### 7.3 Experiments

In Section 7.2 we proposed a number of heuristics for obtaining ground states from the extended ground state of a non-planar Ising graphical model. We described a method for obtaining a low-genus embedding (Section 7.2.1) and discussed methods for constucting a consistent ground state using a minimax spanning tree (Section 7.2.2). We also suggested algorithms for finding a lower-energy state from a given ground state (Sections 7.2.3 and 7.2.4). To measure the effect of each heuristic, we designed inference algorithms with increasing levels of prediction accuracy:

- IS0: Compute the extended ground state using Blossom V. Approximate the ground state using the minimax spanning tree with disagreement edge costs (Section 7.2.2).
- IS: Compute the extended ground state using Blossom V. Approximate the ground state using the minimax spanning tree with blossom's slacks (Section 7.2.2).
- IS0-N: Run IS0 and approximate the ground state using node quenching (Section 7.2.3).
- IS-N: Run IS and approximate the ground state using node quenching (Section 7.2.3).
- IS-FN: Run IS and approximate the ground state using fast tree quenching followed by node quenching (Sections 7.2.3 and 7.2.4).
- IS-SN: Run IS and approximate the ground state using slow tree quenching followed by node quenching (Sections 7.2.3 and 7.2.4).

We compared our inference algorithms to a variety of control algorithms, that are considered state-of-the-art in MAP approximation:

- Iterated Conditional Modes (ICM) (see Section 7.2 .3 and [8]). Implementation from [121].
- Primal-dual algorithm FastPD [71, 72].
- Sequential Tree Reweighting (TRW-S) (see Section 4.3.6 and [66]). Implementation from [121].
- Sequential max-product loopy belief propagation (BP-S) [66]. Implementation from [121].
- Max-product loopy belief propagation (BP-M) [122]. Implementation from [121].

In our experiments we found that FastPD does not converge to low energies. We believe that in our case of random potentials, the LP relaxation used by FastPD is a poor approximation to the true energy. We also found that ICM converges to poor solutions. Therefore, for the sake of clarity we omit the results of FastPD and ICM.

### 7.3.1 Experimental Setup

We tested the algorithms on undirected binary graphical models with the energy function (5.3) defined on a regular grid. As shown in Theorem 33, these models correspond to Ising graphical models defined on the same grid, with every node connected to an additional bias node $y_{0}:=0$. We use the embedding method described in Section 7.2.1. As identified in the literature [51, 66, 105, 121], the performance of the control algorithms depends on the following factors:

1. Number of cycles in the model graph.
2. Percentage of frustrated cycles (Definition 25) in the model graph.
3. Strength of unary versus pairwise potentials.

Computing the number of cycles in a graph is a difficult problem. For grids, however, the size of the grid is a good measure for the number of cycles (Figure 7.6(a)). We can also use the size of the grid to determine the genus of the embedding, which our algorithms depend upon. Similarly it is difficult to compute the percentage of frustrated cycles in a graph. In grids the percentage of frustrated cycles is dependent on the percentage of non-submodular edges (Figure 7.6(b)). Note that when the percentage of non-submodular edges is $100 \%$ there are no frustrated cycles, since all cycles in the regular grid have an even length. With the knowledge of these dependencies, we vary three parameters in our experiments: the size of the grid $N$, the percentage of non-submodular edges $\alpha$, and the strength of the unary potentials $E_{i}^{\prime}$ relative to that of the pairwise potentials $E_{i j}^{\prime}$. The latter is controlled by the variable $\sigma$ and can be interpreted as a type of signal-to-noise ratio (SNR). For each parameter set, we generate 100 grids, following the construction of Kolmogorov and Wainwright [69]:

- Sample node potentials from the normal distribution: $E_{i}^{\prime}(0) \sim \mathcal{N}(0,1)$ and $E_{i}^{\prime}(1) \sim \mathcal{N}(0,1)$.
- Set edge potential to $E_{i j}^{\prime}(0,0)=E_{i j}^{\prime}(1,1):=0$ and $E_{i j}^{\prime}(0,1)=E_{i j}^{\prime}(1,0):=\lambda_{i j}$, where the random variable $\lambda_{i j} \sim-\left|\mathcal{N}\left(0, \sigma^{2}\right)\right|$ with probability $\alpha$ and $\lambda_{i j} \sim\left|\mathcal{N}\left(0, \sigma^{2}\right)\right|$ with probability $(1-\alpha)$.

All experiments were carried out on a Linux PC with 3 GB RAM and dual Intel Pentium 4 processors running at 3.6 GHz , each with 2 MB of level 2 cache.


Figure 7.6: (a) Number of cycles in a regular grid as a function of its size (based on A140517 in [118]). (b) Percentage of frustrated cycles in a regular grid as a function of the percentage of non-submodular edges.

### 7.3.2 Comparison with the Ground State

We first compared different algorithms relative to the true ground state. This enabled us to measure the node error and the energy achieved by each algorithm relative to the minimal energy. Since the computation of ground states in our large 3-dimensional parameter space is rather time consuming, we
decided to explore only certain slices of our parameter space. In each such slice we varied one of the three parameters, while keeping the other two parameters fixed:

- Fix $\sigma=1.0, \alpha=0.5$ and vary the grid size $N \in\{4,8,16,24,32,48,64\}$, which corresponds to the embedding genus of $\{1,9,49,121,225,529,961\}$.
- Fix $\sigma=1.0, N=32$ and vary $\alpha \in\{0.01,0.1\} \times\{0,1,2,3,4,5,6,7,8,9,10\}$.
- Fix $\alpha=0.5, N=32$ and vary $\sigma \in\{0.5,0.75,1.0,1.25,1.5,1.75,2.0,2.25,2.5\}$.

The relative energy was computed by comparing to the energy of the ground state, i.e., the minimal energy. Let $E^{t}$ be the energy of the true ground state in trial $t$ and $E_{i}^{t}$ be the energy achieved by algorithm $i$ in trial $t$. The relative energy for algorithm $i$ is computed as

$$
\begin{equation*}
E_{i}=\frac{1}{T} \sum_{t=1}^{T} \frac{E^{t}}{E_{i}^{t}}, \tag{7.16}
\end{equation*}
$$

where $T$ is the total number of trials. In the above we assume that $E^{t}$ and $E_{i}^{t}$ have the same sign, which is the case in our experiments.

## Energy vs Distance to the Ground State

We also looked at the Hamming distance of the approximate ground states obtained by the various methods to the true ground state, normalized by the total number of nodes. Interestingly, we found that while our heuristics tended to find good local minima farther from the true grounds state, the control methods preferred inferior local minima closer to the true ground state (see Figure 7.7). This may reflect different biases in these approximate optimizers, for instance in their disposition towards making small $v s$. large moves in state space. The effect is pronounced here because we are optimizing over random energy landscapes, which are very non-smooth.

