1

Perfect graphs and graphical modeling

Tony Jebara
Department of Computer Science
Columbia University
New York, NY 10027

Appears in *Tractability*, Cambridge University Press, 2012

Tractability is the study of computational tasks with the goal of identifying which problem classes are *tractable* or, in other words, *efficiently solvable*. The class of tractable problems is traditionally assumed to be solvable in polynomial time by a *deterministic* Turing machine and is denoted by P. The class contains many natural tasks such as sorting a set of numbers, linear programming (the decision version), determining if a number is prime, and finding a maximum weight matching. Many interesting problems, however, lie in another class that generalizes P and is known as NP: the class of languages decidable in polynomial time on a *non-deterministic* Turing machine. We trivially have that P is a subset of NP (many researchers also believe that it is a strict subset). It is believed that many problems in the class NP are, in the worst case, intractable and do not admit efficient inference. Problems such as maximum stable set, the traveling salesman problem and graph coloring are known to be NP-hard (at least as hard as the hardest problems in NP). It is, therefore, widely suspected that there are no polynomial-time algorithms for NP-hard problems. Rather than stop after labeling a problem class as NP-hard by identifying its worst-case instances, this chapter explores the question: what *instances* of otherwise NP-hard problems still admit efficient inference? The study of perfect graphs (Berge, 1963) helps shed light on this question. It turns out that a variety of hard problems such as graph coloring, maximum clique and maximum stable set are all solvable efficiently in polynomial time when the input graph is restricted to be a perfect graph (Grötschel et al., 1988).

Can we extend this promising result to other important problems in applied fields such as data mining and machine learning? This chapter will focus on a central NP-hard problem in statistics and machine learning known as the maximum a posteriori (MAP) problem, in other

words, *finding the most likely outcome from a set of observations under a given probability distribution*. When can such inference be performed efficiently? Perfect graphs will be a useful tool in carving out what types of MAP estimation problems are tractable.

In the past two decades, the fields of machine learning and statistical inference have increasingly leveraged graph representations and graph-based algorithms in a variety of problem settings. For instance, semi-supervised learning, dimensionality reduction, and unsupervised clustering problems are frequently solved by casting data points as nodes in a graph and then applying graph-theoretic algorithms. Similarly, probabilistic inference problems such as MAP estimation, marginal inference, and so on are solved by casting random variables as nodes in a graphical model and then applying graph algorithms such as message passing, tree approximations and other variants. In many situations, most of the above learning problems are *intractable* and provably NP-hard in the worst case (Shimony, 1994; Aloise et al., 2009). However, when the graph structures are restricted to a subfamily (such as the family of tree-structured graphs), a variety of learning problems may become tractable (Pearl, 1988). Therefore, determining which graphs and graphical models admit efficient inference and which learning problems are solvable in polynomial time is of great importance in machine learning. Following upon previous work (Jebara, 2009), this chapter extends the tractability benefits of perfect graphs to statistical inference in graphical models and focuses on the MAP estimation problem as a task of particular interest.

1.1 Berge graphs and perfect graphs

A convenient starting point is the definition of *perfect graphs* according to their pioneer, Claude Berge (Berge, 1963). Consider the undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with n vertices $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$ and edges $\mathcal{E} : \mathcal{V} \times \mathcal{V} \to \mathbb{B}$. To determine if a graph is perfect, it is necessary to consider its induced subgraphs.

Definition 1.1 A subgraph $\mathcal{H}$ of a graph $\mathcal{G}$ is said to be induced if, for any pair of vertices u and v of $\mathcal{H}$, uv is an edge of $\mathcal{H}$ if and only if uv is an edge of $\mathcal{G}$. In other words, $\mathcal{H}$ is an induced subgraph of $\mathcal{G}$ if it has exactly the edges that appear in $\mathcal{G}$ over the same vertex set. If the vertex set of $\mathcal{H}$ is the subset S of the vertices of $\mathcal{G}$, then $\mathcal{H}$ can be written as $\mathcal{G}[S]$ and is said to be *induced* by S.

Perfect graphs must satisfy the following property involving their induced subgraphs.

Definition 1.2 (Perfect Graph) A graph $\mathcal{G}$ is perfect if and only if each of its induced subgraphs has chromatic number equal to clique number.

Thus, every induced subgraph $\mathcal{H} \subseteq \mathcal{G}$ in any perfect graph has chromatic number[1], $\chi(\mathcal{H})$, equal to its clique number, $\omega(\mathcal{H})$. For all graphs, it is easy to see that the clique number serves as a lower bound on the chromatic number, *i.e.* $\omega(\mathcal{G}) \leq \chi(\mathcal{G})$. For a perfect graph $\mathcal{G}$, this bound is met with equality ($\omega(\mathcal{H}) = \chi(\mathcal{H})$) for all its induced subgraphs $\mathcal{H} \subseteq \mathcal{G}$. Examples of perfect graphs are shown in Figure 1.1.

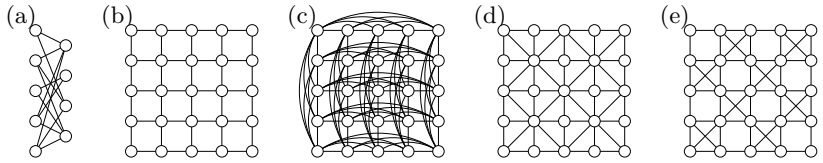


Figure 1.1 Examples of perfect graphs; (a) is bipartite; (b) is a bipartite grid graph; (c) is a Rook's graph; (d) has chromatic number equal to 3; and (e) has chromatic number equal to 4.

Berge also provided two conjectures along with the definition of perfect graphs. The weak perfect graph conjecture was resolved and is now known as the following theorem (Lovász, 1972).

Theorem 1.3 (Weak Perfect Graph Theorem) *A graph is perfect if and only if its complement is perfect.*

The complement of a graph is defined as follows.

Definition 1.4 (Graph Complement) The complement $\overline{\mathcal{G}}$ of a graph $\mathcal{G}$ is a graph with the same vertex set[2] as $\mathcal{G}$, where distinct vertices $u, v \in \mathsf{V}(\mathcal{G})$ are adjacent in $\overline{\mathcal{G}}$ if and only if they are not adjacent in $\mathcal{G}$.

[1] The chromatic number of a graph $\mathcal{G}$, $\chi(\mathcal{G})$, is the minimum number of colors needed to label vertices such that no adjacent vertices have the same color. The clique number of a graph $\mathcal{G}$, $\omega(\mathcal{G})$, is the size of a largest maximal clique (a largest subset of nodes in the graph that are all pairwise adjacent), *i.e.* $\omega(\mathcal{G}) = \max_{\mathbf{c} \in \mathcal{C}} |\mathbf{c}|$ where $\mathcal{C}$ is the set of all maximal-cliques.

[2] We use the notation $\mathsf{V}(\mathcal{G})$ to denote the vertex set $\mathcal{V}$ of a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$.

So far, it seems that deciding if a graph $\mathcal{G}$ is perfect is a daunting endeavor. One brute force approach is to consider each induced subgraph $\mathcal{H}$ of $\mathcal{G}$ and verify that its coloring number $\chi(\mathcal{H})$ equals its clique number $\omega(\mathcal{H})$. However, Berge also conjectured that perfect graphs are equivalent to another family of graphs defined by forbidding certain induced subgraphs. This family was later named *Berge graphs* (Berge, 1963; Berge and Ramírez-Alfonsín, 2001). A Berge graph $\mathcal{G}$ is a graph which has no odd holes and has no odd holes in its complement $\overline{\mathcal{G}}$. The notion of a hole in a graph is defined as follows.

Definition 1.5 (Hole) A hole of a graph $\mathcal{G}$ is an induced subgraph of $\mathcal{G}$ which is a chordless cycle of length at least 5. An odd (even) hole is a chordless cycle with odd (even) length.

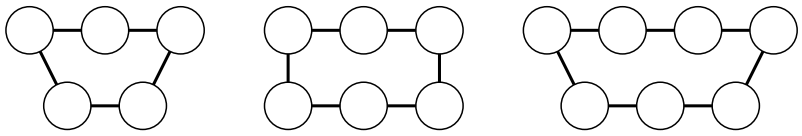


Figure 1.2 Various holes or chordless cycles of length 5 or larger.

Therefore, Berge graphs have no holes of length 5, 7, 9 and so on in the graph and the graph's complement. Figure 1.2 depicts a few examples of holes. Intuitively, it is easier to verify if a graph is Berge by checking for holes than it is to verify the chromatic number and clique number of all its induced subgraphs. The strong perfect graph conjecture proposed that a graph is perfect if and only if it is Berge. A formal proof of this conjecture was established recently and it is now known as the strong perfect graph theorem (Chudnovsky et al., 2006).

Theorem 1.6 (Strong Perfect Graph Theorem) *A graph is perfect if and only if it is Berge.*

The proof of the strong perfect graph theorem also establishes that all perfect graphs either are elements of the following basic classes:

- bipartite graphs
- complements of bipartite graphs
- line graphs of bipartite graphs
- complements of line graphs of bipartite graphs

- double split graphs

or admit one of the following structural decompositions:

- a 2-join
- a 2-join in the complement
- an M-join
- a balanced skew partition.

These decompositions are ways of repeatedly breaking up the graph such that eventually, the remaining parts are in one of the basic classes.

Definition 1.7 (Line Graph) The line graph $\mathsf{L}(\mathcal{G})$ of a graph $\mathcal{G}$ is a graph which contains a vertex for each edge of $\mathcal{G}$ and where two vertices of $\mathsf{L}(\mathcal{G})$ are adjacent if and only if they correspond to two edges of $\mathcal{G}$ with a common end vertex.

The study of perfect graphs involves beautiful combinatorics and is rich with theoretical results. For instance, some important rules have been proposed for adding nodes to perfect graphs while ensuring that perfection is preserved. One example is Lemma 1.8 which is illustrated in Figure 1.3 (Lovász, 1972).

