Approximate marginal inference in binary Markov random fields using a mean squared error energy approximation and the junction tree algorithm

Trygve Bertelsen Wiig

Master of Science in Physics and Mathematics
Submission date: May 2016
Supervisor: Håkon Tjelmeland, MATH

Norwegian University of Science and Technology
Department of Mathematical Sciences

Preface

This thesis is the culmination of the work on my Master's Thesis in Statistics (TMA4905) for my master's degree in Applied Physics and Mathematics, within the specialisation Industrial Mathematics, at the Norwegian University of Science and Technology (NTNU) and the Department of Mathematical Sciences (IMF).

First and foremost, I would like to thank my supervisor Håkon Tjelmeland for his exceptional guidance throughout the research and writing that led to this thesis, both during the past semester and during the work that led up to this thesis in the previous year. I would also like to thank my fellow students for their moral support and helpful comments. The past few years would have been far less enjoyable and interesting without you.

Trygve Bertelsen Wiig

Trondheim,
May 2016

Abstract

In this thesis, we use a mean squared error energy approximation for edge deletion in order to make performing inference in Markov random fields using the junction tree algorithm tractable, and by that develop an approximate marginal inference algorithm for binary Markov random fields. We apply the algorithm to a wide range of example models, including pairwise Ising models and Boltzmann machines, as well as two types of higher-order grid models. Three different approaches to selecting edges for deletion are developed and compared, and the quality of approximation of our algorithm is compared to that of the loopy belief propagation algorithm as well as to Gibbs sampling based inference, two traditional approaches to performing approximate marginal inference.

The results of the numerical experiments we have performed indicate that traditional approaches to approximate inference usually give superior results to those from our algorithm when working with pairwise models. However, the algorithm developed in this thesis shows promise when applied to higher-order models, and in cases where loopy belief propagation does not converge and stochastic methods are undesirable. We also find that choosing edges to delete using the Kullback-Leibler divergence as a criterion is particularly effective, but that other edge selection methods may be as effective for certain types of models.

Sammendrag

I denne oppgaven bruker vi en approksimasjon av energifunksjonen basert på middelkvadratfeil for å slette kanter i binære Markov-nettverk slik at det lar seg gjøre å utføre marginalinferens ved hjelp av *junction tree*-algoritmen. Vi anvender algoritmen på et variert utvalg eksempler, deriblant parvise Ising-modeller og Boltzmann-maskiner, samt to typer høyere ordens gridmodeller. Vi ser på og sammenligner tre ulike metoder for å velge kanter for sletting, og kvaliteten på approksimasjonene fra algoritmen vår sammenlignes med kvaliteten på approksimasjonene regnet ut ved hjelp av *loopy belief propagation* og approksimativ inferens basert på Gibbs-sampling, to klassiske metoder for å utføre approksimativ inferens.

Resultatene fra de numeriske eksperimentene vi har utført indikerer at de tradisjonelle metodene for å utføre approksimativ inferens vanligvis fører til resultater av høyere kvalitet enn vår algoritme når anvendt på parvise modeller. Imidlertid viser vår algoritme potensiale når den blir anvendt på høyere ordens modeller, samt i tilfeller der *loopy belief propagation* ikke konvergerer og stokastiske metoder ikke er ønskelige å bruke. Vi finner også at det er effektivt å velge kanter å slette basert på Kullback-Leibler-divergens, men at de andre foreslåtte metodene for valg av kanter å slette kan være like effektive for visse typer modeller.

Contents

Preface **i**

1 Introduction **1**

2 Background **3**
2.1 Graph theory . . . 3
2.1.1 Representing graphs . . . 5
2.1.2 Special graphs . . . 5
2.1.3 Junction trees . . . 6
2.2 Undirected graphical models . . . 8
2.2.1 Markov random fields . . . 9
2.2.2 Example of a Markov random field . . . 11
2.3 Divergences and information theory . . . 12
2.4 Pseudo-boolean functions . . . 14

3 Marginal inference **17**
3.1 Exact inference . . . 17
3.1.1 Belief propagation . . . 19
3.1.2 Junction tree algorithm . . . 22
3.2 Approximate inference . . . 27
3.2.1 Loopy belief propagation . . . 27
3.2.2 Sampling-based inference methods . . . 29

4 Approximating the junction tree algorithm **33**
4.1 Outline of the approximation algorithm . . . 33
4.2 Mean squared error energy approximation . . . 34
4.3 Edge selection . . . 36
4.4 Formalisation of the algorithm . . . 37
4.5 Time complexity . . . 40

5 Edge ranking methods **41**
5.1 Kullback-Leibler scoring . . . 41
5.2 Lowest potential . . . 42
5.3 Minimising fill-in . . . 43

6 Experimental setup **45**
6.1 Error function . . . 45
6.2 Example models . . . 46
6.2.1 204 and 500 edge Boltzmann machines . . . 46
6.2.2 Ising model . . . 46

6.2.3 Higher-order Markov random fields 47
6.3 Alternative inference algorithms 49
6.4 Implementation 50

7 Results 51
7.1 Ising models 51
7.2 Unbiased Ising models with low or high interaction parameters . . . 56
7.3 204 and 500 edge Boltzmann machines 58
7.4 Higher-order models 60
7.5 Discussion 68

8 Closing remarks 71
8.1 Future work 72

A Additional plots and data 73

Bibliography 93

1 Introduction

The purpose of this text is to introduce an approximate algorithm for marginal inference in binary Markov random fields (MRFs) based on the junction tree algorithm, by deleting edges using the mean squared error approximation that was developed by Austad and Tjelmeland (2015). Binary Markov random fields are a general and widely used class of graphical models, and being able to perform inference in such models is very useful. While there exist several exact inference algorithms for general Markov random fields, the usefulness of applying such algorithms to Markov random fields of some size is hampered by the fact that each algorithm in some way has exponential time complexity. With this in mind, a great number of approximate inference algorithms for Markov random fields have been developed. Two common approximate inference methods for general Markov random fields are *loopy belief propagation* (LBP) and sample-based inference using *Markov chain Monte Carlo* (MCMC) sampling methods, while even more efficient algorithms exist for specialized Markov random fields.

One common *exact* inference algorithm for Markov random fields is the junction tree algorithm. The junction tree algorithm can perform inference in arbitrary Markov random fields. However, the junction tree algorithm has exponential time complexity in the graph's tree width and is therefore not practical to use when working with large Markov random fields.

In this text, we will perform approximate inference by deleting edges from Markov random fields using the mean squared error energy approximation before running the junction tree algorithm on the reduced model. As mentioned, the performance of the junction tree algorithm is heavily dependent on the *width* of the junction tree of the model in question, and by removing edges, we may reduce the junction tree width and by that decrease the runtime of the junction tree algorithm. The challenge is then to choose the right set of edges to delete such that the junction tree width is reduced enough while maintaining a high quality of approximation.

There has been some previous work published on the topic of deleting edges in Markov random fields in order to simplify inference. Riedel, Smith, and McCallum (2010) studies the use of the Kullback-Leibler divergence as a criterion for choosing interactions to ignore, but does not study edge deletion in the context of a specific exact inference algorithm such as the junction tree algorithm. Most of the previous work has been done with a focus on Bayesian networks. Jensen and Anderson (1990) simplifies Bayesian networks by deleting weak interactions in each clique of the network's junction tree. Kjærulff (1994) continues this work by using the Kullback-Leibler divergence as a criterion for selecting interactions to delete.

Thornton (2003) builds on the work of both authors, and compares several strategies for simplifying the graph structure of Bayesian networks. The author proposes several methods based on edge deletion, including one that uses the junction tree algorithm to perform inference. Just as in Riedel, Smith, and McCallum (2010), a score based on the Kullback-Leibler divergence is used as a criterion for choosing edges, instead of exploiting the particular properties of the Junction tree algorithm. The main difference between the work of these authors and the topic of this thesis is our particular focus on binary Markov random fields and the use of the mean squared error energy approximation that was developed by Austad and Tjelmeland (2015).

The content of this thesis is divided into three main parts. In the first part (chapters 2 and 3), we will introduce the requisite knowledge needed for an understanding of Markov random fields, inference algorithms and the theory behind our own inference algorithm. In the second part (chapters 4 and 5), we will present our own inference algorithm based on the principle of edge deletion using the mean squared error energy approximation. We will discuss the algorithm's edge deletion strategy, and will propose three methods for selecting which edges to delete. The results of applying our approximation algorithm to a series of example models will be presented in the third part (chapters 6 and 7) of this text, and the results from our algorithm will be compared to those of other standard approximate algorithms, with a particular focus on quality of the approximate marginal probabilities output by the algorithms.

2 Background

The aim of this chapter is to present the concepts required to understand Markov random fields, inference algorithms for Markov random fields as well as our proposed approximate inference algorithm. Only the necessary topics will be covered – a complete introduction to Markov random fields can be found in such works as Koller and Friedman (2009).

2.1 Graph theory

In order to understand Markov random fields, it is necessary to have some background in graph theory. A *graph* is a set of arbitrary objects called *vertices*, which are connected by *edges*. These edges may be *directed*, in which case they point from one vertex to another, or they may be *undirected*, in which case they link two vertices without any directionality. In this text, we will mainly discuss undirected graphs, which are graphs that contain only undirected edges.

Definition 1. An *undirected graph* is an ordered pair $G = (V, E)$ of a set of vertices V and a set of edges $E \subseteq \{\{u, v\} : u, v \in V\}$.

Figure 2.1 shows an example of an undirected graph of 3 vertices labelled $1, 2$ and 3 and three edges connecting those vertices. In the graph in this figure, all edges connect two distinct vertices, but this need not be the case. We call an edge that connects a vertex to itself a *loop*. We often study only graphs that do not contain loops, and these are called *simple graphs*.

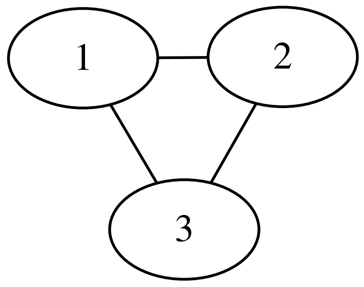


Figure 2.1: An undirected graph with 3 vertices and 3 edges.

Definition 2. A *simple undirected graph* is an undirected graph $G = (V, E)$ where $E \subseteq \{\{u, v\} : u, v \in V, u \neq v\}$.

Because we will not use any graphs containing loops, we will refer to simple undirected graphs simply as *graphs* throughout this text to avoid verbosity.

A given vertex may be connected to multiple other vertices. There are several terms in graph theory that are used to describe the *relationship* between a given vertex and the vertices it is connected to.

Definition 3. Let $G = (V, E)$ be a graph, and let $v, u \in V$ be two vertices in that graph. If there exists an edge $e \in E$ such that $e = \{u, v\}$, we say that u and v are *adjacent* or *neighbours*. The neighbourhood N_v of a vertex v is the set $N_v = \{u \in V : \{u, v\} \in E\}$ of vertices that are adjacent to v.

In other words, two vertices that are connected by an edge are adjacent to each other. Another important concept is that of a *clique*, which describes a mutually adjacent subset of the vertices in a graph.

Definition 4. Let $G = (V, E)$ be a graph, and let $\lambda \subseteq V$ such that $\{u, v\} \in E\} \ \forall u, v \in \lambda$. We then say that λ is a *clique*.

The set of just one vertex is a clique, as is the set of two vertices connected by an edge. As we can imagine, any graph of some size will contain a great number of cliques, not all of which are as important, and for this reason there exist several terms that describe special classes of cliques. We will look at two, the confusingly named *maximal* and *maximum* cliques.

Definition 5. Let Λ be the set of all cliques in a graph $G = (V, E)$, and let $\lambda \in \Lambda$ be an arbitrary clique. If there does not exist another clique $\lambda^* \in \Lambda$ such that $\lambda \subseteq \lambda^*$, then we say that λ is a *maximal clique*.