In a machine learning application the edge potentials would reflect some underlying ground truth for instance, they might be derived from training data by maximum-margin parameter estimation (Section 6.1.2). Unless the model is poorly chosen, good solutions (in terms of energy) would then necessarily lie close (in state space) to the ground truth, and points close to the ground truth would necessarily be good solutions. Under those circumstances distance to the optimum may be a useful stand-in for the criterion that is actually being optimized, the energy. Here, however, there is no ground truth, and in a random energy landscape these two quantities are effectively decoupled.

## Minimax Spanning Tree

To determine whether it is best to use blossom's slacks for the minimax spanning tree (Section 7.2.2) we compared IS0, IS, IS0-N and IS-N in terms of the achieved energy, as shown in Figure 7.8. In terms of the CPU running time all four algorithms were very similar, so we do not report it. Without any quenching there is no significant difference between IS and ISO. However, once node quenching is used IS-N overtakes IS0-N, showing that the minimax spanning tree is more powerful with blossom's slacks. Given this, we choose to use blossom's slack variables for all further experiments. We also notice that


Figure 7.7: A scatter plot of distance vs. energy, relative to the true ground state, for the approximate ground states obtained by a range of methods and parameter settings.

IS-N is significantly better than IS. This shows that node quenching is quite beneficial. Hence, we also omit IS from all further experiments.

## Quenching: Order and Iterations

We have two types of quenching (node and tree) at our disposal, so how many times and in what order should we use them? To answer the latter, we begin by noticing that it does not make sense to run two quenchings of the same type in a row. This is because the second quenching will not have any available moves. Thus the only remaining option is to alternate node quenching with tree quenching. We verified that it is never beneficial to perform node quenching before tree quenching. This makes sense, because one should always make large moves, as those of tree quenching, before applying small local optimisations.

The next question we wanted to answer is whether running more quenching iterations is necessary. Tree quenching followed by node quenching, followed by another tree quenching was only marginally better than IS-SN in terms of the achieved energy, while being significantly slower. Similarly, we do not expect that further quenching iterations can significantly decrease the energy, since most of the work is already done during previous quenching iterations.

## Tree Quenching

In our next set of experiments we analyzed the effects of tree quenching and compared its two running modes. To answer the former, we measured the achieved reduction in energy when tree quenching was used on top of node quenching (Figure 7.9). For all parameter settings we found that the reduction in energy was significant. In particular, the amount of reduction increases with $\sigma$ and the size of the
$\qquad$
(a)

(c)


Figure 7.8: Comparison of IS0, IS, ISO-N and IS-N. $y$-axis shows the energy as a function of (a) genus of the embedding surface, (b) percentage of non-submodular edges and (c) sigma.
grid (Figures 7.9(a) and 7.9(e)). Given the significant improvement in energy, the extra running time incurred by tree quenching is a small price to pay. Hence we can confidently recommend the use of tree quenching.

We also wanted to determine whether slow tree quenching is worth the additional running time (Figure 7.9). As expected, IS-SN is better than IS-FN in terms of the achieved energy in all regimes. Furthermore, the margin of improvement is nearly constant per experiment. The biggest improvement is obtained when we vary $\alpha$, in particular when $\alpha<0.2$. The ratio between the running times of the two methods remains constant for a fixed grid size (Figures 7.9(d) and 7.9(f)). As we vary the grid size the ratio varies between 2 and 6 for small and large grids, respectively (Figures 7.9(a) and 7.9(b)). Given these results, one should prefer IS-SN over IS-FN for smaller grids (e.g., $N \leq 32$ ), where the running time is not an issue. For larger grids, the lower energy achieved by IS-SN is overshadowed by its slower running time, and hence in such situations one might consider using IS-FN.

## Comparison to Control Methods

After removing all our inferior algorithms we were left with just two Ising algorithms: IS-FN, ISSN . We decided to compare these to the control algorithms (TRW-S, BP-S and BP-M) in terms of the achieved energy and CPU running time, as shown in Figure 7.9.

As discussed in Section 7.2.1, the performance of our algorithms should decrease as the genus increases. However, our experiments did not confirm this, since IS-FN and IS-SN outperformed the control algorithms for all genera (Figure 7.9(a)). One explanation is that all algorithms were affected by the increased genus, therefore their rank remained relatively unchanged. In the range $0.1<\alpha<0.9$ our algorithms were better than the control algorithms in terms of the achieved energy (Figure 7.9(c)). This shows that the control algorithms are best suited for submodular or nearly-submodular problems, and confirms the results of previous studies [121]. We also found that the control algorithms become significantly worse for larger $\sigma$ (Figure 7.9(e)), while the performance of our algorithms is unaffected.

The running time of all algorithms increased with the genus, although it increased at a higher rate for our algorithms. Nevertheless, our algorithms were faster than the control algorithms for genus 121 and lower. Unlike our algorithms, the running time of the control algorithms was also affected by the percentage of non-submodularity (Figure 7.9(d)). We wanted to see what would happen if the control algorithms were given the same CPU time as our methods. We increased the running time of the control algorithms by a factor of 10 by changing their default configuration (see MRF-benchmarks code [121]) from 500 outer and 1 inner iterations to 1000 outer and 5 inner iterations, respectively. However this did not significantly improve their performance in terms of the achieved energy, and often made it worse.

### 7.3.3 Overall Comparison

For our final set of experiments we wanted to answer the following important question: under what set of conditions do our algorithms perform better resp. worse than the control algorithms? To answer this question we compared our best algorithm (IS-SN) to the control algorithms TRW-S, BP-S and BP-M in a large parameter space. This was feasible, because we avoided the cost of computing the ground states. We explored all possible sets of parameters:
(a)

(c)

(e)

(b)

(d)

(f)


| - -*- - TRW-S |
| :---: |
| - -m- - BP-S |
| -A- -BP-M |
| - IS-N |
| 1 -IS-FN |
| +-IS-SN |

Figure 7.9: Comparison of IS-N, IS-FN and IS-SN to the three control algorithms. $y$-axis is a function of the genus of the embedding surface in row 1 , percentage of non-submodular edges in row 2 and sigma in row 3 . Left column: $y$-axis is the achieved energy. Right column: $y$-axis is the CPU running time in seconds.


Figure 7.10: Comparison of IS-SN to the three control algorithms. For every combination of parameters we compute at $95 \%$ confidence level whether IS-SN is significantly better (dark red), significantly worse (dark blue) or equal (dark green) to the best control algorithm. We use these known data points to extrapolate the result of the untested parameters settings, i.e., the remaining area (shown with corresponding lighter colors). (a) $\sigma \in$ $\{0.5,0.625,0.75,0.875,1.0\}$. (b) $\sigma \in\{1.0,1.5,2.0,2.5\}$.

| k | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| $q_{k}$ | 1.960 | 2.241 | 2.394 | 2.498 | 2.576 | 2.638 | 2.690 | 2.724 | 2.773 |

Table 7.1: Critical values $q_{k}$ for the two-tailed Bonferroni-Dunn test at the $95 \%$ confidence level. $k$ is the number of classifiers and includes the control classifier [20].