Lemma 1.8 (Replication) *Let $\mathcal{G}$ be a perfect graph and let $v \in \mathsf{V}(\mathcal{G})$. Define a graph $\mathcal{G}'$ by adding to $\mathcal{G}$ a new vertex v' and joining it to v and all the neighbors of v. Then, $\mathcal{G}'$ is perfect.*

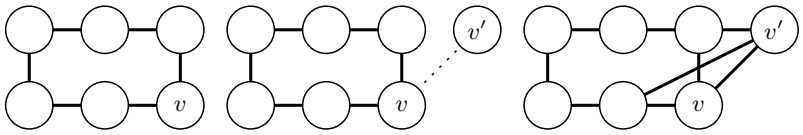


Figure 1.3 Replication. Given an initial perfect graph on the left, a new node can be introduced and attached to any other node v in the perfect graph. If the new node is also adjacent to all the neighbors of v, the resulting graph is also perfect.

Lemma 1.9 (Gluing on Cliques) *A graph $\mathcal{G} = \mathcal{G}_1 \cup \mathcal{G}_2$ is perfect if $\mathcal{G}_1 \cap \mathcal{G}_2$ is a clique and both $\mathcal{G}_1$ and $\mathcal{G}_2$ are perfect graphs.*

Another useful property is Lemma 1.9 which is illustrated in Figure 1.4. This property has been generalized into the so-called skew-partition decomposition (Berge and Chvátal, 1984; Chudnovsky et al.,

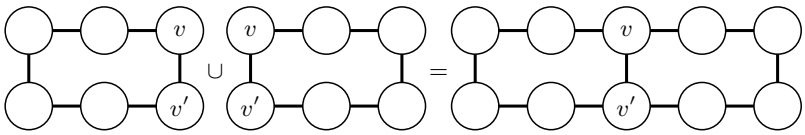


Figure 1.4 Gluing on cliques. The two graphs on the left are perfect graphs. The graph on the right is their union while their intersection is a clique cut-set. Therefore, the rightmost graph must also be perfect.

2006) which is a significantly more powerful tool yet is beyond the scope of this chapter. Perfect graphs have been the subject of much theoretical research. However, they also have fascinating computational properties which will be explored in the following sections.

1.2 Computational properties of perfect graphs

Perfect graphs enjoy very useful computational properties. For instance, it has recently been shown that recognizing if a graph is perfect (or not perfect) only requires polynomial time (Chudnovsky et al., 2005). Similarly, graph coloring[3], the maximum clique problem[4] and the maximum stable set[5] problem are NP-hard in general yet, remarkably, are solvable in polynomial time for perfect graphs (Grötschel et al., 1988). This chapter will focus on the implications perfect graphs have vis-a-vis two key computational tools in applied machine learning and optimization: linear programming and semidefinite programming.

1.2.1 Integral linear programs

Another valuable property is that perfect graphs can be used to construct linear programs which produce solutions that are provably integral. A linear program is an optimization over a vector of variables $\mathbf{x} \in \mathbb{R}^n$ in n-dimensional Euclidean space. A linear program finds the

[3] To color a graph $\mathcal{G}$ find the smallest set of assignment labels called "colors" such that no two adjacent vertices share the same color.
[4] The maximum clique of a graph $\mathcal{G}$ is a largest subset of nodes in $\mathcal{G}$ that are all pairwise adjacent.
[5] The maximum stable set of a graph $\mathcal{G}$ is a largest set of nodes in $\mathcal{G}$ no two of which are adjacent.

solution of the following constrained optimization problem

$$\max_{\mathbf{x}\in\mathbb{R}^n} \mathbf{f}^\top \mathbf{x} \text{ subject to } \mathbf{A}\mathbf{x} \leq \mathbf{b} \tag{1.1}$$

where $\mathbf{f} \in \mathbb{R}^n$, $\mathbf{A} \in \mathbb{R}^{m\times n}$ and $\mathbf{b} \in \mathbb{R}^m$ are inputs that are specified by the user. It is known that solving such linear programs is in P and algorithms are available that recover the optimal $\mathbf{x}^*$ value in $\mathcal{O}(\sqrt{m}n^3)$ time. This recovered solution $\mathbf{x}^*$ generally contains real values.

In contrast, consider the *binary* linear program

$$\max_{\mathbf{x}\in\mathbb{B}^n} \mathbf{f}^\top \mathbf{x} \text{ s.t. } \mathbf{A}\mathbf{x} \leq \mathbf{b}$$

where the solution vector is forced to be binary, in other words $\mathbf{x} \in \mathbb{B}^n$. This problem is notoriously NP-hard in the worst case and no efficient general solver is available (Karp, 1972).

Interestingly, if we require the constraints matrix $\mathbf{A}$ to have a special form, specifically if it is a *perfect matrix* (see below), then we can further guarantee that the solution $\mathbf{x}^*$ obtained by standard linear programming will only have binary values. Consider the following rearranged form of Equation 1.1 known as a set-packing linear program,

$$\max_{\mathbf{x}\in\mathbb{R}^n, \mathbf{x}\geq\mathbf{0}} \mathbf{f}^\top \mathbf{x} \text{ s.t. } \mathbf{A}\mathbf{x} \leq \mathbf{1}$$

where $\mathbf{0} \in \mathbb{R}^n$ is a vector of all zeros and $\mathbf{1} \in \mathbb{R}^m$ is a vector of all ones. The following theorem shows how standard linear programming can be used to solve certain binary programs (Lovász, 1972; Chvátal, 1975).

Theorem 1.10 *For every non-negative vector $\mathbf{f} \in \mathbb{R}^n$, the linear program*

$$\max_{\mathbf{x}\in\mathbb{R}^n, \mathbf{x}\geq\mathbf{0}} \mathbf{f}^\top \mathbf{x} \text{ s.t. } \mathbf{A}\mathbf{x} \leq \mathbf{1}$$

recovers a vector $\mathbf{x}^*$ which is integral if and only if the (undominated) rows of $\mathbf{A}$ form the vertex-versus-maximal cliques incidence matrix of some perfect graph.

The constraint matrix $\mathbf{A}$ is obtained from a graph $\mathcal{G}$ according to the following definition.

Definition 1.11 (Undominated Incidence Matrix) The undominated incidence matrix of a graph $\mathcal{G}$ with vertices $\{v_1, \ldots, v_n\}$ and maximal cliques $\mathcal{C} = \{\mathbf{c}_1, \ldots, \mathbf{c}_m\}$ is matrix $\mathbf{A} \in \mathbb{B}^{m\times n}$ where $\mathbf{A}_{ij} = 1$ if $v_i \in \mathbf{c}_j$ and $\mathbf{A}_{ij} = 0$ otherwise.

A *perfect matrix* is simply the undominated incidence matrix $\mathbf{A}$ obtained from a graph $\mathcal{G}$ which is perfect. In the following sections, this property will be leveraged for MAP estimation with graphical models.

1.2.2 Lovász theta function

The Lovász theta function $\vartheta(\mathcal{G}) \in \mathbb{R}$ accepts as input a graph $\mathcal{G}$ and outputs a non-negative scalar (Lovász, 1979; Knuth, 1994). For any graph, it is computable in polynomial time. Modern solvers can recover $\vartheta(\mathcal{G})$ within a multiplicative approximation factor of $(1+\epsilon)$ in time $\mathcal{O}(\epsilon^{-2}n^5 \log n)$ using primal-dual methods (Chan et al., 2009). The following straightforward semidefinite program recovers the Lovász theta function

$$\vartheta(\mathcal{G}) = \max_{\mathbf{M} \in \mathbb{R}^{n \times n}, \mathbf{M} \succeq \mathbf{0}} \sum_{ij} \mathbf{M}_{ij} \text{ s.t. } \sum_{i} \mathbf{M}_{ii} = 1, \ \mathbf{M}_{ij} = 0 \, \forall (i,j) \in \mathcal{E}.$$

Remarkably, the Lovász number satisfies the *sandwich* property:

$$\omega(\mathcal{G}) \leq \vartheta(\overline{\mathcal{G}}) \leq \chi(\mathcal{G}). \tag{1.2}$$

Since perfect graphs satisfy $\omega(\mathcal{G}) = \chi(\mathcal{G})$, it is always possible to find the coloring number (or the maximum clique size) efficiently for a perfect graph $\mathcal{G}$ simply by computing the Lovász theta function on its complement. Otherwise, in general, the coloring number can be intractable to compute in the worst case. In the following sections, we shall illustrate how the Lovász theta function can be useful for MAP estimation as well.

1.3 Graphical models

Great strides have been made by the combinatorics community in the study of perfect graphs. How does this progress advance machine learning and statistical inference? These fields also use graphs to represent problems and use graph-theoretic algorithms to solve them. In machine learning and statistical inference, graphs are extensively used in the study of *graphical models*. A graphical model represents the factorization of a probability density function (Wainwright and Jordan, 2008). This chapter will focus on the factor graph notation for graphical models and use the non-calligraphic font to distinguish these graphs from those discussed in the previous sections. A factor graph is a bipartite

graph $G = (V, W, E)$ with *variable* vertices $V = \{1, \ldots, k\}$, *factor* vertices $W = \{1, \ldots, l\}$ and a set of edges E between V and W. In addition, define the universe of discrete random variables $Y = \{y_1, \ldots, y_k\}$ each associated with an element of V and define a set of strictly positive[6] *potential functions* $\Psi = \{\psi_1, \ldots, \psi_l\}$ each associated with an element of W. We will often use the term *factor* and *potential function* interchangeably when re refer to $c \in W$, $\psi_c \in \Psi$ or the arguments of ψ_c. Also, for $j \in V$, each $y_j \in \mathbb{N}$ is a discrete random variable with $|y_j|$ possible configurations[7], in other words $y_j \in \{0, \ldots, (|y_j| - 1)\}$. The factor graph implies that the function $p(Y)$ factorizes as

$$p(y_1, \ldots, y_k) \;=\; \tfrac{1}{Z} \prod_{c \in W} \psi_c(Y_c) \tag{1.3}$$

where Z is the partition function[8] and Y_c is a subset of the random variables that are associated with the neighbors of node c. In other words, $Y_c = \{y_j | j \in \mathsf{Ne}(c)\}$ where $\mathsf{Ne}(c)$ is the set of vertices that are neighbors of the factor c with potential function ψ_c. Thus, each potential $\psi_c : Y_c \to \mathbb{R}^+$ is a function over the corresponding set of random variables Y_c. Note that it is possible to divide each ψ_c function by an arbitrary scalar $\gamma_c > 0$ without changing $p(Y)$ (the partition function Z has to be adjusted accordingly, of course). An example factor graph is shown in Figure 1.5.