An equivalent definition to the one stated above is that a maximal clique is a clique that cannot be expanded by including an adjacent vertex, because none of the adjacent vertices are connected to every vertex already in the clique.

Definition 6. Let Λ be the set of all cliques in a graph $G = (V, E)$, and let $\lambda \in \Lambda$ be an arbitrary clique. If there does not exist a clique $\lambda^* \in \Lambda$ such that $|\lambda^*| > |\lambda|$, then we say that λ is a *maximum clique*.

In other words, a maximum clique is a clique that is at least as big as any other clique in the graph. The problem of finding a maximum clique in a general graph is difficult, and there are no known algorithms that can do this in polynomial time in the worst case (Pattabiraman et al., 2013).

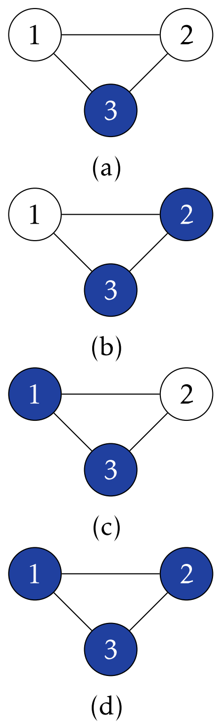


Figure 2.2: A visualization of all the cliques in the graph in Figure 2.1 that involve vertex 3. Only clique (d) is a maximal clique, and it is the sole maximal clique in the graph.

Figure 2.2 shows the cliques involving vertex 3 in the graph in Figure 2.1. This graph contains only one maximal clique, shown in Figure 2.2d. If we added a vertex 4 to the graph as well as an edge between edges 3 and 4, the graph would contain two maximal cliques – one containing vertices 1, 2 and 3, and one containing vertices 3 and 4.

2.1.1 Representing graphs

When doing computations using graphs, we need to represent the graph in terms of data structures that a computer can read and manipulate. A common representation of a graph in terms of a two-dimensional array is called an *adjacency matrix*. In an adjacency matrix, the cells with indices (i,j) and (j,i) are equal to 1 if the graph contains an edge between two vertices labelled i and j.

Definition 7. Let $G = (V,E)$ be a graph with a set of vertices V indexed by the integers $1,\dots,|V|$ and a set of edges E. We say that a square matrix A of size $|V| \times |V|$ is an *adjacency matrix* for G if $A(i,j) = A(j,i) = 1$ for all $i,j \in \{1,\dots,|V|\}, i \neq j$ such that $\{i,j\} \in E$, and the matrix A is zero otherwise.

Values of 1 on the diagonal of the adjacency matrix is often used to represent loops in the graph, but this is not relevant for simple undirected graphs. The adjacency matrix for the graph in Figure 2.1 is

$$A = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}.$$

2.1.2 Special graphs

A graph where every vertex is connected to every other vertex by an edge is called a *complete graph*.

Definition 8. Let $G = (V,E)$ be a graph with a set of vertices V and a set of edges E. G is called a *complete graph* if

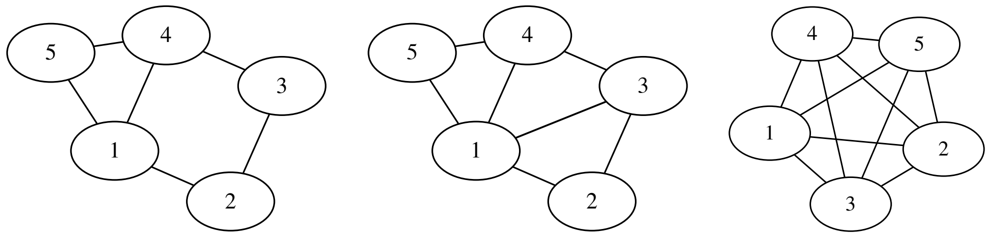
$$E = \{\{u,v\} : u,v \in V,\ u \neq v\}.$$

The complete graph containing n vertices is denoted by K_n.

We note that the graph in Figure 2.1 is the complete graph with 3 vertices, which we denote by K_3 according to the above definition. In light of having defined the notion of a complete graph, we can see that a clique is equivalently a complete subgraph: the graph that contains only the vertices in the clique and the edges connecting those vertices is complete.

There does not need to exist a route between any two vertices along the edges of a graph. When that is the case, we call the graph A *connected*. An *acyclic graph* is a graph in which there is exactly one path without repeated edges between any two vertices. *Trees* combine these two concepts.

Definition 9. A *tree* is a connected, acyclic graph. Alternatively, a tree is any connected graph $G = (V,E)$ with $|V| = n$ and $|E| = n - 1$.



(a) Non-chordal graph (b) One triangulation of the graph (c) Another possible triangulation

Figure 2.3: A non-chordal graph and two chordal graphs generated by adding edges to original graph in a). It is clear that the first triangulation is considerably less "complex" in the sense that it has three fewer edges than the second triangulation, which is a complete graph.

One notion that will become important in the next chapter is that of a *chordal* graph. A graph is chordal if every cycle of length greater than 3 has a chord, that is, an edge between two of the vertices in the cycle which is *not* part of the cycle in question. We may also use the following equivalent definition:

Definition 10. Let $G = (V, E)$ be a graph. G is chordal if for every cycle $C \subset V$, there exist at least $2|C| - 3$ edges $\{u, v\} \in E$ such that $u, v \in C$.

Constructing a chordal graph from some given graph by adding edges is called *triangulating* the graph. For every non-chordal graph, there are multiple ways to triangulate the graph, resulting in graphs of different complexity. Indeed, every chordal graph containing the original graph as a subgraph is a valid triangulation of that graph.

Definition 11. Let $G = (V, E)$ be a graph, and let $G^* = (V^*, E^*)$ be a chordal graph. If $V^* = V$ and $E \subseteq E^*$, then G^* is a *triangulation* of G.

Figure 2.3 shows two example triangulations of a non-chordal graph, generated from the original graph by two different triangulation methods. The additional edges added when triangulating a graph are known as *fill-in edges*. In this example, the first triangulation has just one fill-in edge while the second has four, which makes the first triangulation better for most purposes in which graph triangulations are used. As triangulation is an important part of the junction tree algorithm, this topic will be explored further in Section 3.1.2.

2.1.3 Junction trees

To describe the junction tree inference algorithm, we will later need the concept of a *junction* or *clique tree*. A junction tree groups the vertices of a graph into *supervertices*, that is, vertices representing several vertices in the original graph. Let $G = (V, E)$ be some arbitrary graph. A junction tree of G is a tree graph $J = (C, W)$, where each vertex $\lambda \in C$ represents a subset $\lambda \subseteq V$ of the vertices in the original graph such that

$$\bigcup_{\lambda \in C} \lambda = V.$$

In addition, a junction tree must fullfill several criteria:

i For every two vertices $u, v \in V$ that are connected by an edge $\{u, v\} \in E$, there exists a supervertex $\lambda \in C$ such that $\{u, v\} \subseteq \lambda$. In other words, if two vertices are connected in G there exists a vertex in J that contains both vertices.

ii For every vertex $u \in V$, there exists at least one supervertex $\lambda \in C$ such that $u \in \lambda$. In other words, every vertex in G is contained in at least one supervertex in J.

iii Let $\lambda_1, \lambda_2 \in C$ be two supervertices in the junction tree, let $\text{Path}_{\lambda_1}^{\lambda_2} = \text{Path}_{\lambda_2}^{\lambda_1} \subseteq C$ be the set of supervertices in J along the path between λ_1 and λ_2, and let $u \in V$ be a vertex in the original graph. If $u \in \lambda_1$ and $u \in \lambda_2$, then $u \in \lambda$ for all $\lambda \in \text{Path}_{\lambda_1}^{\lambda_2}$. In other words, if two supervertices in J contain the same vertex from G, then all vertices along the path between them contain that vertex as well.

Definition 12. Let $G = (V, E)$ be a graph with a set of vertices V and a set of edges E. Then the tree $J = (C, W)$ with a set of supervertices

$$C = \{\lambda \subseteq V\},$$
$$V = \bigcup_{\lambda \in C} \lambda$$

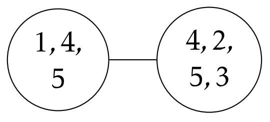
and a set of edges $\{\lambda_1, \lambda_2\}$, $\lambda_1, \lambda_2 \in C$ is a *junction tree* for G if and only if it satisfies criteria i, ii and iii above.

A junction tree is not unique – for any given graph, there are likely to exist a great number of junction trees. Usually, we want to find a junction tree with a low *width*, as the asymptotic time complexity of algorithms that use a graph's junction tree tends to grow exponentially with that quantity. Given some junction tree $J = (C, W)$, the width of that junction tree is defined as

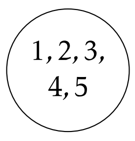
$$\text{Width}(T) = \max_{\lambda \in C} |\lambda|.$$

For example, the width of a junction tree consisting of one vertex containing 3 vertices of some graph G and another vertex containing 5 vertices of G, is 5. Figure 2.4 displays two examples of junction trees for the graph in Figure 2.3a. The first junction tree consists of two supervertices each containing four vertices, and the tree therefore has a width of 4. The second consists of only one supervertex containing all five vertices in the original graph, and therefore has a tree width of 5. In general, we would prefer working with the first junction tree due to its lower width.

The notion of the width of a given junction tree is distinct from that of a graph's *tree width*, which is the minimum width of all the graph's possible junction trees.



(a)



(b)

Figure 2.4: Two valid junction trees for the graph in Figure 2.3a. The numbers inside each supervertex indicate which vertices in the original graph that supervertex contains.

Definition 13 (Tree width)**.** Let $G = (V, E)$ be a graph, and let

$$\tau = \{J = (C, W) : J \text{ is a junction tree for } G\}$$

be the set of all possible junction trees for G. The *tree width* of G is then defined as

$$\text{TreeWidth}(G) = \min_{J \in \tau} \text{Width}(T). \tag{2.1}$$

Determining whether the tree width of G is at most some value t, the decision problem equivalent of calculating the tree width of the graph G, is an NP-complete problem (Arnborg, Corneil, and Proskurowski, 1987). Indeed, it trivially follows from this that the problem of finding an optimal junction tree with minimal width for a given graph G is non-tractable in general, as one otherwise would be able to simply calculate the tree width of G by calculating the width of this optimal junction tree. While the size of the largest clique in the original graph may be used as a minimum bound on the tree width, the tree width is often much larger.

We have not yet treated the question of how to actually generate a junction tree for an arbitrary graph. This question will be discussed further in Section 3.1.2, but for the time being we will mention that junction trees are typically generated by triangulating the graph into a chordal graph and then finding the maximal cliques in that graph. For this reason, the vertices in the junction tree, which we have thus far called *supervertices*, are usually called the *cliques* of the junction tree in the literature.

2.2 Undirected graphical models

The key idea in the field of *graphical probabilistic models* is to use a graph to define the *structure* of a probabilistic model, by associating each variable in the probabilistic model with a vertex in the graph, and using the edges to build up the dependence and independence relationships of the model.

Consider a graph $G = (V, E)$, and let Λ denote the set of all cliques in the graph. We now regard the set of vertices V as a set of random variables x. Our goal is to construct a joint probability distribution for the variables x based on the structure of the graph. One way to do this is to associate a function $\phi_\lambda(\lambda)$, traditionally called a *clique factor*, with each clique in the graph. If we also require that this function be non-negative, we can construct a joint probability distribution by taking the product of the clique factors. This gives us a distribution on the form

$$P(x) = \frac{1}{Z} \prod_{\lambda \in \Lambda} \phi_\lambda(x_\lambda),$$

where $\frac{1}{Z}$ is a normalisation constant which ensures that the distribution sums to 1. This structure is called a *Gibbs random field*, which we will now formally define.