- Grid size $N \in\{4,8,16,24,32,48,64,100\}$.
- $\sigma \in\{0.5,0.625,0.75,0.875,1.0,1.5,2.0,2.5\}$.
- Non-submodularity $\alpha \in\{0.0,0.02,0.04,0.06,0.08,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0\}$.

It was not feasible to plot the results of each experiment, since that would require a 4-dimensional graph ${ }^{12}$. Instead, we decided to compare all the inference algorithms based on their average ranking, and plot only the best ranked algorithms on a 3-dimensional graph. The rank of algorithm $i$ for trial $t$ is computed as

$$
\begin{equation*}
R_{i}^{t}:=1+\left|\left\{k: E_{k}^{t}<E_{i}^{t}\right\}\right|+\frac{1}{2}\left|\left\{k: E_{k}^{t}=E_{i}^{t}, k \neq i\right\}\right|, \tag{7.17}
\end{equation*}
$$

where $E_{i}^{t}$ is the energy achieved by algorithm $i$ in trial $t$. The average rank of algorithm $i$ is then $R_{i}:=\sum_{t} E_{i}^{t} / T$, where $T$ is the total number of trials. Since we were comparing multiple algorithms, rather than just two, we chose to use the Bonferroni-Dunn test over any variant of the paired t-test [20,22]. In particular, two algorithms $i$ and $j$ were considered significantly different at a $95 \%$ confidence confidence level if

$$
\begin{equation*}
\left|R_{i}-R_{j}\right| \geq q_{k} \sqrt{\frac{k(k+1)}{6 T}} \tag{7.18}
\end{equation*}
$$

where $k$ is the number of algorithms and $q_{k}$ is the critical value given in Table 7.1.
We plotted the regions where IS-SN was significantly better, worse or equal (shown in red, blue and green respectively) to the best of the control algorithms for $\sigma \in\{0.5,0.625,0.75,0.875,1.0\}$ in Figure 7.10(a) and for $\sigma \in\{1.0,1.5,2.0,2.5\}$ in Figure 7.10(b). In these figures we set the value of the untested region to the value of the closest (in Euclidean space) tested region.

In general, the region where IS-SN is better becomes larger as $\sigma$ increases. When the node potentials dominate (i.e., at $\sigma=0.5$ ), our algorithm does not outperform the control algorithms. The situation changes rapidly, however, as soon as the strength of the edge potentials increases relative to the node potentials. Already at $\sigma=0.75$, IS-SN is best in more than half of the available space, while for $\sigma>1.5$ it is best nearly everywhere. Interestingly, at $\sigma=2.5 \mathrm{IS}-\mathrm{SN}$ outperformed the control methods even on grids as small as $4 \times 4$. We expect these trends to continue for larger values of $\sigma$.

As we have observed previously (see Section 7.3.2), the relative performance of IS-SN does not get worse with the increase in the grid's size. The most likely explanation is that the size of the grid affects all methods equally, meaning that their ranking relative to each other remains constant. This demonstrates the potential of our approach to scale well to larger problems.

The only parameter setting where IS-SN was significantly worse than the control methods occurred when the model had a low percentage of frustrated cycles, i.e., when $\alpha \leq 0.1$ and $\alpha \geq 0.9$. This should

[^10]come at no surprise, since the control methods are specifically designed to work for such submodular models. Notice that for all other parameter settings, IS-SN was equal or better than the control algorithms.

In conclusion, we have found a large region in the parameter space where one should prefer ISSN over any other state-of-the-art algorithm. In particular, one should use our IS-SN for any binary 4-connected MRF whose edge potentials are either

- strongly non-submodular: $0.2 \leq \alpha \leq 0.8$; or
- slightly non-submodular and stronger than the node potentials: $0.1 \leq \alpha \leq 0.9$ and $\sigma \geq 1.5$.


### 7.4 Conclusion

In this chapter we have proposed a number of heuristics that can be used for approximating the ground state from the extended ground state of an Ising graphical model. These included: obtaining a low-genus embedding, computing a minimax spanning tree with consistent edge states and methods for minimizing the energy of a node state (quenching). We have shown experimentally that all these heuristics benefit our framework. Some of these heuristics are specific to our framework, while others, such as quenching can be easily combined with any other inference framework. For example, one can compute an approximate ground state with any given inference algorithm and then optimise the result with a combination of node and tree quenching. This will be the focus of our future work.

We also compared our inference algorithms to state-of-the-art algorithms on random grid MRFs. Our algorithms were able to achieve lower-energy states on a range of parameter settings, without being significantly slower. We believe that there exist a range of non-trivial problems that are suitable for our algorithms. One such problem is viral marketing in social networks [141]. In viral marketing the task is to market a product to the "best" set of individuals in order to trigger the most widespread adoption of the product. This task has been successfully modelled as a MRF, whose nodes represent customers and edges correspond to relationships between customers [21, 103]. The network value of a customer is the expected profit from sales to that customer. Importantly, this value not only depends on the customer's probability of buying the product, but also on his/her ability to persuade his friends, family and colleagues in buying the product. Hence if a customer is highly connected or is a well-know opinion leader (e.g., celebrity) then his/her network value is dominated by his/her ability to persuade others. In terms of the underlying MRF, this corresponds to large edge potentials, i.e., $\sigma \geq 1.5$ in our model. Therefore, we believe that our inference algorithms would be ideally suitable for such a task, especially if the underlying graph was large, e.g., Facebook, MySpace or LinkedIn.

A number of non-submodular computer vision applications have been proposed recently: texture restoration, image stitching, diagram recognition, super resolution, new view synthesis and image deconvolution $[68,105]$. In all of these applications, however, either the node potentials are too strong or there is not enough non-submodularity, making them unsuitable for our algorithms. We believe that current computer vision problems are designed to suit current available inference algorithms and viceversa. This selection bias impedes progress in both applications and in the development of inference algorithms for non-submodular problems. The presence of our framework, that does not rely on submodularity, should help to dissolve this bias and open doors for new applications in computer vision and other fields.

## Chapter 8

## Conclusion

This thesis introduced the Ising graphical model. This is an undirected graphical model that obeys the constraints of the Ising model, i.e., a binary-labeled energy function that is a sum of edge disagreement costs. Despite its simplicity, we proved that the Ising graphical model with an additional bias node is equivalent to any binary-labeled pairwise MRF (Theorem 33).