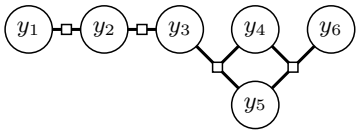


Figure 1.5 An example of a factor graph representing the factorization of a probability distribution $p(y_1, y_2, y_3, y_4, y_5, y_6) = \frac{1}{Z}\psi_{1,2}(y_1, y_2)\psi_{2,3}(y_2, y_3)\psi_{3,4,5}(y_3, y_4, y_5)\psi_{4,5,6}(y_4, y_5, y_6)$. The round nodes represent random variables while square nodes represent potential functions.

[6] This article assumes strictly positive potential functions $\psi_c > 0, \forall c$. Note that it is always possible to rewrite $p(Y)$ as an equivalent pairwise Markov random field (MRF) over binary variables (Yedidia et al., 2001).

[7] We will abuse notation and take the symbol $|\cdot|$ to mean the largest possible value an integer can achieve as well as use $|\cdot|$ in the traditional sense to indicate the cardinality or size of a set. The meaning of the operator should be clear from the context.

[8] The partition function Z is set such that $\sum_{y_1} \cdots \sum_{y_k} p(y_1, \ldots, y_k) = 1$.

A canonical problem that one aims to solve with graphical models is *maximum a posteriori* estimation. Many tasks in image processing, coding, protein folding and more can be cast as MAP estimation which, given a factor graph and potential functions, recovers $\arg\max_Y p(Y)$.

The MAP problem is, in the worst case, NP-hard (Shimony, 1994) and may involve exponential work in the storage size of the input graphical model. However, certain types of graphical models do admit exact inference efficiently. For example, MAP estimation in (junction) tree graphical models is efficient (Pearl, 1988). Similarly, graphical models where the potential functions $\psi_1, \ldots, \psi_l$ are submodular also admit efficient MAP estimation (Greig et al., 1989; Kolmogorov and Zabih, 2004). In many situations, the requisite algorithms are implemented using message passing schemes (such as max-product and variants of max-product), linear programming or network-flow solvers (Kolmogorov and Wainwright, 2005; Globerson and Jaakkola, 2007; Kolmogorov and Zabih, 2004). It is also sometimes possible to consider linear programming relaxations or message passing to potentially approximate the MAP solution when exact inference is intractable (Globerson and Jaakkola, 2007). However, *how* one goes about building such linear programming relaxations will affect the ultimate effectiveness of linear programming solvers (Komodakis and Paragios, 2008). This chapter will propose a method of compiling the MAP estimation problem into a maximum weight stable set problem (MWSS) which is defined below. The MWSS is also referred to as the maximum weight independent set problem (MWIS). These are hard problems in the worst case but remain tractable if the input graph they are applied to is perfect (Grötschel et al., 1981).

1.4 Nand Markov random fields

Consider compiling the graphical model G above into another representation which will be referred to as a nand Markov random field (NMRF). A NMRF is a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with a set of binary variables $\mathcal{X}$ and a set of scalar weights $\mathcal{F}$. For each vertex $v \in \mathcal{V}$, there is one binary variable $x \in \mathcal{X}$ and one scalar weight $f \in \mathcal{F}$ associated with v. More precisely, the set $\mathcal{X}$ (resp. $\mathcal{F}$) contains one Boolean $x_{c,i}$ (resp. non-negative scalar $f_{c,i}$) for each factor $c \in W$ and for each configuration $i \in Y_c$ whenever $\psi_c(Y_c) > 0$. We say that two configurations $x_{c,i}$ and $x_{c',i'}$ are *incompatible* if the configurations i and i' imply different settings of the shared variables in $Y_c \cap Y_{c'}$. The NMRF has edges $\mathcal{E}$ between all pairs of config-

urations that are *incompatible*. The edges imply incompatibility since at most one node at the base of an edge can be instantiated while ensuring that the graphical model is in a consistent configuration (otherwise, the variables in $Y_c \cap Y_{c'}$ have to be in two configurations simultaneously which is impossible). Thus, each edge represents a nand relationship which we write as $\delta[x_{c,i} + x_{c',i'} \leq 1]$ where $\delta[\cdot] = 1$ if the statement inside the brackets is true and $\delta[\cdot] = 0$ otherwise. Consider Algorithm 1 which compiles a graphical model into a NMRF.

Algorithm 1 CompileIntoNMRF

Input factor graph $G = (V, W, E)$ with positive $\psi_1(Y_1), \ldots, \psi_l(Y_l)$

Initialize graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ as an empty graph, $\mathcal{X} = \{\}$, and $\mathcal{F} = \{\}$
For each factor $c \in W$
  Set γ_c to the smallest value of $\psi_c(Y_c)$
  For each configuration i of the set of variables Y_c
    Add a vertex $v_{c,i}$ to $\mathcal{V}$
    Add a Boolean $x_{c,i} \in \mathbb{B}$ associated with $v_{c,i}$ to $\mathcal{X}$
    Add a weight $f_{c,i} \in \mathbb{R}$ associated with $v_{c,i}$ to $\mathcal{F}$
    Set $f_{c,i} = \log(\psi_c(Y_c = i)/\gamma_c)$
    For each $v_{c',i'} \in \mathcal{V}$ that is incompatible with $v_{c,i}$
      Add an edge between $v_{c,i}$ and $v_{c',i'}$ to $\mathcal{E}$

Output nand Markov random field $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, $\mathcal{X}$, and $\mathcal{F}$

Let $n = |\mathcal{V}|$ be the total number of nodes in the NMRF output by Algorithm 1. The (unnormalized) probability associated with the NMRF factorizes as follows:

$$\rho(\mathcal{X}) = \prod_{(c,i) \in \mathcal{V}} \exp(f_{c,i} x_{c,i}) \prod_{((c,i),(c',i')) \in \mathcal{E}} (1 - x_{c,i} x_{c',i'}). \tag{1.4}$$

The nand constraints prevent variables $x_{c,i}$ and $x_{c',i'}$ from both being set to 1 if they share an edge (this drives $\rho(\mathcal{X})$ to zero). We wish to set some of the Boolean entries in $\mathcal{X}$ to 1 to maximize the value of ρ. This is equivalent to maximizing $\sum_{(i,c) \in \mathcal{V}} f_{c,i} x_{c,i}$ while enforcing the nand constraints. This is a *maximum weight stable set* MWSS problem which is a weighted variant of the *maximum stable set* (MSS) problem.

Definition 1.12 (Maximum Stable Set) The maximum stable set of a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a largest subset of nodes in $\mathcal{G}$ such that no two nodes in the subset are pairwise adjacent.

The MSS problem is merely a MWSS problem where all the weights are constant. Thus, the MWSS strictly generalizes the MSS.

Definition 1.13 (Maximum Weight Stable Set) The maximum weight stable set of a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with non-negative weights $\mathcal{F}$ is a subset of nodes in $\mathcal{G}$ with largest total weight such that no two nodes in the subset are pairwise adjacent.

More precisely, to avoid problems in certain special cases, we will be interested in solving a *maximal maximum weight stable set* (MMWSS) problem on the NMRF. This not only maximizes Equation 1.4 but, in the case of ties and in the case where some weights in $\mathcal{F}$ are exactly zero, the MMWSS also selects the maximizer which sets the most possible Boolean variables to 1. Thus, the MMWSS strictly generalizes the MWSS.

Definition 1.14 (Maximal Maximum Weight Stable Set) The maximal maximum weight stable set of a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with non-negative weights $\mathcal{F}$ is a maximum weight stable set of $\mathcal{G}$ with largest cardinality.

Assume we have recovered the MMWSS of the NMRF $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with weights $\mathcal{F}$. This solution can be denoted by $\mathcal{V}^* \subseteq \mathcal{V}$ or by a set of Booleans $\mathcal{X}^*$. In the Boolean representation $\mathcal{X}^*$, for each $v \in \mathcal{V}$, its corresponding Boolean variable $x \in \mathcal{X}^*$ is set to 1 if $v \in \mathcal{V}^*$ and is set to 0 otherwise. Does the solution recovered by the MMWSS of $\mathcal{G}$ coincide with the MAP estimate of $p(Y)$? It is clear that the Booleans in $\mathcal{X}$ span a superset (a relaxation) of the set of valid configurations of Y since $\mathcal{X}$ can represent inconsistent configurations of Y. In other words, there is a surjective relation between Y and $\mathcal{X}$: each setting of Y can be mapped to a unique corresponding setting in $\mathcal{X}$ yet the converse is not true. This leads to an important question: can $\mathcal{X}^*$ be used to recover a valid Y^* configuration which coincides with the MAP estimate, in other words $p(Y^*) = \max_Y p(Y)$? Theorem 1.16 shows that $\mathcal{X}^*$ indeed produces the MAP estimate. The theorem leverages the following lemma which shows that the MMWSS of $\mathcal{G}$ will always produce a unique assignment per factor, in other words, $\sum_i x_{c,i} = 1$ for each $c \in W$.