Definition 14 (Gibbs random field)**.** Let $G = (V, E)$ be a graph with a set of vertices V and a set of edges E, let Λ be the set of all cliques in the graph, and let $x = \{x_v : v \in V\}$ be a set of random variables, each variable $x_v \in x$ defined on some sample space Ω. We say that x is a *Gibbs random field* on the graph G if the joint probability distribution $P(x)$ can be written on the form

$$\begin{aligned} P(x) &= \frac{1}{Z} \prod_{\lambda \in \Lambda} \phi_\lambda(x_\lambda), \\ Z &= \sum_{\omega \in \Omega^{|V|}} \prod_{\lambda \in \Lambda} \phi_\lambda(x_\lambda), \end{aligned} \tag{2.2}$$

in which case $P(x)$ is a *Gibbs distribution*.

2.2.1 Markov random fields

Markov chains have a *Markov property*

$$P(X_n \mid X_1 = x_1, \ldots, X_{n-1} = x_{n-1}) = P(X_n \mid X_{n-1} = x_{n-1}),$$

which specifies that the present state of the chain depend only on its previous state. This property only makes sense for a directed, one-dimensional model, such as a Markov chain. However, as this is a useful property which imposes a considerable amount of structure onto the model, it could be useful to develop a generalisation of the Markov property for undirected graphical models.

The key idea of this Markov property is that the state of a random variable, represented as a vertex in a directed graph, depends only on the state of its one parent. In a model on an arbitrary undirected graph, a given variable may have several parents, or neighbours, and this must be taken into account in our Markov property for undirected models.

Let x be a set of random variables, where each random variable $x_v \in x$ corresponds to a vertex $v \in V$ in some graph $G = (V, E)$. Each vertex in the graph is labelled with an integer in $1, \ldots, |V|$. For the sake of convenience, we will let a vertex v equal the integer it is labelled with in our calculations.

Just as in Markov chains, we want each variable $x_v \in x$ to depend only on the value of its neighbours in the graph. If we let x_{-j} denote the set of variables $\{x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_{|V|}\}$, this property can be expressed as

$$P(x_j \mid x_{-j}) = P(x_j \mid \{x_v : v \in N_j\}). \tag{2.3}$$

This is known as the *local Markov property*. An undirected graphical probabilistic model with this property is known as a *Markov random field* (MRF).

Definition 15 (Markov random field)**.** Let $G = (V, E)$ be a graph with a set of vertices V and a set of edges E, and let $x = \{x_v : v \in V\}$ be a set of random variables. We say that x is a *Markov random field* on the graph G if it satisfies the local Markov property given by (2.3).

We have now defined two different types of probabilistic models on undirected graphs, Gibbs random fields and Markov random fields. It would be useful if these two types of models were in fact equal, such that every Gibbs random field were also a Markov random field. Fortunately, given the additional condition on the model that $P(\omega) > 0$ for all ω in the sample space of x, this can be shown to be true. This result is known as the Hammersley-Clifford theorem, developed by the eponymous authors in an unpublished 1971 paper (Hammersley and Clifford, 1971).

Theorem 1 (Hammersley-Clifford theorem)**.** *Let $G = (V, E)$ be a graph with a set of vertices V and a set of edges E, and let $x = \{x_v : v \in V\}$ be a set of random variables, each defined on some sample space Ω, with a joint probability distribution $P(x)$. If the positivity condition $P(\omega) > 0$ is fulfilled for all $\omega \in \Omega^{|x|}$, then x is a Markov random field if and only if it is a Gibbs random field.*

In other words, given that the positivity condition is fulfilled, if a probabilistic model on an undirected graph satisfies (2.3), it can be written on the form given by (2.2) and vice-versa. Henceforth, a model of this kind will simply be referred to as a Markov random field.

To ensure that the joint probability distribution is positive, it is commonplace to write the clique factors on the form $\phi_\lambda(\lambda) = \mathrm{e}^{-V_\lambda(\lambda)}$. If this is done, the probability distribution of a Markov random field x may be written

$$\begin{aligned} P(x) &= \frac{1}{Z}\mathrm{e}^{-U(x)}, \\ U(x) &= \sum_{\lambda \in \Lambda} V_\lambda(x_\lambda), \\ Z &= \sum_{\omega \in \Omega^{|V|}} \mathrm{e}^{-U(\omega)}. \end{aligned} \tag{2.4}$$

The function $U(x)$ is often called the *energy function* of the Markov random field. This notation and terminology requires some explanation. The theory of Markov random fields has its roots in statistical mechanics, where the *Ising model* is used to study the behaviour of ferromagnetism.

In the Ising model, each random variable $x_i \in x$ models the *spin state* of an individual particle, and the function $U(x)$ is interpreted as the total energy of a configuration of spin states. The normalisation constant Z is conventionally called *the partition function*, a name which also has its origin in statistical mechanics. The individual terms $V_\lambda(x_\lambda)$ of the energy function are called *clique potentials*, *potential functions* or *interactions*.

Markov random fields may be factorised in multiple ways. For example, an MRF on a complete graph may be regarded as a product of many pairwise potential functions, or as one potential function on the clique that consists of the entire graph. Often, a particular factorisation is more easily *interpretable* than the others. However, the graph of a Markov random field contains no information about which factorisation is intended for the model.

Factor graphs are a way of amending this, by making the intended factorisation explicit in the graph structure. A factor graph is an undirected graph with two types of vertices: variables and factors. An edge exists between a factor and one or more variables if the joint probability distribution contains a factor, or potential function, for those variables.

Definition 16. Let x be a Markov random field defined on the graph $G^* = (V^*, E^*)$, and let Φ denote the set of factor functions of x such that

$$P(x) = \frac{1}{Z} \prod_{\phi \in \Phi} \phi(x).$$

Let $V = V^*$ be a set of variable vertices, let U be a set of factor vertices such that there is an $u_i \in U$ for every $\phi_i \in \Phi$, and let $E = \{\{v, u\} \colon v \in V, u \in U\}$ be a set of edges. Then the graph $G = (V, U, E)$ is a *factor graph* for the Markov random field x with factors Φ.

It must be noted that factor graphs are not unique. Because the factors Φ may be arbitrary functions, any given Markov random field will have several possible factor graphs that encode its joint probability distribution.

2.2.2 Example of a Markov random field

Let us now look at an example of a simple discrete, binary Markov random field. The Markov random field in question consists of a set of random variables

$$x = \{x_1, x_2, x_3\}$$

that has the probability distribution

$$P(x) = \frac{1}{Z} \mathrm{e}^{\alpha x_1 x_2 + \alpha x_2 x_3 + \alpha x_3 x_1 + \beta x_1 x_2 x_3}, \tag{2.5}$$

where Z is the partition function defined as in (2.4). Each variable $x_i \in x$ may take either 0 or 1 as its value, so the sample space of the probability distribution is $\{0, 1\}^3$. The energy function contains four interactions: three pairwise interactions between each pairs of the three variables, and one interaction between all three variables.

This Markov random field may be represented by the graph in Figure 2.1, where each vertex labelled by $i \in \{1, 2, 3\}$ corresponds to a variable x_i. However, note that there is in no way a one-to-one relationship between the probability distribution and the graph. Even if we deleted the interaction between x_1, x_2 and x_3, or even all the pairwise interactions, the Markov random field could still be represented by the same graph.

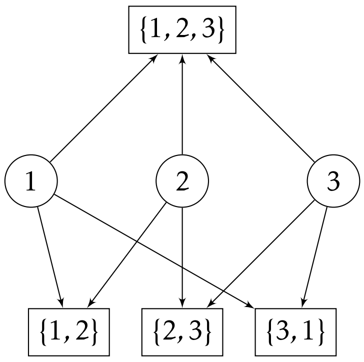


Figure 2.5: A factor graph for the Markov random field with the probability distribution given in (2.5).

A factor graph for our example Markov random field is presented in Figure 2.5. The circle vertices represent the nodes in the graph and the corresponding variables, while the rectangular vertices represent the interactions or factors. Inbound arrows from a set of variable vertices to a factor vertex means that the factor or interaction in question is a function of the variables in question.

The independence relationships of this Markov random field are very simple – as there exists an edge between each of the vertices, none of the variables are independent given the remaining vertex. If we simplify the model by removing any one of the edges, for instance the edge between the vertices 1 and 3, variables x_1 and x_3 will be independent *given* variable x_2. If we remove two edges, one of the variables will be fully independent of the other two variables in the model. If we remove all three edges, all the variables will be fully independent of each other.

2.3 Divergences and information theory

In real analysis, metrics are used to rigorously define the intuitive concept of distances. It would be useful to have an analogous notion of distance between probability distributions. The concept of *divergence* is one attempt at bringing this notion into probability theory. A divergence is a function $D(p\|q)$ from two probability distributions p and q to a positive real number $\mathbb{R}^+$. There is little common theory for divergence, as divergences, unlike metrics, need not be symmetric or satisfy the triangle inequality, and therefore have little structure in common. However, there exist several categories of divergences with common properties, and both these categories and the individual divergences are a topic of study. We will discuss a category of divergences known as *f-divergences* and two examples of f-divergences.

An f-divergence is any function of two probability distributions p, q—here we only consider the case where p and q are discrete—that can be expressed on the form

$$D(p\|q) = \sum_{\omega\in\Omega} f\left(\frac{p(\omega)}{q(\omega)}\right) q(\omega), \tag{2.6}$$

where Ω is some sample space and f is an arbitrary convex function with the property that $f(1) = 0$. The last property is necessary for $D(p\|p) = 0$ to hold. The most famous example of an f-divergence is the *Kullback-Leibler divergence*, often called just KL divergence. To understand the origin of the KL divergence, we need to introduce some tools from information theory.

The field of information theory, which studies *information* as a concrete quantity, was heavily inspired by physics and in particular the concept of thermodynamic entropy. In 1948, mathematician Claude Shannon introduced a concept of entropy of information as a way of measuring the amount of information contained in a random variable (Shannon, 1948).

Definition 17. Let x be a discrete random variable with some sample space Ω. The *entropy* of x is then

$$H(x) = -\sum_{\omega\in\Omega} P(\omega)\log P(\omega). \tag{2.7}$$

A full derivation of the entropy requires too much background to include here. However, it is worth noting the similarity between this expression and that for thermodynamic entropy. In a system with a set of microstates W, where each microstate $w \in W$ has a probability $P(w)$, the thermodynamic entropy of the system can be expressed as

$$S = -k_B \sum_{w\in W} P(w)\log P(w).$$

Up to the constant k_B, the two expressions are essentially identical. Indeed, if regarded through the lens of statistical physics, thermodynamic entropy is essentially an application of information theoretic entropy to physics, although we will not go further down that path.

It would be useful to be able to compare the entropy of two random variables. The concept of *relative entropy* is a way of doing this. Relative entropy is the difference between the entropy and the quantity

$$J(x) = -\sum_{\omega\in\Omega} P(\omega)\log Q(\omega),$$

which is the uncertainty about the outcome of some discrete random variable given that its probability distribution was falsely assumed to be Q. In the context of probability theory, relative entropy is known as *Kullback-Leibler divergence*.

Definition 18 (Kullback-Leibler divergence)**.** Let P and Q be two discrete probability distributions on some sample space Ω. The *Kullback-Leibler divergence* from P to Q is then

$$D(P\,\|\,Q) = \sum_{\omega\in\Omega} P(\omega)\log\frac{P(\omega)}{Q(\omega)}. \tag{2.8}$$

Comparing this expression with the general form of an f-divergence, we see that the Kullback-Leibler divergence is indeed in that category. The KL divergence is not symmetric, which is why we use the phrasing “from P to Q” when refering to the divergence of two probability distributions. It is sometimes symmetrized by instead using the expression $D(P\,\|\,Q) + D(Q\,\|\,P)$, but we will not do that in this text.