Our work can be described in two parts. In the first part (Chapters 5 and 6) we discussed exact inference algorithms for the planar Ising model, while in the second we looked at approximative algorithms for ground states of non-planar Ising models.

In contrast to previous approaches, our construction uses the complementary mapping (Section 3.2.2) between even subgraphs in $G_{0}$ and perfect matchings in the decorated graph $G_{d}$, where $G_{0}$ is the dual graph $G^{*}$ with unmodified edge weights. This gives us a complementary connection between graph cuts of the model graph $G$ and perfect matchings in $G_{d}$. We explain the role of planarity (resp. triangulation) in making this relation a surjection (resp. bijection). Our method for decorating the graph is similar to Kasteleyn's (Definition 24), but does not need dummy nodes since we plane triangulate $G$. Our construction is conceptually simpler than previous constructions, since we do not explicitly construct the dual graph and perform all operations in the model graph $G$.

We show how our framework can be used for the exact computation of the partition function, worst margin violators, marginal edge probabilities and lowest-energy (ground) states in the planar Ising graphical model. This allows us to implement popular parameter estimation frameworks of maximummargin and maximum-likelihood. By reusing the proto-Kasteleyn matrix $\boldsymbol{H}$ (Section 5.3.3) we speed up the parameter estimation loop by orders of magnitude. We demonstrated that our inference algorithms are efficient and effective on a number of real-world machine learning problems; in particular, we found that maximum margin is orders of magnitude faster than penalized maximum likelihood, as well as converging to superior parameters.

In the second part, we proposed a number of heuristic methods for approximating the ground state of a non-planar Ising graphical model. These novel contributions included: obtaining a low-genus embedding, computing a minimax spanning tree with consistent edge states and methods for minimizing the energy of a node state (quenching). We conducted a thorough experimental comparison to show that our framework benefits from every heuristic. When combined with our best heuristics, our inference algorithm outperforms current state-of-the-art algorithms by achieving lower-energy states on a range of parameter settings. Finally, we characterise the set of MRF models for which our inference algorithms are preferable over other algorithms.

### 8.1 Future Work

There are a number of extensions for this work. These can be divided into two categories: new applications and algorithmic developments. Some possible new applications are the following:

- The Ising graphical model can also implement $\alpha$-expansion moves and $\alpha-\beta$-swaps [12]. This will allow us to deal with multi-labeled models and give rise to a number of new applications.
- Our exact inference algorithms replace the submodularity constraint of graph cut methods with a planarity constraint. The existence of two distinct tractable frameworks for inference in binary graphical models implies a powerful hybrid. Such a hybrid could be used when the model graph has biconnected components that are either planar or submodular. It would be interesting to see whether such a hybrid approach can be used for less obvious situations.
- As we discussed in Chapter 7, our algorithms for non-planar Ising graphical models are wellsuited for the task of viral marketing [141]. This is because such tasks can be modeled as a binary pairwise MRF with large edge potentials, i.e., $\sigma \geq 1.5$ in our model (Section 7.3.1). These tasks require efficient inference algorithms for large graphs (social networks).

Some possible algorithmic developments are the following:

- There may still be other constructions for computing the partition function and ground states of an Ising model. For example, to our knowledge nobody has tried to use the complementary mapping (Section 3.2.2) where $G_{0}$ is the model graph $G$.
- Our quenching methods (Sections 7.2.3 and 7.2.4) can be used to improve the node state computed by any inference algorithm. This opens up many possibilities for new algorithms, which may turn out to be better than current algorithms on particular types of models.
- It is possible to use our framework to efficiently compute the exact min-marginals [4], instead of approximate min-marginals that we are currently computing (Section 7.2.2). These can be used to construct a better minimax spanning tree, which may lead to stronger performance of our algorithms.