Lemma 1.15 *The maximal maximum weight stable set $\mathcal{X}^*$ of a NMRF graph $\mathcal{G}$ satisfies $\sum_i x^*_{c,i} = 1$ for each factor $c \in W$.*

Proof Setting all the Boolean variables in $\mathcal{X}$ to zero produces a value $\rho = 1$ in Equation 1.4. For a non-trivial NMRF, at least one of the

weights in $\mathcal{F}$ is strictly positive. Set the Boolean variable that corresponds to this strictly positive weight to 1 while keeping the rest of $\mathcal{X}$ set to zero. This produces a $\rho(\mathcal{X}) > 1$ since $\prod_{((c,i),(c',i'))\in\mathcal{E}}(1 - x_{c,i}x_{c',i'})$ equals 1 if only one Boolean equals 1 in $\mathcal{X}$. Exponentiating a strictly positive weight produces a value $\prod_{(c,i)\in\mathcal{V}} \exp(f_{c,i}x_{c,i})$ that is strictly larger than 1. Therefore, the MMWSS $\mathcal{X}^*$ must achieve $\rho(\mathcal{X}^*) > 1$. Clearly, there can be no incompatible configurations in the MMWSS since $\prod_{((c,i),(c',i'))\in\mathcal{E}}(1 - x^*_{c,i}x^*_{c',i'})$ would go to zero which contradicts the fact that $\rho(\mathcal{X}^*) > 1$. Next note that, for $\mathcal{X}^*$ and for each $c \in W$, one of the following must hold: (i) $\sum_i x^*_{c,i} > 1$, (ii) $\sum_i x^*_{c,i} = 0$ or (iii) $\sum_i x^*_{c,i} = 1$. Since Algorithm 1 places edges between disagreeing configurations, the term $\prod_{((c,i),(c',i'))\in\mathcal{E}}(1 - x_{c,i}x_{c',i'})$ would go to zero whenever condition (i) is met since two incompatible Boolean variables would have to be simultaneously asserted. Since the MAP estimate achieves $\rho(\mathcal{X}^*) > 1$, condition (i) cannot hold. Consider a factor c where condition (ii) is met. Since there are no incompatibilities in the MMWSS, for any such factor there is always a configuration j that may be selected which agrees with neighboring factors. If the value of $f_{c,j} > 0$, it is possible to preserve compatibility by setting $x_{c,j} = 1$ to strictly increase $\rho(\mathcal{X}^*)$. However, the MMWSS achieves the maximal $\rho(\mathcal{X}^*)$ value possible so the objective cannot be strictly increased. If the value of $f_{c,j} = 0$, the MMWSS will still always set $x_{c,j} = 1$ since this increases the cardinality of the solution. Thus, condition (ii) cannot hold for $\mathcal{X}^*$ either. This leaves condition (iii) as the only valid possibility. Therefore, the MMWSS satisfies $\sum_i x^*_{c,i} = 1$ for all factors $c \in W$. □

Theorem 1.16 *The MMWSS of a NMRF finds the MAP estimate of Equation 1.3 for a graphical model with strictly positive potential functions.*

Proof Lemma 1.15 shows that the MAP estimate of Equation 1.4 produces $\sum_i x^*_{c,i} = 1$ for each $c \in W$. So only a single setting, say $x_{c,\hat{i}} = 1$, is asserted for each c and the variables Y_c involved in factor c are in a single configuration $Y_c = \hat{i}$. Therefore, a consistent setting of the discrete random variables Y^* can be recovered from $\mathcal{X}^*$. Since $\rho(\mathcal{X}^*) \geq \rho(\mathcal{X})$ for all $\mathcal{X}$, since $p(Y) \propto \rho(X)$ for all Y, and since $\mathcal{X}$ is a superset of the configurations of Y, it must be the case that $p(Y^*) \geq p(Y)$ for all Y. □

Thus, we have shown how to compile any graphical model into a NMRF. The maximal maximum weight stable set of the NMRF can then be found to obtain $\mathcal{X}^*$. Finally, it is straightforward to recover

a consistent solution Y^* from $\mathcal{X}^*$. Algorithm 2 describes the MMWSS procedure more precisely.

Algorithm 2 FINDMMWSS

Input graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with weights $\mathcal{F}$

Initialize $\mathcal{V}_0 = \{\}$
For each $v \in \mathcal{V}$
  If the weight f associated to v is zero, add v to $\mathcal{V}_0$
Find MWSS on induced subgraph $\mathcal{G}_1 = \mathcal{G}[\mathcal{V} \setminus \mathcal{V}_0]$ and store it in $\hat{\mathcal{V}}_1$
For each $v \in \mathcal{V}_0$
  If v is adjacent to any node in $\hat{\mathcal{V}}_1$ remove v from $\mathcal{V}_0$
Find MSS on induced subgraph $\mathcal{G}_0 = \mathcal{G}[\mathcal{V}_0]$ and store it in $\hat{\mathcal{V}}_0$

Output $\hat{\mathcal{V}}_0 \cup \hat{\mathcal{V}}_1$

In Algorithm 2, it turns out that the maximal maximum weight stable set problem merely requires the solution of a MWSS sub-problem and a MSS sub-problem with some minor book-keeping. The algorithm accepts a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with weights $\mathcal{F}$ and finds a maximal maximum weight stable set. It does so by first discarding any nodes with zero weight (these are denoted by $\mathcal{V}_0$). It then solves the MWSS on the induced subgraph $\mathcal{G}_1$ over the remaining nodes $\mathcal{V} \setminus \mathcal{V}_0$ (using the corresponding weights in $\mathcal{F}$). Given a MWSS solution $\hat{\mathcal{V}}_1$ on $\mathcal{G}_1$ we can extend the solution to $\mathcal{G}$ by deleting any nodes in $\mathcal{V}_0$ which are adjacent to $\hat{\mathcal{V}}_1$. The remaining nodes in $\mathcal{V}_0$ are then non-adjacent to $\hat{\mathcal{V}}_1$ and another MSS solution $\hat{\mathcal{V}}_0$ can be recovered from the induced subgraph $\mathcal{G}_0 = \mathcal{G}[\mathcal{V}_0]$ separately. The final MMWSS solution is the union of both stable sets, $\hat{\mathcal{V}} = \hat{\mathcal{V}}_0 \cup \hat{\mathcal{V}}_1$. By solving MWSS sub-problems separately on two induced sub-graphs of $\mathcal{G}$ (namely $\mathcal{G}_1$ and $\mathcal{G}_0$), the overall MMWSS problem may potentially remain tractable. As long as $\mathcal{G}_1$ and $\mathcal{G}_0$ are perfect graphs, Algorithm 2 is efficient even if the original graph $\mathcal{G}$ itself is a non-perfect graph. Furthermore, since $\mathcal{G}_1$ and $\mathcal{G}_0$ are smaller than $\mathcal{G}$, this approach is also potentially faster even if $\mathcal{G}$ is perfect.

Having shown how to solve the MMWSS problem by solving two MWSS sub-problems (a standard MWSS problem and a MWSS problem with constant weights), we next discuss the implementation of the required MWSS solvers. Further, we ask: what graphical models have easy solutions such that the required MWSS steps of Algorithm 2 remain efficient? Hopefully, the MWSS solvers only operate on perfect subgraphs so that they remain efficient.

1.5 Maximum weight stable set

The previous section has shown that MAP estimation with a graphical model reduces to two maximum weight stable set sub-problems. The MWSS problem, however, is NP-hard in the worst case. Fortunately, it can be solved efficiently when the input graph is perfect. Another family of graphs where MWSS can be efficiently solved is the family of claw-free graphs. The MWSS problem takes as input a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with a set of non-negative weights $\mathcal{F}$ and outputs a subset of nodes with maximal total weight such that no two nodes are adjacent in $\mathcal{G}$. We will show three approaches that solve such a problem when $\mathcal{G}$ is a perfect graph. To solve a maximum stable set (MSS) problem, these three implementations are simply provided with constant weights in $\mathcal{F}$.

1.5.1 Linear programming

One possibility is to use linear programming to solve the MWSS sub-problems in Algorithm 2. The linear program is used to estimate $n = |\mathcal{V}|$ variables as follows. First, obtain the vertex-versus-maximal cliques incidence matrix $\mathbf{A} \in \mathbb{B}^{m \times n}$ from graph $\mathcal{G}$ as in Definition 1.11. In general, this requires using an algorithm to find all the cliques $\mathcal{C} = \{\mathbf{c}_1, \ldots, \mathbf{c}_m\}$ in the graph $\mathcal{G}$ (Bron and Kerbosch, 1973; Tomita et al., 2006). These cliques are then used to construct $\mathbf{A}$. We also rewrite the set of weights $\mathcal{F}$ as a vector $\mathbf{f} \in \mathbb{R}^n$. Then, solve the following linear program

$$\max_{\mathbf{x} \in \mathbb{R}^n, \mathbf{x} \geq \mathbf{0}} \mathbf{f}^\top \mathbf{x} \text{ s.t. } \mathbf{A}\mathbf{x} \leq \mathbf{1} \tag{1.5}$$

to obtain $\mathbf{x}^* \in \mathbb{R}^n$. If the graph $\mathcal{G}$ is perfect, the solution vector $\mathbf{x}^*$ is binary and can be immediately written as a solution over the set of Boolean variables $\mathcal{X}^*$. These, in turn, can easily be used to recover the discrete random variables Y^* in the original graphical model. The runtime of the linear program is $\mathcal{O}(\sqrt{m}n^3)$. The runtime of the clique-finding algorithm is $\mathcal{O}(m)$ however, this component may not be necessary to run each time if the input to the MWSS solver has the same graph topology $\mathcal{G}$ and only the weights $\mathcal{F}$ are allowed to vary. As long as m and n are not too large, such a MWSS solver can be fast in practice.

1.5.2 Message passing

In situations where n is large, linear programming may be unacceptably slow. Furthermore, in situations where computation needs to be distributed across many devices, linear programming may be impractical as it requires centralized storage and computation. An alternative approach is to use message-passing techniques (Wainwright and Jordan, 2008) which iteratively send messages between nodes and factors in a graphical model to converge to the MAP estimate. In our case, message-passing will be used to find the MWSS solution in Algorithm 2. The most popular message-passing method is the max-product (MP) algorithm (Weiss and Freeman, 2001). A more convergent variant is the max-product-linear-programming (MPLP) algorithm which solves linear programs in the dual via coordinate descent (Globerson and Jaakkola, 2007; Jebara, 2009). The MPLP implementation for solving a MWSS problem is depicted in Algorithm 3. Surprisingly, for the MWSS problem, the MPLP algorithm gives the same update rule as the MP algorithm dampened by a factor of $\frac{1-|\mathbf{c}|}{|\mathbf{c}|}$.