2.4 Pseudo-boolean functions

If we restrict ourselves to *binary* Markov random fields, we may analyse them in the context of the theory of *pseudo-Boolean functions*. A pseudo-Boolean function is a function from any set of two elements to the real numbers. In this text, we will specifically use the following definition.

Definition 19. A *pseudo-Boolean function* is a function $f : \{0,1\}^n \to \mathbb{R}$.

It is easy to see that the energy function of a binary Markov random field is a pseudo-Boolean function. Its arguments are the values of the variables in the Markov random field, and it returns a real number, the “energy”.

We will use the theory of pseudo-Boolean functions on the energy function in order to develop the mean squared error energy approximation that we will use in our approximate inference algorithm. For this reason, we will use the notation $U(x)$ for a pseudo-Boolean function throughout this section. Furthermore, our notation will stay close to that used by Austad and Tjelmeland (2015), in which the theory of the mean squared error energy approximation was developed.

Boros and Hammer (2002) shows that pseudo-Boolean functions may be written on the form

$$U(x) = \sum_{\Lambda\subseteq V} \beta_\Lambda \prod_{i\in\Lambda} x_i, \tag{2.9}$$

where V is a set $V = \{1,2,3,\ldots,n\}$ and each β_Λ is a real-valued constant. While any energy function for a binary Markov random field may be represented on this form, many or even most constants β_Λ may be zero for any given energy function. We would like to have a more compact representation of the energy function that does not require us to specify the constant β_Λ for every single subset of the variables.

We now define a set of "relevant" subsets of V

$$S = \{\lambda \subseteq V : \exists \lambda^* \supseteq \lambda, \beta_{\lambda^*} \neq 0\}.$$

In other words, S is the set of subsets of V which has at least one superset $\lambda^* \subseteq V$ for which β_{λ^*} is nonzero. Using S, we can write any pseudo-Boolean function on the form

$$U(x) = \sum_{\Lambda \in S} \beta_\Lambda \prod_{i \in \Lambda} x_i. \tag{2.10}$$

Let us look at the example of a Markov random field represented by the probability distribution

$$P(x) = \frac{1}{Z} \mathrm{e}^{-U(x)}, \tag{2.11}$$

$$U(x) = \beta_1 x_1 x_2 x_3 + \beta_2 x_1 x_2 x_4. \tag{2.12}$$

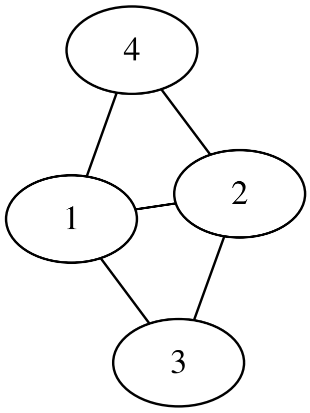


Figure 2.6: The graph of our example Markov random field from (2.12).

The graph of this Markov random field is displayed in Figure 2.6. In this example, we have two subsets $\Lambda \in V$ for which β_Λ is nonzero, namely $\{1,2,3\}$ and $\{1,2,4\}$. To represent the energy function on the same form as in (2.10), we define

$$S = \{\{1,2,3\},\{1,2,4\},\{1,2\},\{1,3\},\{1,4\},\{2,3\},\{2,4\},\{1\},\{2\},\{3\},\{4\},\varnothing\}.$$

Note that we may represent the relationships between the subsets $\Lambda \in S$ graphically, as in Figure 2.7.

In the context of Markov random fields, we call the subsets $\Lambda \in S$ *interactions*. Note that the relationships in Figure 2.7 dictate the structure of the Markov random field's graph. For an interaction to exist in the model, say $\{1,2,3\}$, all of its child interactions must be able to exist in the graph. To be able to have the interaction $\{1,2,3\}$, the graph must therefore have the edges $\{1,2\}$, $\{1,3\}$ and $\{2,3\}$, whether or not $\beta_{\{1,2\}}$, $\beta_{\{1,3\}}$ and $\beta_{\{2,3\}}$ are nonzero.

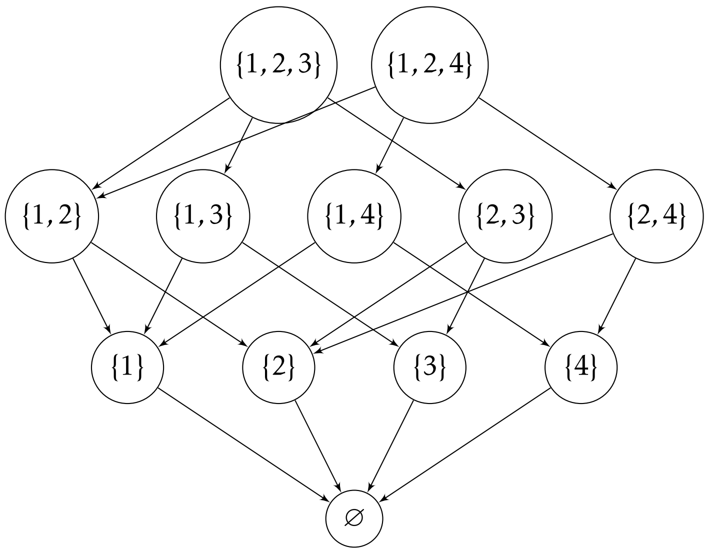


Figure 2.7: The relationships between the subsets $\Lambda \in S$. An arrow from a vertex to another vertex means that the latter is a subset of the former.

3 Marginal inference

The usual inference task in a Markov random field x is to compute the *marginal probability* of a subsetset of variables $x' \subset x$. In this text, we will mainly look at the particular case of computing the marginal probability distributions $P_{x_i}(x_i)$ for single variables $x_i \in x$.

3.1 Exact inference

Let x be a discrete Markov random field where each variable $x_i \in x$ takes a value s in the sample space $\Omega = \{0, \dots, k\}$ for some n, such that the sample space of x is $\Omega^{|x|} = \{0, \dots, k\}^{|x|}$. The marginal probability distribution for a variable x_i may then be written

$$P_{x_i}(x_i) = \sum_{s=0}^{n} \rho_s^{x_i} I(x_i = s),$$

where $I(P)$ is an *indicator function* which is equal to 1 whenever the predicate P is true, and equal to 0 whenever it is false. What we actually want to compute is the set of *marginal outcome probabilities*

$$\Phi(x_i) = \left\{ \rho_s^{x_i} : s \in \Omega \right\}.$$

Based on this, we define the notion of an *marginal inference algorithm*, or just inference algorithm in short, to be an algorithm which takes a discrete Markov random field x as input and computes the set of marginal outcome probabilities $\Phi(x_i)$ for some variable $x_i \in x$.

The simplest way to actually calculate $\Phi(x_i)$ and by that $P_{x_i}(x_i)$ is to explicitly sum over all possible values of the other variables $x^c = x \setminus x_i$ in the joint probability distribution $P(x)$. Let us write the probability distribution of our Markov random field x as in Equation (2.4), and let $P(a, B)$ denote the evaluation of the joint probability distribution at $x_i = a$ and $x^c = B$. Then the marginal probability distribution $P_{x_i}(x_i)$ may be written as

$$P_{x_i}(x_i) = \sum_{\omega \in \Omega^{|x|-1}} P(x_i, \omega),$$

where $\Omega^{|x|-1}$ denotes the set of all possible values of the variables $x_j \in x^c$. Using our alternative notation, the marginal outcome probability for a value s may consequently be written as

$$\rho_s^{x_i} = \sum_{\omega \in \Omega^{|x|-1}} P(s, \omega).$$

Let us again look at the example from Section 2.2.2. We wish to calculate the marginal outcome probabilities of the variable x_1. Each variable $x_i \in x$ has a sample space $\{0,1\}$. We start by calculating the value $\rho_0^{x_1}$, which written out becomes

$$\begin{aligned}
\rho_0^{x_1} &= P(0,0,0) + P(0,0,1) + P(0,1,0) + P(0,1,1) \\
&= \frac{1}{Z}\left(\mathrm{e}^0 + \mathrm{e}^0 + \mathrm{e}^0 + \mathrm{e}^\alpha\right) \\
&= \frac{1}{Z}(3 + \mathrm{e}^\alpha)
\end{aligned}$$

This calculation was simplified considerably due to the fact that the exponentiated terms in Equation (2.5) are simple products of x_1, x_2 and x_3, and so are usually zero. The calculation of $\rho_1^{x_1}$

$$\begin{aligned}
\rho_0(x_1) &= P(1,0,0) + P(1,0,1) + P(1,1,0) + P(1,1,1) \\
&= \frac{1}{Z}\left(\mathrm{e}^0 + \mathrm{e}^\alpha + \mathrm{e}^\alpha + \mathrm{e}^{3\alpha+\beta}\right) \\
&= \frac{1}{Z}\left(1 + 2\mathrm{e}^\alpha + \mathrm{e}^{3\alpha+\beta}\right)
\end{aligned}$$

is somewhat more complicated, but still doable. To get rid of the factors $\frac{1}{Z}$, we keep in mind that the sum of ρ_0 and ρ_1 must necessarily equal 1, and find that

$$Z = 4 + 3\mathrm{e}^\alpha + \mathrm{e}^{3\alpha+\beta}.$$

Inserting the example values $\alpha = 0.5$ and $\beta = 2$, we get the marginal outcome probabilities

$$\begin{aligned}
\rho_0^{x_1} &= 0.111, \\
\rho_1^{x_1} &= 0.889.
\end{aligned}$$

The simplicity of these calculations may be deceiving. Performing marginal inference for even a single variable rapidly becomes intractable as the size of the Markov random field grows and as the number of possible values increases. In a Markov random field of k variables and n possible values for each variable, calculating the marginal probability distribution of one variable requires summation over every possible value of the remaining $k-1$ variables, in total n^{k-1} summations. In realistic models, where we may have thousands of variables and hundreds of possible values, performing marginal inference by this method is infeasible for even the strongest supercomputers.

For this reason, several other exact inference algorithms have been developed that are usually considerably more efficient. We will discuss two, the belief propagation algorithm, an inference algorithm for Markov random fields defined on trees, and the junction tree algorithm, which is related to belief propagation but is able to perform exact inference in any Markov random field.

3.1.1 Belief propagation

Belief propagation is an inference algorithm for Markov random fields that are defined on trees. In its essence, belief propagation is a scheme to perform exact inference of marginal probabilities more efficiently by exploiting the structure of trees to avoid performing the same computations multiple times.

Seen from an algorithmic point of view, the belief propagation algorithm takes place in the context of the factor graph of a Markov random field on a tree. In belief propagation, *messages* are sent between the vertices of the factor graph, according to a set of rules known as the *message passing protocol*. In the belief propagation algorithm, the marginal probability is also known as the *belief*. The belief at a node is the normalised product of the messages received from its neighbouring nodes – a variable's belief about its own value is the aggregate of what it is being told by its neighbouring nodes, that is, the messages.