## Bibliography

[1] Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst. Templates for the solution of Algebraic Eigenvalue Problems: A Practical Guide. Society for Industrial and Applied Mathematics, Philadelphia, USA, 2000.
[2] F. Barahona. On the computational complexity of Ising spin glass models. Journal of Physics A: Mathematical, Nuclear and General, 15(10):3241-3253, 1982.
[3] F. Barahona, R. Maynard, R. Rammal, and J. P. Uhry. Morphology of ground state of twodimensional frustration model. Journal of Physics A: Mathematical and General, 15:673-699, 1982.
[4] Dhruv Batra and Tsuhan Chen. Dynamic planar-cuts: Efficient computation of min-marginals for outer-planar models. NIPS Workshop on Discrete Optimization in Machine Learning, 2009. http://www.ece.cmu.edu/~dbatra/publications/assets/batra_discml_nips09.pdf.
[5] Peter Benner, Ralph Byers, Heike Fassbender, Volker Mehrmann, and David Watkins. Choleskylike factorizations of skew-symmetric matrices. Electronic Transactions on Numerical Analysis, 11:85-93, 2000.
[6] A. Bertoni, P. Campadelli, G. Gangai, and R. Posenato. Approximability of the ground state problem for Ising spin glasses. Journal of Complexity, 13:326-339, 1997.
[7] Julian Besag. Spatial interaction and the statistical analysis of lattice systems (with discussion). Journal of the Royal Statistical Society. Series B, 36(2):192-236, 1974.
[8] Julian Besag. On the statistical analysis of dirty pictures. Journal of the Royal Statistical Society B, 48(3):259-302, 1986.
[9] I. Bieche, R. Maynard, R. Rammal, and J. P. Uhry. On the ground states of the frustration model of a spin glass by a matching method of graph theory. Journal of Physics A: Mathematical and General, 13:2553-2576, 1980.
[10] S. Boettcher and A. G. Percus. Optimization with extremal dynamics. Physical Review Letters, 86:5211-5214, 2001.
[11] John M. Boyer and Wendy J. Myrvold. On the cutting edge: Simplified $O(n)$ planarity by edge addition. Journal of Graph Algorithms and Applications, 8(3):241-273, 2004. Reference implementation (C source code): http://jgaa.info/accepted/2004/BoyerMyrvold2004.8.3/ planarity.zip.
[12] Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization via graph cuts. IEEE Transactions on Pattern Analysis and Machine Intelligence, 23(11):1222-1239, 2001.
[13] Stephen G. Brush. History of the Lenz-Ising model. Reviews of modern physics, 39:883-893, 1967.
[14] James R. Bunch. A note on the stable decomposition of skew-symmetric matrices. Mathematics of Computation, 38(158):475-479, April 1982.
[15] Arthur Cayley. Sur les déterminants gauches. Journal für die reine und angewandte Mathematik, 38:93-96, 1849.
[16] Chandra Chekuri, Sanjeev Khanna, Joseph (Seffi) Naor, and Leonid Zosin. Approximation algorithms for the metric labeling problem via a new linear programming formulation. In SODA '01: Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms, pages 109-118, Philadelphia, PA, USA, 2001. Society for Industrial and Applied Mathematics.
[17] William Cook and André Rohe. Computing minimum-weight perfect matchings. INFORMS Journal on Computing, 11(2):138-148, 1999. C source code available at http://www.isye. gatech.edu/~wcook/blossom4/.
[18] Marcel Crasmaru and John Tromp. Ladders are PSPACE-complete. In Computers and Games, pages 241-249, 2000.
[19] James W. Demmel, Stanley C. Eisenstat, John R. Gilbert, Xiaoye S. Li, and Joseph W. H. Liu. A supernodal approach to sparse partial pivoting. SIAM Journal on Matrix Analysis and Applications, 20(3):720-755, 1999. C source code available from http://crd.lbl.gov/~xiaoye/ SuperLU/.
[20] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. Journal of Machine Learning Research, 7:1-30, 2006.
[21] Pedro Domingos and Matthew Richardson. Mining the network value of customers. In Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 57-66, New York, NY, USA, 2001. ACM.
[22] Olive Jean Dunn. Multiple comparisons among means. Journal of the American Statistical Association, 56(293):52-64, March 1961.
[23] Jack Edmonds. Maximum matching and a polyhedron with 0,1-vertices. Journal of Research of the National Bureau of Standards, 69B:125-130, 1965.
[24] Jack Edmonds. Paths, trees, and flowers. Canadian Journal of Mathematics, 17:449-467, 1965.
[25] S. F. Edwards and P. W. Anderson. Theory of spin glasses. Journal of physics F: metal physics, 5:965-974, 1975.
[26] Markus Enzenberger. Evaluation in Go by a neural network using soft segmentation. In Advances in Computer Games, pages 97-108, 2003.
[27] Markus Enzenberger. The integration of a priori knowledge into a Go playing neural network. Technical report, University of Munich, September 1996.
[28] Jeff Erickson and Kim Whittlesey. Greedy optimal homotopy and homology generators. In Proc. 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pages 1038-1046, Vancouver, BC, 2005. SIAM. ISBN 0-89871-585-7.
[29] Richard P. Feynman. Statistical Mechanics. A Set Of Lectures. Addison-Wesley, Reading, MA, 1972.
[30] Michael E. Fisher. Statistical mechanics of dimers on a plane lattice. Physical Review, 124(6): 1664-1672, 1961.
[31] Michael E. Fisher. On the dimer solution of planar Ising models. Journal of Mathematical Physics, 7(10):1776-1781, 1966.
[32] L. R. Ford and D. R. Fulkerson. Flows in Networks. Princeton University Press, Princeton, NJ, 1962.
[33] C. Fox and G. K. Nicholls. Exact map states and expectations from perfect sampling: Greig, Porteous and Seheult revisited. In A. Mohammad-Djafari, editor, Bayesian Inference and Maximum Entropy Methods in Science and Engineering, volume 568 of American Institute of Physics Conference Series, pages 252-263, 2001.
[34] Brendan J. Frey and David J. C. MacKay. A revolution: Belief propagation in graphs with cycles. In M. I. Jordan, M. J. Kearns, and S. A. Solla, editors, Advances in Neural Information Processing Systems 10, volume 10, pages 479-485. MIT Press, 1998.
[35] K. J. Friedenbach, Jr. Abstraction hierarchies: a model of perception and cognition in the game of Go. PhD thesis, University of California, Santa Cruz, 1980.
[36] Anna Galluccio and Martin Loebl. On the theory of Pfaffian orientations. i. perfect matchings and permanents. Journal of Combinatorics, 6:81-100, 1999.
[37] Anna Galluccio, Martin Loebl, and Jan Vondrák. New algorithm for the Ising problem: Partition function for finite lattice graphs. Physical Review Letters, 84(26):5924-5927, Jun 2000.
[38] Amir Globerson and Tommi Jaakkola. Approximate inference using planar graph decomposition. In B. Schölkopf, J. Platt, and T. Hofmann, editors, Advances in Neural Information Processing Systems 19, pages 473-480, Cambridge, MA, 2007. MIT Press.
[39] GNU Go 3.6. http://www. gnu.org/software/gnugo, 2004.
[40] Thore Graepel, Mike Goutrié, Marco Krüger, and Ralf Herbrich. Learning on graphs in the game of Go. In International Conference on Artificial Neural Networks, pages 347-352, London, UK, 2001. Springer-Verlag. ISBN 3-540-42486-5.
[41] D. M. Greig, B. T. Porteous, and A. H. Seheult. Exact maximum a posteriori estimation for binary images. Journal of the Royal Statistical Society B, 51(2):271-279, 1989.
[42] Carsten Gutwenger and Petra Mutzel. Graph embedding with minimum depth and maximum external face. In G. Liotta, editor, Graph Drawing 2003, volume 2912 of Lecture Notes in Computer Science, pages 259-272. Springer Verlag, 2004.
[43] P. L. Hammer, P. Hansen, and B. Simeone. Roof duality, complementation and persistency in quadratic 0-1 optimization. Mathematical Programming, 28(2):121-155, February 1984.
[44] J. M. Hammersley and P. E. Clifford. Markov fields on finite graphs and lattices. unpublished manuscript, 1971.
[45] F. Harary. Feynman's simplification of the Kac-Ward treatment of the two-dimensional Ising model, 1958. In the draft chapter of Harary's book on graph theory.
[46] Nicholas J. Higham. Accuracy and Stability of Numerical Algorithms. SIAM, 2002.
[47] John E. Hopcroft and Robert E. Tarjan. Algorithm 447: Efficient algorithms for graph manipulation. Communications of the ACM, 16(6):372-378, 1973.
[48] J. Houdayer and O. C. Martin. Renormalization for discrete optimization. Physical Review Letters, 83(5):1030-1033, Aug 1999.
[49] Koji Hukushima and Koji Nemoto. Exchange Monte Carlo method and application to spin glass simulations. Journal of the Physical Society of Japan, 65, 1996.
[50] Charles A. Hurst and Herbert S. Green. New solution of the Ising problem for a rectangular lattice. Journal of chemical physics, 33:1059-1062, 1960.