Algorithm 3 SOLVEMWSSVIAMPLP

Input: $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, cliques $\mathcal{C} = \{\mathbf{c}_1, \ldots, \mathbf{c}_m\}$ and weights f_i for $i \in \mathcal{V}$

Initialize $\lambda_{i,\mathbf{c}}^0 \in \mathbb{R}^+$ arbitrarily for $i \in \mathcal{V}$ and $\mathbf{c} \in \mathcal{C}$

Until converged do

Randomly choose $i \in \mathcal{V}$ and $\mathbf{c} \in \mathcal{C}$

Set $\lambda_{i,\mathbf{c}}{}^{(t+1)} = \frac{1-|\mathbf{c}|}{|\mathbf{c}|} \sum_{\mathbf{c}' \in \mathcal{C} \setminus \mathbf{c}: i \in \mathbf{c}'} \lambda_{i,\mathbf{c}'}^{(t)} + \frac{1}{|\mathbf{c}|} \frac{f_i}{\sum_{\mathbf{c} \in \mathcal{C}} [i \in \mathbf{c}]}$

$$- \frac{1}{|\mathbf{c}|} \max\left[0, \max_{i' \in \mathbf{c} \setminus i}\left[\frac{f_{i'}}{\sum_{\mathbf{c} \in \mathcal{C}} [i' \in \mathbf{c}]} + \sum_{\mathbf{c}' \in \mathcal{C} \setminus \mathbf{c}: i' \in \mathbf{c}'} \lambda_{i',\mathbf{c}'}^{(t)}\right]\right]$$

Output: $x_i^* = \sum_{\mathbf{c} \in \mathcal{C}: i \in \mathbf{c}} \lambda_{i,\mathbf{c}}{}^{(t)}$ for $i \in \mathcal{V}$

The message passing steps underlying Algorithm 3 perform coordinate descent in the dual of a linear program. It was later shown that the MPLP algorithm is performing dual coordinate descent in a different linear program rather than the one shown in Equation 1.5. A message passing scheme which repairs this problem was later proposed (Foulds et al., 2011). It works directly with the dual of the set-packing linear program in Equation 1.5, also known as a covering linear program:

$$\min_{\mathbf{z} \in \mathbb{R}^m, \mathbf{z} \geq \mathbf{0}} \mathbf{1}^\top \mathbf{z} \text{ s.t. } \mathbf{A}^\top \mathbf{z} \geq \mathbf{f}.$$

Coordinate descent in this dual can be implemented as a message-passing

scheme with better convergence properties and seems to consistently agree with the output of the set-packing linear program (Foulds et al., 2011). This improved message-passing solver is known as CD2MP (pairwise coordinate descent message passing) and is summarized in Algorithm 4. Given the dual solution, it is straightforward to recover $\mathcal{X}^* = \{x_1^*, \ldots, x_n^*\}$. Recall that, by complementary slackness of the dual linear programs, whenever $z_j^* > 0$ the corresponding constraint in the primal is achieved with equality, in other words: $\sum_{i=1}^{n} \mathbf{A}_{ij} x_i^* = 1$.

Algorithm 4 SolveMWSSviaCD2MP

Input: $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, cliques $\mathcal{C} = \{\mathbf{c}_1, \ldots, \mathbf{c}_m\}$ and weights f_i for $i \in \mathcal{V}$

Initialize $z_j = \max_{i \in \mathbf{c}_j} \frac{f_i}{\sum_{\mathbf{c} \in \mathcal{C}} [i \in \mathbf{c}]}$ for $j \in \{1, \ldots, m\}$

Until converged do

- Randomly choose $a \neq b \in \{1, \ldots, m\}$
- Compute $h_i = \max\left(0, \left(f_i - \sum_{j : i \in \mathbf{c}_j, j \neq a,b} z_j\right)\right)$ for $i \in \mathbf{c}_a \cup \mathbf{c}_b$
- Compute $s_a = \max_{i \in \mathbf{c}_a \setminus \mathbf{c}_b} h_i$
- Compute $s_b = \max_{i \in \mathbf{c}_b \setminus \mathbf{c}_a} h_i$
- Compute $s_{ab} = \max_{i \in \mathbf{c}_a \cap \mathbf{c}_b} h_i$
- Update $z_a = \max\left[s_a, \frac{1}{2}(s_a - s_b + s_{ab})\right]$
- Update $z_b = \max\left[s_b, \frac{1}{2}(s_b - s_a + s_{ab})\right]$

Output: $\mathbf{z}^* = [z_1, \ldots, z_m]^\top$

1.5.3 Semidefinite programming

In many cases, the linear programming and message passing methods above may not be practical. This is because the number of maximal cliques m in a graph $\mathcal{G}$ with n vertices can equal or exceed $2^{n/2}$. Thus, regardless of how quickly one can enumerate the maximal cliques in a graph, the computations underlying linear programming and message passing may still require exponential work in the worst case (even if the input graph is perfect). In those situations, it is best to avoid clique enumeration altogether and instead use the Lovász theta function to directly solve the MWSS problem (Yildirim and Fan-Orzechowski, 2006). Such semidefinite programming methods can require as little as $\tilde{\mathcal{O}}(n^5)$ time and are potentially more efficient than linear programming and message passing methods, particularly if $m \geq \mathcal{O}(n^4)$.

Recall that one may solve the MWSS problem on a weighted graph $\mathcal{G}$ by finding the maximum weight clique in the graph $\overline{\mathcal{G}}$. For perfect graphs,

the Lovász theta-function $\vartheta(\overline{\mathcal{G}})$ computes the weight of the maximum clique. Thus, we can recover the size of the maximal stable set via $\vartheta(\mathcal{G})$.

Since we are dealing with weighted graphs (with weights on nodes), however, consider the *weighted* Lovász theta-function which accepts a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with weights $\mathcal{F} = \{f_1, \ldots, f_n\}$ where $n = |\mathcal{V}|$ and computes

$$\vartheta_{\mathcal{F}}(\mathcal{G}) = \max_{\mathbf{M} \succeq \mathbf{0}} \sum_{ij} \sqrt{f_i f_j} \mathbf{M}_{ij} \text{ s.t. } \sum_i \mathbf{M}_{ii} = 1, \ \mathbf{M}_{ij} = 0 \, \forall (i,j) \in \mathcal{E}$$

via semidefinite programming. Setting all the weights to 1 gives the usual Lovász theta-function which was introduced earlier. Solving for $\vartheta_{\mathcal{F}}(\mathcal{G})$ on a NMRF with a perfect graph outputs the total weight of the MWSS. Let $\mathbf{M} \in \mathbb{R}^{n \times n}$ be the maximizer of $\vartheta_{\mathcal{F}}(\mathcal{G})$ and let ϑ be the recovered total weight of the MWSS. Under mild conditions, we can recover the solution $\mathcal{X}^*$ from the corresponding vector solution $\mathbf{x}^* = \text{round}(\vartheta \mathbf{M} \mathbf{1})$. In other words, we round the matrix multiplied by the all ones vector after scaling by the total weight. While slightly faster semidefinite programs have been proposed (Chan et al., 2009) that require $\tilde{\mathcal{O}}(n^5)$, even off-the-shelf interior-point solvers return $\mathcal{X}^*$ within $\mathcal{O}(n^6)$. In summary, this method is truly polynomial time for any perfect graph $\mathcal{G}$ even if the graph has an exponential number of cliques. Furthermore, this method is fully polynomial in the input size of the original graphical model G as long Algorithm 2 only computes MWSS and MSS sub-problems on perfect graphs. Thus, the semidefinite programming methods underlying the Lovász theta function avoid the clique enumeration problems that plague linear programming and message passing. In this sense, perfect graphs can still provide computational efficiency when the linear programming approaches (despite achieving integral solutions) are inefficient. This is a clear advantage of perfect graphs over traditional linear program integrality and total unimodularity approaches.

The next section enumerates several graphical models that compile into NMRFs that only require solving MWSS problems on perfect graphs. Therefore, the above solution methods (linear programming, message passing and semidefinite programming) are guaranteed to return the necessary MWSS efficiently.

1.6 Tractable graphical models

Certain families of graphical models are known to admit efficient MAP estimates. This leads to the natural question: do these graphical models readily compile into NMRFs with perfect graphs? For instance, junction trees and graphical models without cycles can be solved via efficient message passing techniques such as the max-product algorithm (Pearl, 1988; Wainwright and Jordan, 2008). Similarly, graphical models that solve matching and generalized matching problems admit exact inference in cubic time (or better) via the max-product algorithm (Bayati et al., 2005; Huang and Jebara, 2007; Sanghavi et al., 2008; Bayati et al., 2008). Graphical models with arbitrary topologies yet associative (submodular) potentials[9] also admit efficient algorithms via graph-cuts or min-cost network flow methods (Greig et al., 1989; Kolmogorov and Zabih, 2004). We next consider the NMRF representation of these specific MAP problems.

1.6.1 Acyclic graphical models

Tree-structured and acyclic graphical models are known to admit both efficient MAP estimation and efficient marginal inference. Recall that we are given as input a graphical model G with discrete random variables $y_1, \ldots, y_k$ with $|y_1|, \ldots, |y_k|$ possible configurations each. A simple acyclic graphical model is a tree with the following factorization

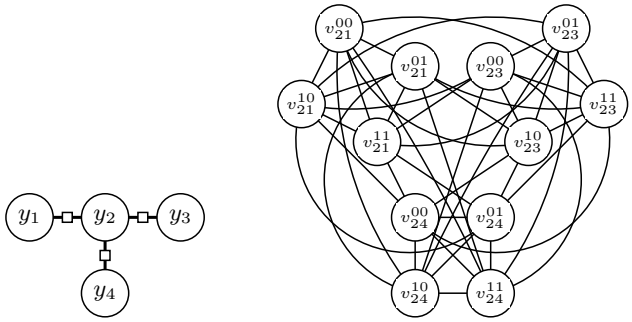
$$p(Y) = \frac{1}{Z} \prod_{i=2}^{k} \psi_i(y_i, pa(y_i))$$

where $pa(y_i)$ is a single node which is a parent of y_i in a directed acyclic graph (DAG) and where y_1 is the root of the tree and has no parent node in the DAG. Consider compiling this graphical model into a NMRF using Algorithm 1 as depicted in Figure 1.6.