Let x be some Markov random field, and let $F = (V, U, E)$ be the factor graph for the Markov random field with a set of variable vertices V and a set of interaction vertices U. We will denote the interaction vertices by $i, j, k, \ldots$ and the variable vertices by $u, v, w, \ldots$. For the potential function corresponding to an interaction vertex $i \in U$ we will use the the notation ϕ_i. Furthermore, by x_i for any $i \in U$ we will mean a vector of the variables involved in the potential function ϕ_i, which number at most two in a Markov random field defined on a tree graph. Let us write the joint distribution of our Markov random field x on the form

$$P(x) = \frac{1}{Z} \prod_{\phi \in \Phi} \phi(x) \tag{3.1}$$

where Φ is the set of factors of x. Mathematically, belief propagation is essentially a way of iteratively transforming this joint probability distribution into a different form which makes it easy to calculate the marginal probabilities of the individual variables in x. Letting $u \sim i$ denote a pair of adjacent vertices in the factor graph, we may write this same distribution using messages as

$$P(x) = \frac{\prod_{i \in U} P_{x_i}(x_i)}{\prod_{u \sim i} m_{u \to i}(x_i) m_{i \to u}(x_u)}, \tag{3.2}$$

where $m_{u \to i}$ denotes a message from a variable node u to a interaction node i, and $m_{i \to u}$ denotes a message from an interaction node i to a variable node u. (Koller and Friedman, 2009, pp. 361-362). With the joint probability function written on this form, a marginal probability $P_{x_a}(x_a)$, $a \in V \cup U$, can be calculated as the expression

$$P_{x_a}(x_a) = \frac{1}{Z} \phi_a(x_a) \prod_{b \in N(a)} m_{b \to a}(x_a), \tag{3.3}$$

where $N(a)$ denotes the set of neighbours of the vertex a in the graph. As usual, $\frac{1}{Z}$ is a normalisation constant, while ϕ_a is the potential function corresponding to x_a. As we treat the potentials $\phi_u(x_u)$ involving only one variable independently from the variables themselves, we assign the potential 1 to every variable vertex, while each interaction x_i has the potential ϕ_i. This gives us the separate expressions

$$\begin{aligned} P_{x_u}(x_u) &= \frac{1}{Z} \prod_{i \in N(u)} m_{i\to u}(x_u), \\ P_{x_i}(x_i) &= \frac{1}{Z} \phi_i(x_i) \prod_{u \in N(i)} m_{u \to i}(x_i) \end{aligned} \tag{3.4}$$

for the marginal probabilities of variable and interaction vertices, respectively. Next, we note that if we know the marginal probability $P(x_i)$ of some interaction, we can calculate the marginal probability of any variable u taking part involved in the interaction using the expression

$$P_{x_u}(x_u) = \sum_{\omega \in \Omega_{x_i \setminus x_u}} P(\omega; x_u), \tag{3.5}$$

where $\Omega_{x_i \setminus x_u}$ denotes the sample space of x_i without x_u. In other words, we sum over the possible values of the variables involved in the interaction, but fix the value of the variable we are interested in calculating the marginal probability of. This is similar to what we did when naively calculating the marginal probabilities in Section 3.1.

From (3.4) and (3.5), we may derive expressions for the messages $m_{i\to u}$, from an interaction vertex to a variable vertex, and $m_{u\to}$, from a variable vertex to an interaction vertex. We will not show the full derivation here, but the interested reader may see e.g. Khosla (2009). The result is the update formulas

$$\begin{aligned} m_{u\to i}(x_i) &= \prod_{j \in N_{-i}(u)} m_{j \to u}(x_u), \\ m_{i \to u}(x_u) &= \sum_{x_N \in N_{-u}(i)} \phi_i(x_N) \prod_{v \in N_{-u}(i)} m_{v \to i}(x_i), \end{aligned} \tag{3.6}$$

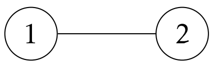
where $N_{-u}(i)$ denotes the set of neighbours of vertex i, except for a vertex u. In the second expression, we sum over the possible values of the variables in the neighbourhood.

The order in which the messages are computed is known as the *schedule*. When the belief propagation algorithm is used on a Markov random field defined on a tree graph, we may use a simple version of the algorithm in which messages are simply computed whenever they are "ready" to be computed. In this context, we say that a message $m_{a\to b}$ is *ready to be computed* when vertex a has received messages from all its neighbours except for a vertex b. Once every message has been computed, we calculate the marginal probabilities for a variable using (3.4).

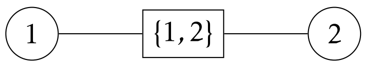
This simple version of the belief propagation algorithm is presented in Figure 3.1. While we do not explicitely specify a schedule, the message computation will

```
Algorithm: Belief propagation
  Data: factor graph of Markov random field $x$ with potentials $\phi_i$
  Result: marginal probabilities $P_{x_i}(\omega)\ \forall x_i \in x, \omega \in \Omega_{x_i}$
  while a message $m_{a \to b}$ is ready to be computed do
      compute $m_{a \to b}$ according to (3.6);
  end
  for each variable $x_i \in x$ do
      for $\omega \in \Omega_{x_i}$ do
          compute $P_{x_i}[\omega]\ \forall x_i \in x$ according to (3.4);
      end
  end
```

Figure 3.1: Pseudocode for the belief propagation algorithm on a tree Markov random field.



(a) Graph of the example Markov random field



(b) Factor graph of the example Markov random field

Figure 3.2: The graph and factor graph of the example Markov random field used to demonstrate belief propagation.

in our simple version of the algorithm always begin at the leaf vertices – the vertices that have only one neighbour – as these do not need to receive any messages in order to be able to send a message.

Consider for instance the example Markov random field in Figure 3.2, with two variable vertices and one interaction vertex representing a pairwise interaction. Here we may for example start with calculating the message from the variable vertex labelled 1 to the interaction vertex, but we might as well have started with calculating the message from variable vertex 2 to the interaction vertex. Once the messages $m_{1\to\{1,2\}}$, $m_{2\to\{1,2\}}$, $m_{\{1,2\}\to 1}$ and $m_{\{1,2\}\to 2}$ have been computed, the message passing is completed and we may calculate the marginal probabilities using (3.4).

Figure 3.3 shows a more example of the order of message passing, with the arrows and numbers indicate the direction and order of each message. The schedule shown in this figure is only one of several possible schedules for this graph – for instance, any one of the leaf vertices could have been chosen as the first vertex to send a message.

While the belief propagation algorithm is only exact when applied to Markov random fields defined on trees, it can often successfully be used with arbitrary Markov random fields. When this is done, the result is often high-quality approximations to the exact marginals, but the guarantee that the messages converge is lost. Belief propagation is often known as *loopy* belief propagation in this context, and

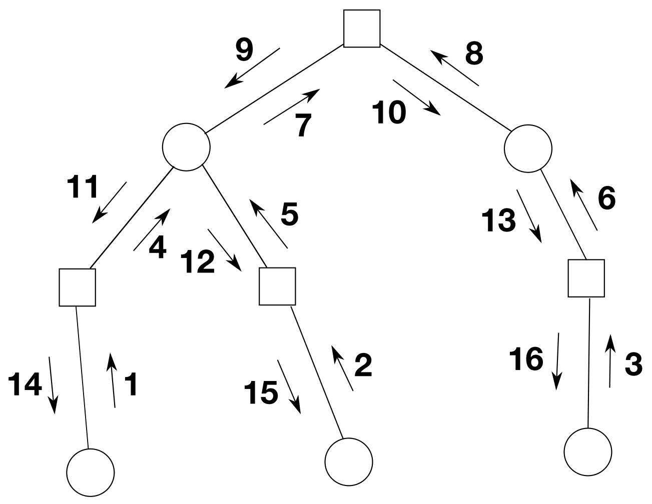


Figure 3.3: A run of the belief propagation algorithm on an example factor graph. As usual, the squares are interaction vertices while the circles are variable vertices. The arrows indicate messages, and the number next to each arrow indicates when that message is sent, with the message labelled 1 being sent first.

will be studied later.

3.1.2 Junction tree algorithm

As mentioned in the previous section, the belief propagation algorithm is only guaranteed to be exact for Markov random fields defined on trees. On general graphs, it is not even guaranteed to converge. However, there exists another exact inference algorithm that, by modifying the graph before running belief propagation, is able to perform inference on general graphs. This algorithm is called the *junction tree algorithm*.

The junction tree algorithm as applied to a Markov random field consists of three main steps:

- Triangulate the graph
- Build a junction tree from the triangulated graph by finding the maximal cliques
- Pass messages throughout the junction tree

We will now look at each of these steps in detail.

In the previous chapter, we discussed chordal graphs and mentioned that the process of transforming a graph into a chordal graph is known as *triangulation*. Triangulation is done by adding edges to the cycles preventing the graph from being chordal. The triangulation of a given graph is not unique. For most graphs, there

```
Algorithm: Graph triangulation
  Data: adjacency matrix $adj$ of a graph $G = (V, E)$
  Result: adjacency matrix $adj_{new}$ of the triangulated graph $G' = (V', E')$
  $adj_{temp} \leftarrow adj$;
  $adj_{new} \leftarrow adj$;
  for each vertex $u$ in the graph do
      $N \leftarrow$ set of neighbours of $u$ in $adj_{temp}$;
      for each $v \in N$ do
          for each $w \in N$ do
              if $v \neq w$ then
                  $adj_{new}(v, w) \leftarrow 1$;
                  $adj_{new}(w, v) \leftarrow 1$;
              end
          end
      end
      $adj_{temp}(u, :) \leftarrow 0$;
      $adj_{temp}(:, u) \leftarrow 0$;
  end
```

Figure 3.4: Description of the standard triangulation algorithm used in this text using an adjacency matrix to represent the graph. This algorithm is also known as the *vertex elimination algorithm*.

is a large number of possible triangulations, which result from different choices of edges to add in the triangulation process. Notably, different triangulations may result in cliques with different tree widths.

As we will see, the junction tree algorithm has exponential time complexity in the tree width of the triangulated graph, which means that the performance of the junction tree algorithm may vary widely depending on which triangulation is chosen. However, calculating the optimal triangulation is an NP-hard problem. We will not concern ourselves with this issue, and will use a standard algorithm for the triangulation process similar to the one given by Cano and Moral (1995). Figure 3.4 displays pseudocode for the this standard triangulation algorithm.

The triangulation algorithm goes through each vertex in the original graph in some arbitrary order. The algorithm starts by creating the output graph, which at the beginning is equal to the input graph. For each vertex, the algorithm then adds an edge between every neighbour of the chosen vertex in this new graph. The vertex is then removed from the original graph. This process is continued until every vertex has been removed from the original graph. The new graph that was created by adding edges is then a chordal graph with the original graph as its subgraph.

The choice in this algorithm which impacts the number of fill-in edges created

```
Algorithm: Junction tree construction algorithm
    Data: chordal graph $G = (V, E)$
    Result: junction tree $J = (C, W)$
    $C \leftarrow$ MaximalCliques($G$);
    $\gamma_{ij} \leftarrow$ intersection between cliques $i, j \in C$;
    $w_{ij} \leftarrow |\gamma ij|$;
    $J \leftarrow$ MaximumSpanningTree($C, \gamma, w$);
```

Figure 3.5: Constructing a junction tree from a chordal graph

and the tree width of the resulting chordal graph is the order in which vertices are deleted from the original graph – the *elimination ordering*. Choosing the order which results in the lowest fill-in and the lowest tree-width are known as the *minimum fill-in* and *minimum tree width* problems, respectively. These problems are not solvable in polynomial time. As previously mentioned, we will not bother ourselves with the issue of choosing a triangulation method, and will take the elimination order in the above algorithm as arbitrary. For the interested, however, there exist numerous methods of choosing an elimination ordering that usually give better results than choosing the order by random (Heggernes, 2006).

Once we have a chordal graph, we may build our junction tree. While we have already *defined* the notion of a junction tree, we have not looked at how to actually construct a junction tree from an arbitrary graph. It turns out that the first step is to triangulate the graph, as we have already done. Once the graph is chordal, we obtain the junction tree by finding the *maximum spanning tree* of another graph that is made up of the chordal graph's maximal cliques (Koller and Friedman, 2009, pp. 374–376). The pseudocode for building a junction tree from a chordal graph is displayed in Figure 3.5.

The general idea of this algorithm is to find a list of maximal cliques in the graph, and building a new graph consisting of the maximal cliques and edges between the cliques that have an intersection. We then find the maximum spanning tree of this new graph using the number of variables in each intersection as the edge weights. The resulting tree is the junction tree of the original graph (Koller and Friedman, 2009, pp. 374–375).