[51] Alexander T. Ihler, John W. Fischer III, and Alan S. Willsky. Loopy belief propagation: Convergence and effects of message errors. J. Mach. Learn. Res., 6:905-936, 2005.
[52] Ernst Ising. Beitrag zur Theorie des Ferro- und Paramagnetismus. PhD thesis, Hamburg University, 1924.
[53] Ernst Ising. Beitrag zur Theorie des Ferromagnetismus. Zeitschrift für Physik, 31:253-258, February 1925.
[54] Sorin Istrail. Statistical mechanics, three-dimensionality and NP-completeness: I. universality of intractability of the partition functions of the Ising model across non-planar lattices. In Proceedings of the 32nd ACM Symposium on the Theory of Computing, pages 87-96, Portland, Oregon, 2000. ACM Press.
[55] Mark Jerrum and Alistair Sinclair. Polynomial-time approximation algorithms for the Ising model. SIAM Journal on Computing, 22:1087-1116, 1993.
[56] Mark Kac and John C. Ward. A combinatorial solution of the two dimensional Ising model. Physical Review, 88:1332-1337, 1952.
[57] Dmitry Kamenetsky, Nicol N. Schraudolph, Simon Günter, and S. V. N. Vishwanathan. Modelling Go positions with planar CRFs. NIPS workshop on Statistical Models of Networks, 2007. http://www.cs.ubc.ca/~murphyk/nips07NetworkWorkshop/abstracts/kamenetsky.pdf.
[58] Dmitry Kamenetsky, Nicol N. Schraudolph, Simon Günter, and S. V. N. Vishwanathan. Modelling Go positions with planar CRFs. NIPS workshop on Machine Learning and Games, 2007. http://www.cs.ubc.ca/~murphyk/nips07NetworkWorkshop/abstracts/kamenetsky.pdf.
[59] Pieter W. Kasteleyn. The statistics of dimers on a lattice: I. the number of dimer arrangements on a quadratic lattice. Physica, 27(12):1209-1225, 1961.
[60] Pieter W. Kasteleyn. Dimer statistics and phase transitions. Journal of Mathematical Physics, 4 (2):287-293, 1963.
[61] Pieter W. Kasteleyn. Graph theory and crystal physics. In Frank Harary, editor, Graph Theory and Theoretical Physics, chapter 2, pages 43-110. Academic Press, London and New York, 1967.
[62] KGS. The Kiseido Go Server, 2000. http://www.gokgs.com/.
[63] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi. Optimization by simulated annealing. Science, 220:671-680, 1983.
[64] Uffe Kjærulff. Triangulation of graphs - algorithms giving small total state space. Technical Report R-90-09, Department of Mathematics and Computer Science, Aalborg University, 1990. citeseer.ist.psu.edu/111946.html.
[65] Pushmeet Kohli and Philip H. S. Torr. Measuring uncertainty in graph cut solutions. Computer Vision and Image Understanding, 112(1):30-38, 2008.
[66] Vladimir Kolmogorov. Convergent tree-reweighted message passing for energy minimization. IEEE Transactions on Pattern Analysis and Machine Intelligence, 28(10):1568-1583, 2006.
[67] Vladimir Kolmogorov. Blossom V: A new implementation of a minimum cost perfect matching algorithm. Mathematical Programming Computation, 1(1):43-67, 2009. http://www.cs.ucl. ac.uk/staff/V.Kolmogorov/papers/BLOSSOM5.html.
[68] Vladimir Kolmogorov and Carsten Rother. Minimizing nonsubmodular functions with graph cuts - a review. IEEE Transactions on Pattern Analysis and Machine Intelligence, 29(7):12741279, July 2007.
[69] Vladimir Kolmogorov and Martin J. Wainwright. On the optimality of tree-reweighted maxproduct message-passing. In Proc. 21st Annual Conf. Uncertainty in Artificial Intelligence (UAI), pages 316-323. AUAI Press, 2005.
[70] Vladimir Kolmogorov and Ramin Zabih. What energy functions can be minimized via graph cuts? IEEE Transactions on Pattern Analysis and Machine Intelligence, 26(2):147-159, February 2004.
[71] Nikos Komodakis, Georgios Tziritas, and Nikos Paragios. Fast, approximately optimal solutions for single and dynamic MRFs. In IEEE Conf. Computer Vision and Pattern Recognition (CVPR), 2007.
[72] Nikos Komodakis, Georgios Tziritas, and Nikos Paragios. Performance vs computational efficiency for optimizing single and dynamic MRFs: Setting the state of the art with primal-dual strategies. Computer Vision and Image Understanding, 112(1):14-29, 2008. C++ source code available from http://www.csd.uoc.gr/~komod/FastPD/.
[73] Arie M. C. A. Koster, Stan P. M. van Hoesel, and Antoon W. J. Kolen. The partial constraint satisfaction problem: facets and lifting theorems. Operations Research Letters, 23:89-97, October 1998.
[74] H. A. Kramers and G. H. Wannier. Statistics of the two-dimensional ferromagnet. part I. Physical Review, 60:252-262, 1941.
[75] Frank Kschischang, Brendan J. Frey, and Hans-Andrea Loeliger. Factor graphs and the sumproduct algorithm. IEEE Transactions on Information Theory, 47(2):498-519, 2001.
[76] Sanjiv Kumar and Martial Hebert. Discriminative fields for modeling spatial dependencies in natural images. In S. Thrun, L. Saul, and B. Schölkopf, editors, Advances in Neural Information Processing Systems 16, 2004.
[77] Sanjiv Kumar and Martial Hebert. Discriminative random fields. International Journal of Computer Vision, 68(2):179-201, 2006.
[78] Kazimierz Kuratowski. Sur le problème des courbes gauches en topologie. In Fundamenta Mathematicae, volume 15, pages 217-283, 1930.
[79] J. D. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic modeling for segmenting and labeling sequence data. In Proceedings of International Conference on Machine Learning, volume 18, pages 282-289, San Francisco, CA, 2001. Morgan Kaufmann.
[80] David Landau and Kurt Binder. A Guide to Monte Carlo Simulations in Statistical Physics. Cambridge University Press, New York, NY, USA, 2005.
[81] Steffen L. Lauritzen and David J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. Journal of the Royal Statistical Society, B 50(2):157-224, 1988.
[82] Wilhelm Lenz. Beiträge zum Verständnis der Magnetischen Eigenschaften in festen Körpern. Physikalische Zeitschrift, 21:613-615, 1920.
[83] Vasilica Lepar and Prakash P. Shenoy. A comparison of Lauritzen-Spiegelhalter, Hugin, and Shenoy-Shafer architectures for computing marginals of probability distributions. In Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence, pages 328-337. Morgan Kaufmann, 1998.
[84] Frauke Liers and Gregor Pardella. A simple max-cut algorithm for planar graphs. Technical Report zaik2008-579, Zentrum für Angewandte Informatik, Universität Köln, September 2008.
[85] Charles H. C. Little. An extension of Kasteleyn's method of enumerating the 1-factors of planar graphs. In Combinatorial Mathematics, volume 403 of Lecture Notes in Mathematics, pages 63-72. Springer Berlin, Heidelberg, 1974.
[86] Dong C. Liu and Jorge Nocedal. On the limited memory BFGS method for large scale optimization. Mathematical Programming, 45(3):503-528, 1989.
[87] László Lovász and Michael D. Plummer. Matching Theory, volume 29 of Annals of Discrete Mathematics. North-Holland, Amsterdam, 1986.
[88] E. Marinari and G. Parisi. Simulated tempering: A new Monte Carlo scheme. Europhysics Letters, 19(6), 1992.
[89] Kurt Mehlhorn and Guido Schäfer. Implementation of $O(n m \log n)$ weighted matchings in general graphs: The power of data structures. ACM Journal of Experimental Algorithms, 7:138-148, 2002. Article and C source code (requires LEDA 4.2 or higher) at http://www.jea.acm.org/ 2002/MehlhornMatching/.
[90] Talya Meltzer, Chen Yanover, and Yair Weiss. Globally optimal solutions for energy minimization in stereo vision using reweighted belief propagation. In International Conference On Computer Vision, pages 428-435, 2005.
[91] Martin Müller. Computer Go. Artificial Intelligence, 134(1-2):145-179, 2002.
[92] Xiaozhen Niu and Martin Müller. An improved safety solver for Computer Go. In Computers and Games 2004, Ramat-Gan, Israel, pages 97-112. Springer-Verlag, 2006.
[93] Xiaozhen Niu and Martin Müller. An improved safety solver in Go using partial regions. In 6th International Conference on Computers and Games, pages 102-112. Springer-Verlag, 2008.
[94] NNGS. No Name Go Server, 1995-2005. http://senseis. xmp.net/?NNGS.
[95] Jorge Nocedal. Updating quasi-Newton matrices with limited storage. Mathematics of Computation, 35:773-782, 1980.
[96] Sergey Norin. Drawing Pfaffian graphs. In International Symposium on Graph Drawing, Lecture Notes in Computer Science, pages 371-376, 2005.
[97] Lars Onsager. Crystal statistics. I. a two-dimensional model with an order-disorder transition. Physical Review, 65:117-149, 1944.
[98] Gregor Pardella and Frauke Liers. Exact ground states of large two-dimensional planar Ising spin glasses. Physical Review E, 78(5):056705(10), November 2008.
[99] J. Pearl. Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufman, 1988.