Theorem 1.17 *A graphical model with a tree graph G produces a NMRF with a perfect graph $\mathcal{G}$.*

Proof First consider the simplest case where the input tree graphical model is merely a star. Consider a star graphical model G_a with an internal random variable y and leaf random variables $\{y_1, \ldots, y_p\}$ with $c = 1, \ldots, p$ factors over the pairs of variables $Y_c = \{y, y_c\}$. Compile

[9] In physics, models with associative potentials are also called ferromagnetic.



(a) Graphical model G (b) Equivalent NMRF $\mathcal{G}$

Figure 1.6 Compiling a tree structured graphical model G with binary random variables into its NMRF representation $\mathcal{G}$ (therein, subscripts denote the factor the NMRF node corresponds to and superscripts denote the configuration of its corresponding random variables).

the graphical model G_a into an NMRF $\mathcal{G}_a$ as follows. Initialize $\mathcal{G}_a = \{\}$ and introduce a NMRF node $v_{c,0+j}$ for each factor Y_c for each of the $j = 0, \ldots, |y| - 1$ configurations of y while fixing the setting $y_c = 0$. Join all these NMRF nodes pairwise if they correspond to different configurations of y. The resulting graph is a complete $|y|$-partite graph which is known to be perfect (Berge and Chvátal, 1984). To obtain $\mathcal{G}_a$ from the current complete $|y|$-partite graph, sequentially introduce additional nodes $v_{c,i|y|+j}$ for each Y_c for each of the $j = 0, \ldots, |y| - 1$ configurations of y as well as for each of the remaining $i = 1, \ldots, |y_c| - 1$ settings of y_c. Each sequentially introduced node is joined to the corresponding node $v_{c,0+j}$ that is already in $\mathcal{G}_a$ as well as all the neighbors of $v_{c,0+j}$. By the replication procedure in Lemma 1.8, this sequential introduction of additional nodes and edges maintains graph perfection. Once all nodes are added, the resulting graph is precisely the final graph $\mathcal{G}_a$ obtained by compiling a star G_a into NMRF form. Therefore, $\mathcal{G}_a$ is perfect by construction. Next, consider merging two such star-structured graphical models. The first star, G_a, contains the internal random variable y and leaf random variables $\{y_1, \ldots, y_p\}$. The second star, G_b, contains the internal random variable $\tilde{y}$ and leaf random variables $\{\tilde{y}_1, \ldots, \tilde{y}_q\}$. Consider merging these two stars by merging random variable y_1 into node $\tilde{y}$, merging random variable $\tilde{y}_1$ into node y and merging the factor ψ_{y,y_1} and factor $\psi_{\tilde{y},\tilde{y}_1}$ into a single factor $\psi_{y,\tilde{y}}$. The resulting union of

two star-shaped graphical models, denoted G_{a+b}, is no longer a star but rather a simple tree-shaped graphical model. The stars G_a and G_b separately give rise to NMRFs $\mathcal{G}_a$ and $\mathcal{G}_b$ which have already been shown to be perfect. The tree G_{a+b} gives rise to a NMRF denoted $\mathcal{G}_{a+b}$. The star shaped graphical models intersect over the factor $\psi_{y,\tilde{y}}$. It is clear, then, that the NMRFs $\mathcal{G}_a$ and $\mathcal{G}_b$ overlap only over the configurations of the factor $\psi_{y,\tilde{y}}$. Thus, $\mathcal{G}_a \cap \mathcal{G}_b$ form a clique cut-set. Furthermore, since $\mathcal{G}_a$ is perfect and $\mathcal{G}_b$ is perfect, Lemma 1.9 ensures that the graph $\mathcal{G}_{a+b}$ is perfect. By induction, incrementally merging star-shaped graphical models with the current tree G_{a+b} can be used to eventually create any arbitrary tree structure in the graph G. Since merging each star preserves the perfection of the compiled NMRF, the final NMRF $\mathcal{G}$ obtained from any tree-structured graphical model G is perfect. □

Theorem 1.17 can easily be generalized to handle the case where the input graphical model G is not a simple tree as illustrated above but, more generally, any *junction tree* as defined below (Wainwright and Jordan, 2008).

Definition 1.18 (Junction Tree) A graphical model G over the universe of random variables $Y = \{y_1, \ldots, y_k\}$ with factors $1, \ldots, l$ with corresponding potential functions $\psi_1, \ldots, \psi_l$ over subsets of random variables $Y_1, \ldots, Y_l$ is called a *junction tree* if and only if the factors can be made adjacent to each other as nodes in an acyclic graph such that, for any two factors i and j, all factors on the unique path joining i and j contain the variables in the intersection $Y_i \cap Y_j$.

Thus, graph perfection can be used to re-establish that MAP estimation in graphical models without loops is efficient (Pearl, 1988).

1.6.2 Associative graphical models

In the previous sub-section, the topology of a graphical model was restricted (to tree-structures) in order to guarantee that inference remains efficient for any set of potential functions $\psi_1, \ldots, \psi_l$. Instead of restricting topology while allowing arbitrary choices for potential functions, we may explore restrictions on the potential functions themselves and allow *arbitrary* topology. If all potential functions in the graphical model are associative (or, more generally, submodular), then MAP estimation is known to remain efficient (Greig et al., 1989; Kolmogorov and Zabih, 2004). Such problems frequently arise in computer vision and image processing. Can graph perfection be used to reproduce this efficiency result?

Consider a graphical model over $V = \{1, \ldots, k\}$ nodes with binary variables $Y = \{y_1, \ldots, y_k\}$ where $y_i \in \mathbb{B}$ (the extension beyond binary variables is possible yet outside the scope of this chapter) with pairwise potential functions between all pairs[10] of variables as implied by

$$p(Y) = \tfrac{1}{Z} \prod_{i \in V} \psi_i(y_i) \prod_{i \neq j} \psi_{ij}(y_i, y_j).$$

The singleton potential functions are specified via two arbitrary scalar values $\psi_i(0), \psi_i(1) \in \mathbb{R}^+$ for all $i \in V$. Meanwhile, the pairwise potential functions (i.e. factors) over a pair of binary variables are specified by four scalar values $\psi_{ij}(0,0), \psi_{ij}(0,1), \psi_{ij}(1,0), \psi_{ij}(1,1) \in \mathbb{R}^+$ for all $i \neq j$. We say that the pairwise potential functions are *associative* (or, more generally, submodular) if their values are restricted to obey the following relation

$$\psi_{ij}(0,0)\psi_{ij}(1,1) \geq \psi_{ij}(0,1)\psi_{ij}(1,0).$$

If such a property is satisfied for all pairwise potential functions, then the graphical model is associative. Consider compiling such a graphical model into NMRF form. However, before doing so, it will be helpful to rewrite the potential functions as follows

$$\psi_{ij}(y_i, y_j) = \hat{\psi}_{ij}(y_i, y_j)\phi_{ij}(y_i)\eta_{ij}(y_j)$$

where $\phi_{ij}(0) = \psi_{ij}(0,1)$, $\phi_{ij}(1) = \psi_{ij}(1,1)$, $\eta_{ij}(0) = \psi_{ij}(1,0)/\psi_{ij}(1,1)$ and $\eta_{ij}(1) = 1$. Next, consider the following modified pairwise potentials functions

$$\hat{\psi}_{ij}(y_i, y_j) = \begin{cases} \frac{\psi_{ij}(0,0)\psi_{ij}(1,1)}{\psi_{ij}(0,1)\psi_{ij}(1,0)} & \text{if } y_i = y_j = 0 \\ 1 & \text{otherwise.} \end{cases} \tag{1.6}$$

Rewrite the distribution as follows

$$p(Y) = \tfrac{1}{Z} \prod_{i \in V} \psi_i(y_i) \prod_{j \neq i} \hat{\psi}_{ij}(y_i, y_j)\phi_{ij}(y_i)\eta_{ij}(y_j).$$

Next, consider the following modified singleton potential functions

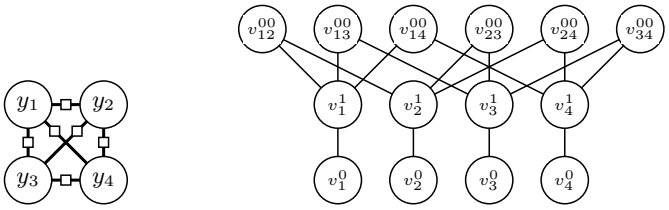
$$\hat{\psi}_i(y_i) = \psi_i(y_i) \prod_{j \neq i} \phi_{ij}(y_i)\eta_{ji}(y_i).$$

[10] To consider an associative graphical model that has sparse connectivity, one may simply assume that the pairwise potentials between some pairs of random variables are chosen to be constant.

These modified functions allow us to write the distribution for any associative graphical model as follows

$$p(Y) = \tfrac{1}{Z} \prod_{i \in V} \hat{\psi}_i(y_i) \prod_{i \neq j} \hat{\psi}_{ij}(y_i, y_j)$$

where the pairwise potential functions are restricted to have the form in Equation 1.6. Apply Algorithm 1 to each potential function. Then apply Algorithm 2 which obtains a graph $\mathcal{G}_1$ after removing nodes that correspond to $\hat{\psi}_{ij}(0,1)$, $\hat{\psi}_{ij}(1,0)$ and $\hat{\psi}_{ij}(1,1)$ since they all have zero weight (after the logarithm). This leaves a graph where there are two NMRF nodes for each singleton potential $\hat{\psi}_i$ for $i \in V$ and a single node for each pairwise potential $\hat{\psi}_{ij}$ for $i \neq j$. The resulting NMRF has a perfect graph $\mathcal{G}_1$ as shown in the following theorem and as depicted in Figure 1.7.