There are two loose ends in this pseudocode that we will need to elaborate on. First, how do we find the maximal cliques in the chordal graph? It turns out that the problem of listing all the maximal cliques in a chordal graph can be done in polynomial time using a method known as maximum cardinality search (Koller and Friedman, 2009, pp. 374). Maximum cardinality search generates an ranking of the vertices in the graph such that each vertex is ranked according to how many of its neighbours are before it in the ranking. After generating such a ranking, known as a *perfect elimination ordering*, we can find all the maximal cliques by, for each vertex $v \in V$, creating a clique consisting of v as well as every vertex $u \in V$ that succeeds v in the perfect elimination ordering. If we then eliminate every clique that is a subset of any other clique, we are left with the maximal cliques.

Second, how do we find the maximum spanning tree? To do this, we can use any standard algorithm for finding the *minimum* spanning tree, such as Prim's (PRIM, 1957) or Kruskal's (KRUSKAL, 1956) algorithms, and simply use negative weights on the edges. Both algorithms complete in polynomial time. In our case, we run either of the two mentioned algorithms on a graph in which the maximum cliques found in the previous step are the vertices and there is an edge between any two cliques that have at least one vertex from the original graph in common. The weight of an edge is the *number of vertices from the original graph* that the two maximal cliques it connects have in common. Neither pseudo-code for the minimum spanning tree algorithms nor for maximum cardinality search will be given here – the interested may refer to KOLLER and FRIEDMAN (2009, pp. 312, 1147).

Having constructed a junction tree, we may pass messages throughout the tree in order to perform inference, in a similar way to that described when we discussed belief propagation in the previous section. In fact, there are several ways to do this, and the choice of method often depends on the inference task in question. The method we will use in this exposition, is often known as the *Shafer-Shenoy algorithm* (T. S. LEVITT and SHACHTER, 2014). As this is similar to the method used when we discussed belief propagation, we refer readers to that section for additional detail.

As usual, the inference task is to compute the marginal probability of every individual variable in a Markov random field. The first step is to calculate an initial potential for every clique in the junction tree, which is the product of all the potentials in the original Markov random field x that belong to that clique. Let Φ denote the set of potential functions in x, let $N(\phi)$ denote the set of variables associated with a particular potential function $\phi \in \Phi$, and let Φ_λ denote the set of potential functions that are associated with the clique $\lambda \in C$ such that

$$\Phi_\lambda = \Big\{\phi(x_\phi) \in \Phi : x_i \in \Lambda \ \forall x_i \in N(\phi)\Big\}. \tag{3.7}$$

For every clique $\lambda \in C$, we then define an initial potential

$$\Phi_\lambda = \prod_{\phi \in \phi_\lambda} \phi(x_\phi). \tag{3.8}$$

We can now start passing messages between the cliques in the junction tree. The message passing procedure is similar to that of the belief propagation algorithm described in the previous chapter, but with some changes as we are no longer working with a factor graph of variable and interaction vertices.

Let $\lambda_i \in C$ be a clique in the junction tree, with a set of adjacent cliques $N_{\lambda_i} \subseteq C$. In order for this clique to send a message to one of its adjacent cliques $\lambda_j \in N_\lambda$, it has to already have received messages from every of its other neighbours. Once this condition has been fulfilled, it calculates the product of its own potential and all the messages it has received. Finally, all variables except for those in the intersection

```
Algorithm: Junction tree algorithm
  Data: potentials $\phi_i$, graph $G = (V, E)$
  Result: marginal probabilities $P_{x_i}(\omega)\ \forall x_i \in x, \omega \in \Omega_{x_i}$
  $G' = (V', E') \leftarrow$ Triangulate($G$);
  $J = (C, W) \leftarrow$ ConstructJunctionTree($G'$);
  for each clique $\lambda \in C$ do
    $\Phi_\lambda \leftarrow$ (3.8);
  end
  while there exists at least one clique $\lambda_i$ that may send a message to a clique $\lambda_j$ do
    $m_{\lambda_i \to \lambda_j} \leftarrow$ (3.9);
  end
  for each variable $x_i \in x$ do
    for $\omega \in \Omega_{x_i}$ do
      compute $P_{x_i}(\omega)$ according to (3.10);
    end
  end
```

Figure 3.6: Computation of marginal probabilities for every variable in a Markov random field using the junction tree algorithm.

$w_{ij} \in W$ are summed out. If $V_\lambda \subseteq V$ is the set of variables in some clique λ, the message to be sent to λ_j will be

$$m_{\lambda_i \to \lambda_j}(\lambda_j) = \sum_{\lambda_i \setminus w_{ij}} \left[\Phi_{\lambda_i}(x_{\lambda_i}) \prod_{\lambda \in N(\lambda_i) \setminus \lambda_j} m_{\lambda \to \lambda_i}(\lambda_i) \right]. \tag{3.9}$$

The junction tree algorithm is executed by continuously looking for possible messages that can be sent in the graph, and sending every message that can be sent until there are none remaining, similarly to how we performed the belief propagation algorithm. When this is done, we can compute the marginal probability of a variable x_i in a clique λ_i by the expression

$$P_{x_i}(x_i) = \sum_{\lambda_i \setminus x_i} \left[\Phi_{\lambda_i}(x_{\lambda_i}) \prod_{\lambda \in N(\lambda_i)} m_{\lambda \to \lambda_i}(\lambda_i) \right], \tag{3.10}$$

where we multiply the potential of the variable's clique with all the incoming messages to that clique, and sum out the other variables. Figure 3.6 displays a summary of the junction tree algorithm in pseudocode.

While more efficient than the naive variable elimination procedure, the junction tree algorithm has exponential time complexity in the tree width (Sinoquet and Mourad, 2014) due to the summations over the variables in each clique that need to be done when calculating messages. While this allows us to perform inference on graphs of arbitrary size as long as they are loosely connected, such that the tree width remains low, inference in models defined on *general* graphs where the tree width may be arbitrarily large remains intractable.

An important example of a class of model where the tree width increases without bound as a function of the total number of variables in the graph is models defined on a grid, but there are other examples of pathological models where the junction tree algorithm is highly ineffecient even for a low total number of variables. For instance, any complete graph with k nodes has a tree width of k.

3.2 Approximate inference

Several exact inference algorithms exist, but all have superpolynomial time complexity, making them unwieldy for all but the smallest Markov random fields. In the following chapter, we will introduce our own approximate inference algorithm based on the junction tree algorithm. However, in our numerical experiments, we will need to compare the performance of our algorithm to that of other approximate inference algorithms. Two of the most widely used are *Markov chain Monte Carlo* and *loopy belief propagation*, which will be described in this section.

3.2.1 Loopy belief propagation

As it happens, although it only is guaranteed to produce exact results when applied to trees, the belief propagation algorithm can be run on any Markov random field, and will often converge to some set of probabilities. While these probabilities may not necessarily be correct, this procedure nevertheless often produces useful results. Belief propagation as applied to Markov random fields defined on general graphs is typically called *loopy* belief propagation.[1]

Let $P(x)$ be an arbitrary Markov random field on some graph $G = (V, E)$ that is not a tree, and let ϕ be the set of factors of the Markov random field. Again, we regard a factor graph with the variables and interactions as vertices. The belief propagation algorithm as we have defined it requires that a interaction vertex, before sending a message to some variable vertex, must first receive messages from all adjacent variable vertices. In a graph containing cycles, this will not always be possible to do, as there will be sets of vertices where each vertex requires that the other vertices have sent their messages before they can send their own.

In order to solve this, we alter the algorithm in Figure 3.1 by, instead of calculating each message once it is ready to be calculated, repeatedly recalculating all the messages in some order until convergence is reached and the messages stop

[1] This is somewhat of a misnomer—a loop is typically defined as an edge between a vertex and itself, while the main idea of loopy belief propagation is that it applies belief propagation to Markov random fields on graphs containing *cycles*. However, the name has stuck.

```
Algorithm: Loopy belief propagation
Data: factor graph of Markov random field $x$ with potentials $\phi_i$
Result: marginal probabilities $P_{x_i}(\omega)\ \forall x_i \in x, \omega \in \Omega_{x_i}$
while any the messages $m_{a\to b}$ for some $a, b \in x$ have not converged do
    while a message $m_{a\to b}$ is ready to be computed do
        compute $m_{a\to b}$ according to (3.6);
    end
end
for each variable $x_i \in x$ do
    for $\omega \in \Omega_{x_i}$ do
        compute $P_{x_i}[\omega]\ \forall x_i \in x$ according to (3.4);
    end
end
```

Figure 3.7: Pseudocode for the loopy belief propagation algorithm on an arbitrary Markov random field.

changing. The order in which the messages are calculated is called the *schedule*. Figure 3.7 shows pseudo-code for the loopy belief propagation algorithm.

Two important points regarding the loopy belief propagation algorithm have not yet been discussed. First, how do we choose a schedule? The choice of schedule impacts the number of iterations required before the messages converge. A good schedule is one where few message computations require the use of default values. Choosing an optimal schedule is not tractable, but several efficient schedules have been developed in the literature. Discussing the choice of schedule at a deeper level falls outside the scope of this thesis, but the interested reader may refer to Elidan, McGraw, and Koller (2006) and Sutton and McCallum (2007).

The second question is that of conditions for convergence. As previously mentioned, the loopy belief propagation does not always converge. Is it possible to specify a set of necessary conditions for loopy belief propagation to converge on some graph? This is for the most part unknown and is an active topic of research. Empirical research, however, shows that loopy belief propagation often converges, and to good approximations to the exact marginal probabilities (Murphy, Weiss, and Jordan, 1999). However, it is known to work less well on many types of Ising models, which are some of the most commonly used models (Stoop, Ott, and Stoop, 2006).

Another downside of loopy belief propagation is that it has exponential time complexity in the size of highest-order interaction in the original graph. This is due to the fact that each time an interaction vertex passes a message, it needs to sum over the value of each of its neighbours apart from the vertex the message will be sent to. This is not quite as limiting as the junction tree algorithm's exponential time complexity in the tree width, as a high tree width is much more common in practice, but can still be a problem.

3.2.2 Sampling-based inference methods

Imagine that we have a set of independent samples from the joint probability distribution $P(\mathbf{x})$ of our Markov random field $\mathbf{x}$. Can we use these samples to calculate approximate marginal probabilities for each variable $x \in \mathbf{x}$? In this section, we will first look at how to calculate the marginal probabilities from a set of independent samples. Having done that, we will present the Gibbs sampling algorithm for generating such samples from an arbitrary Markov random field.

Assume that we have generated a set of independent samples $\left\{\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^N\right\}$ from a Markov random field and want to calculate the marginal probability $\rho_0^{x_1}$ for the variable $x_1 \in \mathbf{x}$ with value 0. From the set of samples, we can extract the realisations of the specific variable x_1,

$$\left\{x_1^1, x_1^2, \ldots, x_1^N\right\}.$$

A certain proportion of these samples will take the value 0. By calculating this proportion, we get an approximation $\widetilde{\rho}_0^{x_1}$ to the marginal probability $\rho_0^{x_1}$:

$$\widetilde{\rho}_0^{x_1} = \frac{1}{N} \sum_{i=1}^{N} I(x_1^i = 0).$$

As long as the samples are independent, $\widetilde{\rho}_0^{x_1}$ will get arbitrarily close to the true value $\rho_0^{x_1}$ as $N \to \infty$ (Koller and Friedman, 2009, pp. 523–524). The question remains, however, of how to generate the samples required to perform this procedure. While there are a number of possible methods, in this text we will use Gibbs sampling, a Markov chain Monte Carlo method.

Markov chain Monte Carlo

Markov chain Monte Carlo (MCMC) describes a wide family of algorithms from computational statistics used for sampling from probability distributions. In this section, we will describe such methods as applied to Markov random fields and used to perform approximate inference.