[100] Rudolf Peierls. On Ising's model of ferromagnetism. In Proceedings of the Cambridge Philosophical Society, volume 32, pages 477-481, 1936.
[101] Renfrey B. Potts. Some generalized order-disorder transformations. Proceedings of the Cambridge Philosophical Society, 48:106-109, 1952.
[102] Renfrey B. Potts and John C. Ward. The combinatorial method and the two-dimensional Ising model. Progress of Theoretical Physics, 13:38-46, 1955.
[103] Matthew Richardson and Pedro Domingos. Mining knowledge-sharing sites for viral marketing. In Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 61-70, New York, NY, USA, 2002. ACM.
[104] Carsten Rother, Vladimir Kolmogorov, Andrew Blake, and Matthew Brown. GrabCut ground truth database. http://research.microsoft.com/en-us/um/cambridge/projects/ visionimagevideoediting/segmentation/grabcut.htm, 2007.
[105] Carsten Rother, Vladimir Kolmogorov, Victor Lempitsky, and Martin Szummer. Optimizing binary MRFs via extended roof duality. In IEEE Conference on Computer Vision and Pattern Recognition, Minneapolis, MN, June 2007.
[106] Lawrence K. Saul. Exact computations in the statistical mechanics of disordered systems. PhD thesis, Department of Physics, Massachusetts Institute of Technology, 1994.
[107] Lawrence K. Saul and Mehran Kardar. The 2D +/-J Ising spin glass: exact partition functions in polynomial time. Nuclear Physics B, 432:641-667, December 1994.
[108] Michail I. Schlesinger. Syntactic analysis of two-dimensional visual signals in noisy conditions. Kibernetika, 4:113-130, 1976. In Russian.
[109] Helga Schramm and Jochem Zowe. A version of the bundle idea for minimizing a nonsmooth function: Conceptual idea, convergence analysis, numerical results. SIAM J. Optimization, 2: 121-152, 1992.
[110] Nicol N. Schraudolph and Dmitry Kamenetsky. Efficient exact inference in planar Ising models. Technical Report 0810.4401, arXiv, October 2008. http://aps.arxiv.org/abs/0810. 4401.
[111] Nicol N. Schraudolph and Dmitry Kamenetsky. Efficient exact inference in planar Ising models. In D. Koller, Y. Bengio, D. Schuurmans, L. Bottou, and A. Culotta, editors, Advances in Neural Information Processing Systems 21, Cambridge, MA, 2009. MIT Press.
[112] Nicol N. Schraudolph, Peter Dayan, and Terrence J. Sejnowski. Learning to evaluate Go positions via temporal difference methods. In N. Baba and L. C. Jain, editors, Computational Intelligence in Games. Springer-Verlag, 2001.
[113] Nicol N. Schraudolph, Jin Yu, and Simon Günter. A stochastic quasi-Newton method for online convex optimization. In Proc. 11th Intl. Conf. Artificial Intelligence and Statistics (AIstats), San Juan, Puerto Rico, March 2007. Society for Artificial Intelligence and Statistics. http: //nic.schraudolph.org/pubs/SchYuGue07.pdf.
[114] Prakash P. Shenoy. Binary join trees for computing marginals in the Shenoy-Shafer architecture. International Journal of Approximate Reasoning, 17(2-3):239-263, 1997.
[115] Prakash P. Shenoy and Glenn Shafer. Propagating belief functions with local computations. IEEE Expert, 1:43-52, 1986.
[116] Prakash P. Shenoy and Glenn Shafer. Axioms for probability and belief-function propagation. In Uncertainty in Artificial Intelligence, pages 169-198. North-Holland, 1990.
[117] S. Sherman. Combinatorial aspects of the Ising model for ferromagnetism. I. a conjecture of Feynman on paths and graphs. Journal of Mathematical Physics, 1:202-217, 1960.
[118] N. J. A. Sloane. The on-line encyclopedia of integer sequences, 2009. http://www.research. att.com/~njas/sequences/.
[119] David H. Stern, Thore Graepel, and David J. C. MacKay. Modelling uncertainty in the game of Go. In Lawrence K. Saul, Yair Weiss, and Léon Bottou, editors, Advances in Neural Information Processing Systems 17, pages 1353-1360, Cambridge, MA, 2004. MIT Press.
[120] Robert H. Swendsen and Jian-Sheng Wang. Nonuniversal critical dynamics in Monte Carlo simulations. Physical Review Letters, 58(2):86-88, Jan 1987.
[121] Richard Szeliski, Ramin Zabih, Daniel Scharstein, Olga Veksler, Vladimir Kolmogorov, Aseem Agarwala, Marshall Tappen, and Carsten Rother. A comparative study of energy minimization methods for Markov random fields with smoothness-based priors. IEEE Trans. Pattern Analysis and Machince Intelligence, 30(6):1068-1080, 2008. C++ source code available from http: //vision.middlebury.edu/MRF/code/.
[122] Marshall F. Tappen and William T. Freeman. Comparison of graph cuts with belief propagation for stereo, using identical MRF parameters. In Proc. 9th IEEE Intl. Conf. Computer Vision (ICCV), pages 900-907, 2003.
[123] B. Taskar, C. Guestrin, and D. Koller. Max-margin Markov networks. In S. Thrun, L. Saul, and B. Schölkopf, editors, Advances in Neural Information Processing Systems 16, pages 25-32, Cambridge, MA, 2004. MIT Press.
[124] H. N. V. Temperley and M. Fisher. Dimer problem in statistical mechanics - an exact result. Philosophical Magazine, 6:1061-1063, August 1961.
[125] G. Tesler. Matching in graphs on non-orientable surfaces. Journal of Combinatorial Theory Series B, 78:198-231, March 1961.
[126] Creighton K. Thomas and A. Alan Middleton. Matching Kasteleyn cities for spin glass ground states. Physical Review B, 76(22):22406, 2007.
[127] Robin Thomas. A survey of Pfaffian orientations of graphs. International Congress of Mathematicians, 3:963-984, 2006.
[128] Carsten Thomassen. The graph genus problem is NP-complete. Journal of Algorithms, 10(4): 568-576, 1989.
[129] G. Toulouse. Theory of the frustration effect in spin glasses: I. Communications on Physics, 2: 115, 1977.
[130] Erik C. D. van der Werf, Hendrik Jaap van den Herik, and Jos W. H. M. Uiterwijk. Learning to estimate potential territory in the game of Go. In 4th International Conference on Computers and Games, pages 81-96. Springer-Verlag, 2004.
[131] Erik C. D. van der Werf, Hendrik Jaap van den Herik, and Jos W. H. M. Uiterwijk. Learning to score final positions in the game of Go. Theoretical Computer Science, 349(2):168-183, 2005.
[132] Erik C. D. van der Werf, Mark H. M. Winands, Hendrik Jaap van den Herik, and Jos W. H. M. Uiterwijk. Learning to predict life and death from Go game records. Information Science, 175 (4):258-272, 2005.
[133] J. Vannimenus and G. Toulouse. Theory of the frustration effect: II. Ising spins on a square lattice. Journal of Physics C: Solid state physics, 10:537-542, 1977.
[134] R. Vilà and T. Cazenave. When one eye is sufficient: A static classification. In H. Jaap van den Herik, Hiroyuki Iida, and Ernst A. Heinz, editors, Advances in Computer Games, volume 263 of IFIP, pages 109-124. Kluwer, 2003.
[135] S. V. N. Vishwanathan, Karsten Borgwardt, and Nicol N. Schraudolph. Fast computation of graph kernels. In B. Schölkopf, J. Platt, and T. Hofmann, editors, Advances in Neural Information Processing Systems 19, Cambridge MA, 2007. MIT Press.
[136] Martin J. Wainwright, Tommi S. Jaakkola, and Alan S. Willsky. MAP estimation via agreement on trees: Message-passing and linear programming. IEEE Transactions on Information Theory, 51(11):3697-3717, 2005.
[137] Yair Weiss. Belief propagation and revision in networks with loops. Technical Report AI memo 1616, CBCL paper 155, MIT, Department of Brain and Cognitive Sciences, November 1997.
[138] Arthur T. White and Lowell W. Beineke. Topological graph theory. In Lowell W. Beineke and Robin J. Wilson, editors, Selected Topics in Graph Theory, chapter 2, pages 15-49. Academic Press, 1978.
[139] Aaron Windsor. Planar graph functions for the Boost graph library. C++ source code available from http://boost-consulting.com/vault/index.php?directory=Algorithms/graph, 2007.
[140] David Wolfe. Go endgames are PSPACE-hard. In Richard Nowakowski, editor, More Games of No Chance, pages 125-136. Cambridge University Press, 2002.
[141] Jennifer Wortman. Viral marketing and the diffusion of trends on social networks. Technical Report MS-CIS-08-19, University of Pennsylvania, Department of Computer and Information Science, May 2008.
[142] Lin Wu and Pierre Baldi. A scalable machine learning approach to Go. In B. Schölkopf, J. Platt, and T. Hoffman, editors, Advances in Neural Information Processing Systems 19, pages 15211528. MIT Press, Cambridge, MA, 2007.
[143] Jonathan S. Yedidia, William T. Freeman, and Yair Weiss. Generalized belief propagation. In T. K. Leen, T. G. Dietterich, and V. Tresp, editors, Advances in Neural Information Processing Systems 13, pages 689-695. MIT Press, 2001.