(a) Graphical model G (b) Equivalent NMRF $\mathcal{G}_1$

Figure 1.7 Compiling an associative graphical model G with binary random variables into its NMRF representation $\mathcal{G}_1$ (therein, subscripts denote the factor the NMRF node corresponds to and superscripts denote the configuration of its corresponding random variables).

Theorem 1.19 *A graphical model over binary variables with singleton and associative pairwise potential functions compiles into a MWSS problem on a perfect graph.*

Proof The NMRF graph $\mathcal{G}_1$ from Algorithm 2 contains edges between all disagreeing nodes. Therefore, each pair of NMRF nodes v_i^0 and v_i^1 corresponding to the singleton potentials $\hat{\psi}_i$ for $i \in V$ are pairwise adjacent. Each pairwise potential $\hat{\psi}_{ij}$ produces a single NMRF node v_{ij}^{00} which is only adjacent to v_i^1 and v_j^1. Since this set of edges forms a bipartite graph, $\mathcal{G}_1$ is perfect. □

Due to the perfection of the MWSS sub-problem, Algorithm 2 remains

efficient. In fact, the required MWSS sub-problem can easily be handled via linear programming which produces an integral solution. The subsequent MSS sub-problem in Algorithm 2 is trivial. Finally, given the solution of Algorithm 2, say $\mathcal{X}^*$, it is straightforward to deduce the MAP estimate of the random variables Y^*.

It may be the case that a perfect NMRF also emerges if higher-order potentials are used rather than merely associative (pairwise submodular) functions. These potentials give rise to so-called P^n Potts models which are widely used in computer vision (Kohli et al., 2009). It is sometimes possible that Algorithms 1 and 2 remain efficient for such graphical models and only require solving MWSS sub-problems on perfect graphs.

1.6.3 Matching graphical models

It was recently shown that certain graphical models known as matching (or generalized matching) graphical models also admit efficient MAP estimation (Bayati et al., 2005; Huang and Jebara, 2007; Sanghavi et al., 2008; Bayati et al., 2008). These graphical models are not associative *and* contain many cycles (i.e. are not trees). A matching problem arises, for instance, in a marriage problem, where q males and q females are to be paired together. There is a non-negative scalar score $f_{ij} \geq 0$ between each male i and each female j representing how compatible they are. Our goal is to find the matching where the total score is maximized for the whole population. This problem is known to admit efficient MAP estimation in $\mathcal{O}(q^3)$ via the classic Hungarian marriage algorithm. This maximum weight bipartite matching problem can be written as a graphical model (Bayati et al., 2005; Huang and Jebara, 2007) over a set of discrete variables $Y = \{y_1, \ldots, y_q, \tilde{y}_1, \ldots, \tilde{y}_q\}$ as follows:

$$p(Y) \;=\; \tfrac{1}{Z} \prod_{i=1}^{q} \prod_{j=1}^{q} \psi_{ij}(y_i, \tilde{y}_j) \tag{1.7}$$

where each potential function is defined as

$$\psi_{ij}(y_i, \tilde{y}_j) = \begin{cases} \varepsilon & y_i = j \text{ and } \tilde{y}_j \neq i \\ \varepsilon & y_i \neq j \text{ and } \tilde{y}_j = i \\ \exp(f_{ij}) & y_i = j \text{ and } \tilde{y}_j = i \\ 1 & \text{otherwise.} \end{cases}$$

Here, $1 >> \varepsilon > 0$ is an arbitrarily small positive quantity. Also, take $y_i \in \{1, \ldots, q\}$ to be the partner choice the i'th male makes (where $i = 1, \ldots, q$). Also, take $\tilde{y}_j \in \{1, \ldots, q\}$ to be the partner choice the

j'th female makes (where $j = 1, \ldots, q$). Each ψ_{ij} potential function captures the additional value gained from the marriage of a couple via the entry $\exp(f_{ij})$. The functions also make marriage choices reciprocal (an individual cannot chose someone who has chosen someone else) by returning ε in such situations (for a small enough ε, it is easy to see that reciprocity will be strictly enforced). Clearly, the potential functions are not associative (they are also not submodular). Furthermore, the graphical model has many cycles. Thus, the tractability of this problem falls outside the scope of the two previous sub-sections.

As in the previous section, we shall replace each ψ_{ij} function by factorizing it into a product of another modified pairwise potential $\hat{\psi}_{ij}$ and two singleton potentials ϕ_{ij} and η_{ij}. The intuition lies in finding a modified pairwise potential $\hat{\psi}_{ij} \geq 1$ with as many elements as possible equal to one (in other words, ψ_{ij} is log-sparse). Pseudo-code for finding such factorizations automatically is provided in Section 1.9. The result suggests the following factorization which is used to rewrite each potential function in $p(Y)$ as

$$\psi_{ij}(y_i, \tilde{y}_j) = \hat{\psi}_{ij}(y_i, \tilde{y}_j)\phi_{ij}(y_i)\eta_{ij}(\tilde{y}_j)$$

where

$$\hat{\psi}_{ij}(y_i, \tilde{y}_j) = \begin{cases} \exp(f_{ij})/\varepsilon^2 & y_i = j \text{ and } \tilde{y}_j = i \\ 1 & \text{otherwise,} \end{cases}$$

with

$$\phi_{ij}(y_i) = \begin{cases} \varepsilon & y_i = j \\ 1 & \text{otherwise,} \end{cases}$$

and

$$\eta_{ij}(\tilde{y}_j) = \begin{cases} \varepsilon & \tilde{y}_j = i \\ 1 & \text{otherwise.} \end{cases}$$

These potentials are collected to form the following equivalent graphical model involving both singleton and pairwise potential functions

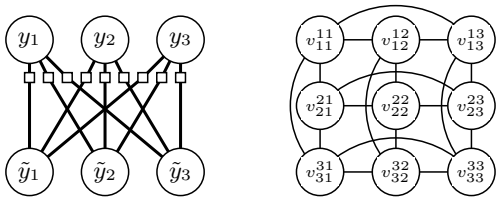
$$p(Y) = \frac{1}{Z}\left(\prod_{i=1}^{q}\prod_{j=1}^{q}\hat{\psi}_{ij}(y_i, \tilde{y}_j)\right)\prod_{i=1}^{q}\hat{\phi}_i(y_i)\prod_{j=1}^{q}\hat{\eta}_j(\tilde{y}_j).$$

where $\hat{\phi}_i(y_i) = \prod_{j=1}^{q}\phi_{ij}(y_i)$ and $\hat{\eta}_j(\tilde{y}_j) = \prod_{i=1}^{q}\eta_{ij}(\tilde{y}_j)$. Remarkably, these final singleton potentials are all constant. Therefore, they can be

absorbed into a new normalizer $\hat{Z}$ and the graphical model can be rewritten more succinctly as

$$p(Y) = \frac{1}{\hat{Z}} \prod_{i=1}^{q} \prod_{j=1}^{q} \hat{\psi}_{ij}(y_i, \tilde{y}_j).$$

Consider compiling the above graphical model into a NMRF graph $\mathcal{G}$ via Algorithm 1. For each of the q^2 potential functions $\hat{\psi}_{ij}$, $\mathcal{G}$ contains q^2 nodes. Denote the total set of q^4 NMRF nodes in $\mathcal{G}$ by v_{ij}^{kl} where $i, j, k, l \in \{1, \ldots, q\}$. Here, subscripts indicate which potential function the NMRF node comes from and superscripts identify the specific entry of that potential function. More precisely, a left subscript i and a right superscript l means that the i'th male chooses the l'th female. Similarly, a right subscript j and a left superscript k means that the j'th female chooses the k'th male. Whenever $k = i$ and $l = j$, the marriage choices are reciprocal and the nodes v_{ij}^{ij} have weight $f_{ij} - 2\log(\varepsilon)$. Denote by $S = \{v_{11}^{11}, \ldots, v_{1q}^{1q}, \ldots, v_{qq}^{qq}\}$ this subset of q^2 nodes with reciprocated marriages. The remaining nodes in $\mathcal{G}$ have weight zero. We then form $\mathcal{G}_1$ via Algorithm 2 by removing nodes with zero weight from $\mathcal{G}$. The graph $\mathcal{G}_1$ thus only contains the q^2 nodes in S and the following theorem shows that it is perfect.



(a) Graphical model G (b) Equivalent NMRF $\mathcal{G}_1$

Figure 1.8 Compiling a bipartite matching graphical model G with binary random variables into its NMRF representation $\mathcal{G}_1$.

Theorem 1.20 *The maximum weight bipartite matching problem compiles into a MWSS problem on a perfect graph.*

Proof Graph $\mathcal{G}_1$ obtained from the bipartite matching problem contains q^2 nodes v_{ij}^{ij} where $i, j \in \{1, \ldots, q\}$. Pairwise edges exist between v_{ij}^{ij} and v_{ik}^{ik} when $k \neq j$ and between v_{ij}^{ij} and v_{kj}^{kj} when $k \neq i$ since these correspond to incompatible settings of the random variables. It is easy

to see that $\mathcal{G}_1$ is a Rook's graph which is the line graph of a (complete) bipartite graph. Since these are basic Berge graphs, $\mathcal{G}_1$ is perfect. □

Since $\mathcal{G}_1$ is perfect, the MWSS sub-problem in Algorithm 2 remains efficient. Furthermore, the subsequent MSS sub-problem is trivial. Figure 1.8 depicts the induced subgraph $\mathcal{G}_1$.