In this discussion, we will assume that the reader has at least elementary knowledge about Markov chains, but we here present the definition of a Markov chain as a refresher.

Definition 20. A sequence $\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^n, \ldots$ of random variables, where each variable $\mathbf{x}^n$ can take a state from some set Ω, is called a *Markov chain* if each $\mathbf{x}^n$ satisfies the condition

$$P(\mathbf{x}^n \mid \mathbf{x}^1, \ldots, \mathbf{x}^{n-1}) = P(\mathbf{x}^n \mid \mathbf{x}^{n-1}).$$

```
Algorithm: Gibbs sampling
Data: Markov random field $x$, desired number of samples $N$, burn-in value $M$
Result: samples $x^M, x^{M+1}, \dots, x^{M+N}$
Set $x^0$ to an initial value, e.g. a vector of ones;
for $n \in \{1, \dots, M+N\}$ do
    $x^n \leftarrow x^{n-1}$;
    for $x_i \in x$ do
        $x_i^n \leftarrow$ sample from $P(x_i \mid x_{-x_i})$;
    end
end
```

Figure 3.8: Pseudocode for the Gibbs sampling algorithm used on a Markov random field.

The possible values that each variable x^i can take are known as the *states* of the Markov chain. A Markov chain is defined by the probabilities $p_{ij}^n = P(x^{n+1} \mid x^n)$ of transitioning from a state x^j to another state x^i. If this probability is independent of n, that is if $p_{ij}^n = p_{ij}\ \forall n$, we say that the Markov chain is *homogeneous*.

Let us now look at the marginal probability distribution $P(x^{n+1})$. By using the definition of a Markov chain and summing over all the possible states of the preceding step x^n, we find that the distribution function can be written on the form

$$P(x^{n+1}) = \sum_{x^n} P(x^n) P(x^{n+1} \mid x^n).$$

As $n \to \infty$, it would be useful if $P_{n+1}(x^{n+1})$ converged to some *stationary distribution*. In fact, we can show that if there is a number Δn such that there is a positive probability of going from each state x_1 to another state x_2 in Δn steps, then such a stationary distribution exists (Koller and Friedman, 2009, pp. 510–511).

Let us go back to Markov random fields and see how all this applies. Define a Markov chain $x^1, x^2, \dots$ as above where the set of states the Markov chain can take at some point x^n is equal to the sample space of the Markov random field x. Imagine that we define the Markov chain such that the stationary distribution is equal to the joint probability distribution of the Markov random field. If we could do that, it would be possible to sample from the Markov random field by sampling from that Markov chain.

The particular Markov chain Monte Carlo algorithm we will discuss and use in this thesis is known as *Gibbs sampling*. In Gibbs sampling, we consider each variable $x_i \in x$ in turn. Gibbs sampling is an iterative algorithm where, for each generated sample, we consider each variable in the Markov random field in turn, updating the sample based on a sample from the conditional distribution $P(x_i \mid x_{-x_i})$ conditioned on the previously generated samples for each of the other variables in x. Pseudo-code for the Gibbs sampling algorithm is displayed in Figure 3.8.

The Markov chain defined by this algorithm can be shown to converge to a stationary distribution given by the joint probability distribution, as long as the potential functions are positive (Koller and Friedman, 2009, p. 515). However, it can often take a significant number of iterations before the algorithm starts to output samples from the stationary distribution. In particular, it is obvious that the initial sample x^0 is not a sample from the stationary distribution, but it can often take hundreds or even thousands of iterations before the algorithm outputs useful samples. We usually let the Gibbs sampling algorithm generate a considerable number of samples before we start using the samples it generates, and these discarded samples are known as *burn-in samples*.

4 Approximating the junction tree algorithm

The intractability of the junction tree algorithm on general graphs motivates the search for approximate inference algorithms that can perform inference on big and complex graphs while maintaining a high quality of approximation. Approximate inference algorithms for Markov random fields is a well-studied field, and there exist several commonly used algorithms including the already described loopy belief propagation algorithm. In this chapter, we will introduce a novel approximate marginal inference algorithm that by deleting edges in the graph of a binary Markov random field reduces the width of its junction tree to a managable level, and by that rendering marginal inference using the junction tree algorithm tractable.

4.1 Outline of the approximation algorithm

Let us recall from the previous chapter that the junction tree algorithm has exponential time complexity in the tree width of the triangulated graph. Applying the exact junction tree algorithm to a given Markov random field rapidly becomes intractable for large, dense graphs with big cliques, such as the highly important Ising model. In this chapter, we will assume that we have chosen a method for generating a junction tree for our model's graph, and discuss the width of this junction tree rather than the computationally intractable tree width of the graph.

Let the parameter $\tau > 0$ be the greatest junction tree width at which the junction tree algorithm is tractable for a given Markov random field x represented by a graph $G = (V, E)$. The greatest tractable junction tree width may vary somewhat from computer to computer, and depending on how much time we are willing to spend on our calculations, but owing to the nature of exponential time complexity, the junction tree algorithm quickly becomes intractable even for the strongest supercomputers.

As the asymptotic time complexity of the junction tree algorithm is a function only of the width of the chosen junction tree, reducing the width of the junction tree to τ is a guaranteed way to make the computation tractable. We can do this by deleting edges in the original graph, until the junction tree generated from it has a tree width of no more than τ. Reducing the junction tree width to some $\tau > 0$ is guaranteed to be possible, since the tree width is 1 in the extreme case where we have deleted every edge in E.

With some desired tree width τ as an input variable to an edge-deleting approximation algorithm, the goal is then to select a set of edges E_{del} to be deleted such that the final graph does indeed have a tree width of τ while simultaneously optimizing for quality of approximation. Once we have selected the edges to be deleted, we delete them using some approximation which attempts to compensate for their deletion. In this chapter, we will propose an approximation algorithm which combines the described edge deletion procedure with a *mean squared error enegy approximation* which partly compensates for deleting the edges. We will first look at the details of the mean squared error approximation, and then discuss our strategy for choosing edges to delete. Finally, we will present a formalisation of our approximation algorithm along with pseudocode, and discuss some issues of performance.

4.2 Mean squared error energy approximation

In Chapter 2, we introduced the concept of a pseudo-Boolean function and discussed the fact that the energy function of any binary Markov random field is a pseudo-Boolean function. Suppose we have a Markov random field whose energy function can be written on the form

$$U(x) = \sum_{\Lambda \in S} \beta_\Lambda \prod_{i \in \Lambda} x_i \tag{4.1}$$

where S is the set of interactions in the Markov random field for which the corresponding β_Λ is nonzero as well as the subsets of those interactions. In order to approximate the model, we wish to delete an edge from the model. This corresponds to deleting a pairwise interaction from S. If the model contains interactions of higher order than pairwise interactions, this will require the deletion of all interactions containing that pairwise interaction as a subset.

Let S^* denote the set of interactions remaining after deleting the pairwise interactions E_{del} and all interactions containing any of the edges in E_{del} as a subset. We can now write our approximate model as

$$U^*(x \mid \beta^*) = \sum_{\Lambda \in S^*} \beta^*_\Lambda \prod_{i \in \Lambda} x_i. \tag{4.2}$$

The open question is what the set of interaction parameters β^* should be in the approximate model. Austad and Tjelmeland (2015) select interaction parameters in the approximate model by minimizing the error sum of squares.

$$\beta^* = \arg\min_{\beta^*} \sum_{\omega \in \Omega} \left(U(\omega) - U^*(\omega \mid \beta^*)\right)^2, \tag{4.3}$$

where $\Omega = \{0,1\}^n$ is the sample space of a binary Markov random field with n variables. We now present two theorems regarding this type of approximation of pseudo-Boolean energy functions. We refer to Austad and Tjelmeland (2015) for proofs of both theorems.

Theorem 2. *Let $U^*(x)$ and $U^{**}(x)$ be two approximations of a pseudo-Boolean energy function $U(x)$, and let $S^{**} \subseteq S^*$. Then*

$$\begin{aligned}\beta^{**} &= \arg\min_{\beta^*} \sum_{\omega \in \Omega} (U(\omega) - U^{**}(\omega \mid \beta^*))^2 \\ &= \arg\min_{\beta^*} \sum_{\omega \in \Omega} (U^*(\omega) - U^{**}(\omega \mid \beta^*))^2 .\end{aligned}$$

This theorem implies that we may approximate the energy function $U(x)$ in steps, as going $U(x) \to U^*(x) \to U^{**}(x)$ yields the same set of interaction parameters β^{**} as going directly $U(x) \to U^{**}(x)$.

Theorem 3. *Let $U(x)$ be a pseudo-Boolean energy function with a set of interactions S and let $U^*(x)$ be an approximation of that energy function with a set of interactions S^*, such that $|S \setminus S^*| = 1$, where we will denote $S \setminus S^*$ by k. We can then write*

$$\sum_{\omega \in \Omega} (U(\omega) - U^*(\omega \mid \beta))^2 = \beta_k \left(\sum_{\omega \in \{0,1\}^{|k|}} [U(\omega) - U^*(\omega \mid \beta)] \right).$$

This theorem gives an alternative expression for the error sum of squares in the case where we only approximate a single interaction. We can now construct our approximation. Let E_{del} denote the set of edges we wish to delete from the graph $G = (V, E)$. Deleting these edges requires the removal of each of the pairwise interactions corresponding to the edges $e \in E_{\text{del}}$, as well as all interactions that are supersets of any of those interactions.

First, let us look at the case of deleting only a single edge $e \in E_{\text{del}}$. The set of interactions to be deleted consists of the pairwise interaction that corresponds to the edge e, as well as all of its supersets.

$$S \setminus S^* = \{s \in S : e \subseteq s\}.$$

To delete e, we go through the interactions in $S \setminus S^*$ and approximate the energy function for each interaction in order of interaction size, until we have approximated every interaction $\Lambda \in S \setminus S^*$. At this point, the edge e has been deleted from the model. If we wish to remove additional edges, we may repeat the procedure.

The final question is how to minimize the error sum of squares in Equation (4.3) in practice. Austad and Tjelmeland (2015) present an expression for the interaction parameters in the approximate energy function that follows from deleting an edge.

Theorem 4. *Let $e \in E$ be an edge in a binary Markov random field with an energy function $U(x)$ and a set of interactions S. The minimum error sum of squares approximation resulting from deleting the edge e can then be written on the form in Equation (4.2) with a set of remaining interactions*

$$S^* = \{\Lambda \in S : e \nsubseteq \Lambda\}$$

and a set of interaction parameters $\beta^* = \left\{\beta^*_\Lambda \; \forall \Lambda \in S^*\right\}$ *with*

$$\beta^*_\Lambda = \begin{cases} \beta_\Lambda - \frac{1}{4}\beta^{\Lambda \cup e} & \Lambda \cup e \in S, \\ \beta_\Lambda + \frac{1}{2}\beta^{\Lambda \cup \{e_0\}} & \Lambda \cup \{e_0\} \in S,\ \Lambda \cup \{e_1\} \notin S, \\ \beta_\Lambda + \frac{1}{2}\beta^{\Lambda \cup \{e_1\}} & \Lambda \cup \{e_1\} \in S,\ \Lambda \cup \{e_0\} \notin S, \\ \beta_\Lambda & \textit{otherwise.} \end{cases} \tag{4.4}$$

This approximation is what we call the *mean squared error energy approximation*, and has its origin in Austad and Tjelmeland (2015, pp. 4–7).

4.3 Edge selection

We now have a way of deleting individual edges in a binary Markov random field and calculating the interaction parameters in the resulting approximate model. However, the question remains how the edge deletion should be done in practice.