[^0]:    ${ }^{1} \mathrm{C}++$ code implementing the algorithms described herein is available from http://nic.schraudolph.org/isinf/

[^1]:    ${ }^{2}$ We use two notations for permutations: two-line notation (2.3) and the cycle notation (2.4).

[^2]:    ${ }^{3}$ The Potts model [101] is a generalization of the Ising model which allows the spins to have more than 2 states

[^3]:    ${ }^{4}$ The recursion used in Algorithm 1 is for illustrative purposes only, since in practice such an implementation would rapidly run out of stack space. Our implementation of Algorithm 1 uses an iterative breadth-first-search.

[^4]:    ${ }^{5}$ This should not be confused with the junction tree (Definition 30).

[^5]:    ${ }^{6}$ For notational clarity we suppress here the fact that we are usually modeling a collection of data items. The objective function for such a set is simply the sum of objectives for each individual item in it.

[^6]:    ${ }^{7}$ See http://en.wikipedia.org/wiki/Pink_noise and http://en.wikipedia.org/wiki/White_noise for more details.
    ${ }^{8}$ The regularizer $\lambda$ was set to 2 .

[^7]:    ${ }^{9}$ For presentation purposes we show the segmented image, obtained by masking the original image with our binary segmentation.

[^8]:    ${ }^{10}$ To be rigorous, these definitions would have to assume perfect play on both sides. Except for very restricted situations, however, optimal play in Go is unknown; working definitions must therefore be relative to some notion of "reasonable" play. For instance, in Figure 6.4(a) black could rescue (12) and (17) by joining them to an alive block with two eyes, but only if white were to play quite irrationally. Less clear-cut cases are what makes Go territory prediction so challenging.

[^9]:    ${ }^{11} \mathrm{~A}$ surface of genus $g$ contains $2 g$ kinds of non-contractible cycles. By applying homology transformations to these cycles we can obtain every other cycle in the surface [28].

[^10]:    ${ }^{12}$ we need one axis for each of the three parameters, as well as another axis to display the value of the relative energy.