Thus, Algorithm 2 finds the MAP estimate for graphical models representing bipartite matching problems in polynomial time. In fact, under mild assumptions, certain variants of message passing converge in just $\mathcal{O}(q^2)$ time when applied to the (dense) bipartite matching problem (Salez and Shah, 2009). Perfection can also be used to show when unipartite matching problems achieve integral linear programming relaxations (Sanghavi et al., 2008; Jebara, 2009). Informally, this is done by examining the line graph of the unipartite matching graphical model (much like the Rook's graph is the line graph of the bipartite graph in Figure 1.8(a)). If the line graph of the unipartite matching graphical model is a perfect graph, Algorithm 2 may potentially involve a MWSS problem over a perfect graph $\mathcal{G}_1$. Thus, the linear programming integrality of matching problems can be explained using graph perfection and the efficiency of MAP estimation for such graphical models is re-confirmed using the tools introduced in this chapter.

1.7 Discussion

Perfect graphs are a useful class of inputs that help outline when otherwise intractable problems admit efficient algorithms. This family of graphs is rich with theoretical and computational properties. Combinatorial problems such as graph coloring, maximum stable set and maximum clique can be efficiently solved when the input is a perfect graph. Off-the-shelf solvers such as linear programming and semidefinite programming are known to provide exact solutions. Similarly, certain canonical problems in machine learning and statistical inference can exploit perfect graph theory and algorithms. This chapter focused on the maximum a posteriori (MAP) estimation problem and showed how it can be compiled into a maximal maximum weight stable set problem on a nand Markov random field. MWSS solution methods such as linear programming, message passing and semidefinite programming can then be brought to bear and remain efficient if the input is a weighted perfect graph. Thus, the tractability benefits of perfect graphs extend into

the realm of MAP estimation which otherwise, in the worst case, can be intractable. Furthermore, in cases where MAP estimation was previously known to admit efficient algorithms, it was possible to compile the graphical models into NMRFs with perfect graph topologies that admit efficient MWSS solution. Thus, the perfect graph formalism helps reconfirm previously known efficiency results in graphical modeling. With further work, perfect graphs can potentially help the machine learning and statistics communities discover new families of graphical models that admit efficient MAP estimation.

1.8 Acknowledgments

The author thanks D. Bondatti, K. Choromanski, M. Chudnovsky, M. Danisch, D. Dueck and A. Weller for discussions. This material is based upon work supported by the National Science Foundation under Grant No. 1117631. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation.

1.9 Appendix

The following Matlab code rewrites a higher-order potential function (over a pair of variables or more) as a product of a log-sparse higher-order potential function (with all elements greater than 1) and singleton potentials. This is accomplished by linear programming to minimize an ℓ_1 norm on the output higher-order potential function. This helps reveal the representation of the graphical model that may lead to a MWSS problem on a perfect graph.

```
function [opsi,singletons] = factorizeClique(psi)
lpsi = log(psi./min(reshape(psi,prod(size(psi)),1)));
spsi = size(lpsi);
N = prod(spsi);
n = sum(spsi);
f = zeros(N+n+1,1);
f(1:(N+n))=1;
lb = -250*ones(N+n+1,1);
lb(1:(N+n))=0;
```

```
b = [reshape(lpsi,prod(spsi),1)];
A = zeros(N,N+n+1);
A(1:N,1:N)=eye(N,N);
for i=1:N
  q=ind2subT(spsi,i);
  for j=1:length(spsi)
      A(i,N+sum(spsi(1:(j-1)))+q(j))=1;
  end
end
A(:,end)=1;
ub = 250*ones(N+n,1);
X = linprog(f,[],[],A,b,lb,ub);
opsi = exp(reshape(X(1:N),spsi));
ind = N+1;
for i=1:length(spsi)
  singletons{i} = exp(X(ind:((ind+spsi(i))-1)));
  ind=ind+spsi(i);
end
siz = sz;
nout = length(siz);
siz = double(siz);
```

The above function also uses the following sub-function that simply returns multiple subscripts from a linear index.

```
function [outs] = ind2subT(sz,ndx)
if length(siz)<=nout,
  siz = [siz ones(1,nout-length(siz))];
else
  siz = [siz(1:nout-1) prod(siz(nout:end))];
end
n = length(siz);
k = [1 cumprod(siz(1:end-1))];
outs = zeros(1,nout);
for i = n:-1:1,
  vi = rem(ndx-1, k(i)) + 1;
  vj = (ndx - vi)/k(i) + 1;
  outs(i) = vj;
  ndx = vi;
end
```

References

Aloise, D., Deshpande, A., Hansen, P., and Popat, P. 2009. NP-hardness of Euclidean sum-of-squares clustering. *Machine Learning*, **75**, 245–248.

Bayati, M., Shah, D., and Sharma, M. 2005. Maximum weight matching via max-product belief propagation. In: *IEEE International Symposium on Information Theory*.

Bayati, M., Borgs, C., Chayes, J., and Zecchina, R. 2008. On the exactness of the cavity method for weighted b-matchings on arbitrary graphs and its relation to linear programs. *Journal of Statistical Mechanics: Theory and Experiment*, **2008**(06), L06001 (10pp).

Berge, C. 1963. *Six Papers on Graph Theory*. Calcutta: Indian Statistical Institute. Chap. Perfect graphs, pages 1–21.

Berge, C., and Chvátal, V. (eds). 1984. *Topics on perfect graphs*. North-Holland, Amsterdam.

Berge, C., and Ramírez-Alfonsín, J.L. 2001. *Perfect Graphs*. Wiley. Chap. Origins and genesis, pages 1–12.

Bron, C., and Kerbosch, J. 1973. Algorithm 457: finding all cliques of an undirected graph. *Communications of the ACM*, **16**(9), 575–577.

Chan, T-H. H., Chang, K.L., and Raman, R. 2009. An SDP primal-dual algorithm for approximating the Lovász-theta function. In: *IEEE International Symposium on Information Theory*.

Chudnovsky, M., Cornuéjols, G., Liu, X., Seymour, P., and Vusković, K. 2005. Recognizing Berge graphs. *Combinatorica*, **25**, 143–186.

Chudnovsky, M., Robertson, N., Seymour, P.D., and Thomas, R. 2006. The strong perfect graph theorem. *Ann. Math*, **164**, 51–229.

Chvátal, V. 1975. On certain polytopes associated with graphs. *Journal of Combinatorial Theory, Series B*, **13**, 138–154.

Foulds, J.R., Navaroli, N., Smyth, P., and Ihler, A. 2011. Revisiting MAP estimation, message passing, and perfect graphs. In: *Artificial Intelligence and Statistics*.

Globerson, A., and Jaakkola, T. 2007. Fixing max-product: Convergent message passing algorithms for MAP LP-relaxations. In: *Neural Information Processing Systems*.

Greig, D., Porteous, B., and Seheult, A. 1989. Exact maximum a posteriori estimation for binary images. *J. Royal Statistical Soc., Series B*, **51**(2), 271–279.

Grötschel, M., Lovász, L., and Schrijver, A. 1981. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, **1**, 169–197.

Grötschel, M., Lovász, L., and Schrijver, A. 1988. *Geometric Algorithms and Combinatorial Optimization*. Springer-Verlag. Chap. Stable sets in graphs.

Huang, B., and Jebara, T. 2007. Loopy belief propagation for bipartite maximum weight b-matching. In: *Artificial Intelligence and Statistics*.

Jebara, T. 2009. MAP estimation, message passing, and perfect graphs. In: *Uncertainty in Artificial Intelligence*.

Karp, R.M. 1972. *Complexity of Computer Computations*. New York: Plenum. Chap. Reducibility Among Combinatorial Problems, pages 85–103.

Knuth, D. 1994. The sandwhich theorem. *Electronic Journal of Combinatorics*, **1**.

Kohli, P., Ladicky, L., and Torr, P. 2009. Robust Higher Order Potentials for Enforcing Label Consistency. *International Journal of Computer Vision*, **82**(3), 302–324.

Kolmogorov, V.N., and Wainwright, M.J. 2005. On optimality of tree-reweighted max-product message-passing. In: *Uncertainty in Artificial Intelligence*.

Kolmogorov, V.N., and Zabih, R. 2004. What energy functions can be minimized via graph cuts? *IEEE Trans. Pattern Analysis and Machine Intelligence*, **26**(2), 147–159.

Komodakis, N., and Paragios, N. 2008. Beyond loose LP-relaxations: Optimizing MRFs by repairing cycles. Pages 806–820 of: *European Conference on Computer Vision*.

Lovász, L. 1972. Normal hypergraphs and the weak perfect graph conjecture. *Discrete Math.*, **2**, 253–267.

Lovász, L. 1979. On the Shannon capacity of a graph. *IEEE Transactions on Information Theory*, **25**, 1–7.

Pearl, J. 1988. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann.

Salez, J., and Shah, D. 2009. Optimality of belief propagation for random assignment problem. In: *Symposium on Discrete Algorithms*.

Sanghavi, S., Malioutov, D., and Willsky, A. 2008. Linear programming analysis of loopy belief propagation for weighted matching. In: *Neural Information Processing Systems*.

Shimony, S.E. 1994. Finding MAPs for belief networks is NP-hard. *Aritifical Intelligence*, **68**(2), 399–410.

Tomita, E., Tanaka, A., and Takahashi, H. 2006. The worst-case time complexity for generating all maximal cliques and computational experiments. *Theoretical Computer Science*, **363**(1), 28–42.

Wainwright, M.J., and Jordan, M.I. 2008. Graphical models, exponential families and variational inference. *Foundations and Trends in Machine Learning*, **1**(1-2), 1–305.

Weiss, Y., and Freeman, W.T. 2001. On the optimality of solutions of the max-product belief-propagation algorithm in arbitrary graphs. *IEEE Transactions on Information Theory*, **47**(2), 736–744.

Yedidia, J.S., Freeman, W.T., and Weiss, Y. 2001. Understanding Belief Propagation and Its Generalizations. In: *International Joint Conference on Artificial Intelligence, Distinguished Lecture Track*.

Yildirim, E.A., and Fan-Orzechowski, X. 2006. On extracting maximum stable sets in perfect graphs using Lovász's theta function. *Computational Optimization and Applications*, **33**(2-3), 229–247.