Let x be a binary Markov random field defined on a graph $G = (V, E)$. Our goal is to delete a set of edges $E_{\text{del}} \subseteq E$ in x using the mean squared error energy approximation such that performing inference suing the junction tree algorithm is tractable while minimizing some error function. Let x^* be the binary Markov random field created as an approximation to x by deleting the set of edges E_{del} using the mean squared error energy approximation, and let $G^* = (V, E \setminus E_{\text{del}})$ be the graph of this reduced Markov random field. We then want to choose E_{del} such that we minimize some error function $\varepsilon(x^*)$, that is

$$E_{\text{del}} = \arg\min_{E_{\text{del}}} \varepsilon(x^*), \tag{4.5}$$

while simultaneously ensuring that G^* has a junction tree width of at most τ, such that inference is tractable using the junction tree algorithm. The error function $\varepsilon(E_{\text{del}})$ may be any suitable function, for example the mean relative error of the marginals calculated in the approximate model compared to the marginals calculated in the exact model. The choice of error function will be discussed further in Chapter 6.

Solving the above optimization problem exactly is intractable, as it requires us to try every combination of edges to find whatever combination minimizes the error. Instead, we will attempt to construct a method of deleting a set of edges E_{del} such that the error is not necessarily minimized, but merely is "as low as possible" for the types of models in which we are interested in performing inference. We call the way that we select edges to delete the *edge deletion strategy*.

Several different edge deletion strategies are possible. One strategy could be to delete random edges in the graph until the width of the junction tree happens to reach our goal τ. As the tree width of a graph without any edges is 1, the junction tree width is guaranteed to reach τ even when deleting random edges in the graph. If we do this, however, we run the risk of deleting many unnecessary edges before reaching the desired value of τ, or deleting the wrong edges such that the quality of approximation is lowered.

Another possible way could be to look at each of the cliques of maximum size in the junction tree, repeatedly deleting edges in the largest cliques until we have managed to reduce the tree width by 1, and then repeating the procedure until the junction tree has the desired width. This "divide and conquer" algorithm sounds attractive, as it allows us to focus on only some cliques in the graph. In practice, however, the triangulation process often recreates the deleted edges in the form of fill-in edges, which means that if we only ever delete edges from the biggest cliques we lose the guarantee that the algorithm eventually reaches the target tree width τ.

Instead, we choose a third approach in which we regard *every* clique in the junction tree in order, starting with the widest cliques, and delete an edge in each clique until the number of cliques of maximum width has been reduced. By repeating this procedure, the idea is that we will eventually reduce the size of all the cliques of maximum width, and through that reduce the junction tree width to our target τ. This approach allows us to focus on the biggest cliques in the junction tree, but because the strategy is able to delete every edge in the graph in the "worst case", we are guaranteed that the target tree width τ is always eventually reached.

Within each clique, we will need to make a choice of which edge to delete at each step of the algorithm. This will be done by ranking every edge in the clique according to some criteria, and choosing the highest-ranking edge for deletion. We will call a method used to rank the edges in a clique an *edge ranking method*. In the next chapter, we will discuss three different edge ranking methods, based on the Kullback-Leibler divergence, the sum of potentials involving the edge, and the number of fill-in edges created during the triangulation process. For the remainders of this chapter, we will simply assume that some edge ranking method has been chosen.

4.4 Formalisation of the algorithm

Assume we have a binary Markov random field represented on the form in (4.1) with a set of interactions S and a set of interaction parameters Λ, defined on some graph $G = (V, E)$. Our algorithm also takes as input a target junction tree width τ at which using the junction tree algorithm to calculate the marginal probabilities is feasible. At the start of the algorithm, we generate a junction tree $J = (C, W)$ for the graph G and calculate its width

$$\tau_0 = \text{Width}(J) = \max_{\lambda \in C} |\lambda|.$$

In the case that $\tau_0 \leq \tau$, we apply the junction tree algorithm as described in Figure 3.6 to the Markov random field without performing any approximations.

If $\tau_0 > \tau$, we enumerate all the cliques in the junction tree, and go through them starting with the widest cliques, that is, the cliques containing the most variables.

```
Algorithm: Edge deletion (DeleteEdge)
Data: Markov random field $x$ defined on a graph $G = (V, E)$, with a set of
      interactions $S = \{\Lambda \subseteq V\}$ and a set of parameters $\beta = \{\beta_\Lambda : \Lambda \in S\}$, edge
      to be deleted $e = \{i, j\}$
Result: Markov random field $x^*$ defined on a graph $G^* = (V, E_{\mathrm{del}})$, with a set of
        interactions $S^* = \{\Lambda \subseteq V\}$ and a set of parameters $\beta^* = \{\beta_\Lambda : \Lambda \in S\}$
$S^* \leftarrow \varnothing$;
$\beta^* \leftarrow \varnothing$;
for $\Lambda \in S$ do
    if $e \not\subseteq \Lambda$ then
        $S^* \leftarrow S^* \cup \{\Lambda\}$;
    end
end
for $\Lambda^* \in S^*$ do
    Update $\beta^*_{\Lambda^*}$ according to (4.4);
end
```

Figure 4.1: Algorithm for deleting a single edge in the graph of a Markov random field and adjust the interaction parameters according to the mean squared error energy approximation.

Working our way down the list of cliques, from the widest to the narrowest cliques, we delete one edge in each clique using the theory developed in Section 4.2 until either

1. the number of cliques of maximum size has been reduced from the original number,
2. the tree width has been reduced, or
3. we have gone through every clique.

This procedure is repeated until the goal junction tree width τ has been reached. At this point, we calculate the marginal probabilities by applying the junction tree algorithm to the approximate model that we have created.

Pseudocode for the subprocedure that deletes a single edge from the model is presented in Figure 4.1, and pseudocode for the full algorithm is presented in Figure 4.2. The main remaining issue is how to choose which edge to delete next, or equivalently, rank the edges according to some criterion. As previously mentioned, this will be discussed in the next chapter.

Algorithm: Approximate inference

```
Data: Markov random field $x$ defined on a graph $G = (V, E)$, with a set of
      interactions $S = \{\Lambda \subseteq V\}$ and a set of parameters $\beta = \{\beta_\Lambda : \Lambda \in S\}$, goal
      tree width $\tau \geq 1$
Result: Set of marginal probabilities $\left\{P_{x_i}\ \forall x_i \in x\right\}$
Build junction tree according to $G$;
Let $\tau_0$ be the width of the initial junction tree;
if $\tau \geq \tau_0$ then
    Calculate set of marginal probabilities for every variable $x_i \in x$ using the
    junction tree algorithm;
    return;
end
$E_{\text{ordered}} \leftarrow \text{RankEdges}(E, S, \beta)$;
while Width($J$) > $\tau$ do
    Let $|C_{\max}|$ be the number of cliques of maximum size;
    $|C_{\max-\text{start}}| \leftarrow |C_{\max}|$;
    Rank each edge in $G$ according to some criterion;
    for $\lambda \in C$ do
        if Width($J$) <= $\tau$ or $C_{\max} < C_{\max-\text{start}}$ then
            break out of For loop;
        end
        Delete the highest ranking edge in $\lambda$ that has not yet been deleted;
        Update junction tree according to the altered graph $G$;
    end
end
Calculate set of marginal probabilities for every variable $x_i \in x$ using the
junction tree algorithm;
```

Figure 4.2: Our approximate inference algorithm combining the mean squared error energy approximation with the junction tree algorithm, using the main approach described in Section 4.3.

4.5 Time complexity

As the goal of our approximation algorithm is to work around the exponential time complexity of the junction tree algorithm by bounding the junction tree width to some value τ, it is important that we do not reintroduce exponential time complexity at any point in the algorithm. In addition, even if we know that the time complexity of the algorithm is polynomial, we would like for it to be as low as possible.

Because the focus of this thesis is not primarily on speed, and because the time complexity depends greatly on the details of the edge ranking methods and to some degree on the implementational details with regards to representation of the Markov random field, we will not conduct a full look into the time complexity of the algorithm here. Instead, we will discuss some issues of time complexity of the individual parts of the algorithm. As we have already discussed, a graph may be triangulated to a non-optimal chordal graph and a junction tree may be generated from that chordal graph in polynomial time.

The complexity of deleting an edge using the mean squared error energy requires deleting all interactions containing the variables in that edge. The time complexity of this operation, done on an efficient representation of the Markov random field, is a polynomial function of the size of the largest interaction in the graph. Finally, with a fixed upper junction tree width bound τ, the tree width is no longer a variable in the time complexity of the junction tree algorithm. The belief propagation stage of the junction tree algorithm is polynomial in the number of variables in the Markov random field (Ajroud et al., 2012), so we also escape exponential time complexity here.

5 Edge ranking methods

Let x be a Markov random field with probability distribution $P(x)$, defined on some graph $G = (V, E)$. As outlined in the previous chapter, our goal is to rank the edges in a clique $\lambda = (V_\lambda, E_\lambda)$ according to some criterion, allowing us to select the "best" edge to delete. In practice, we will do this by assigning a *score* $Z(e)$ to each edge $e \in E_\lambda$, and ranking the edges according to their score. The approximation algorithm can then delete the edges with the highest scores in turn for as long as necessary until the number of cliques of maximum size has been reduced.

In this chapter, we will discuss three different methods for assigning scores $Z(e)$ to edges: Kullback-Leibler scoring, lowest potential scoring, and minimum fill-in scoring. All three methods will be used in the numerical experiments and compared.

5.1 Kullback-Leibler scoring

Let $P_{-e}(x)$ denote the probability distribution of x resulting from deleting the edge $e \in E$ using the mean squared error energy approximation. The Kullback-Leibler divergence $D(P \| P_{-e})$ from P to P_{-e} quantifies the amount of *information* lost in the approximation. Intuitively, one would then expect that using a score

$$Z_{\mathrm{KL}}(e) = \frac{1}{D(P \| P_{-e})}$$

to rank the edges would lead to good results. Attempting to calculate this score, however, poses a problem. The Kullback-Leibler divergence from P to P_{-e} may be written as

$$D(P \| P_{-e}) = \sum_{\omega \in \Omega} P(\omega) \log \frac{P(\omega)}{P_{-e}(\omega)},$$
$$D(P \| P_{-e}) = \frac{1}{Z} \sum_{\omega \in \Omega} \mathrm{e}^{-U_P(\omega)} \Big(U_{p_{-e}}(\omega) - U_p(\omega) \Big).$$

The factor $\frac{1}{Z}$ is difficult to compute, but is of no bother as we do not need the absolute divergences to be able to rank edges. A greater issue is the fact that in order to calculate this expression, we need to sum over every value $\omega \in \Omega$, which takes exponential time.

In order to avoid the exponential time complexity, we need to use an approximation of the KL divergence. As the divergence is only used as a method for creating an approximate ranking of edges, an excellent approximation is not necessary. Because of this, we can approximate the KL divergence using Gibbs sampling with a low burn-in. By generating samples $x^1, \ldots, x^N$ from the exact joint probability distribution using Gibbs sampling, we may calculate an approximation $\widetilde{D}(P\|P_{-e})$ to the Kullback-Leibler divergence from P to P_{-e} using the parametric estimator (Budka, Gabrys, and Musial, 2011)

$$\widetilde{D}(P\|P_{-e}) = \sum_{i=1}^{N} \log \frac{P(x^i)}{P_{-e}(x^i)}.$$

We refer to Section 3.2.2 for an introduction to Gibbs sampling. Note that in this calculation, we assume that the normalisation constants of the original and approximate models are equal, up to a factor given by the mean square error approximation. This is not strictly true, but is a necessary simplification for the computations to be tractable.

In order for Kullback-Leibler scoring using Gibbs sampling to be tractable, we must avoid using too many samples and too many burn-in iterations. In our implementation, we have used 10 burn-in iterations and 20 samples. The exact number of iterations needed is unclear, and it is likely that this number is too small for large models. When Gibbs sampling is typically used one uses far more burn-in iterations than this, but for tractability reasons that was not possible here.

Note that in our above calculations, we assume that the normalisation constants of the exact and approximate models are equal, allowing us to avoid computing their ratio. This is not exactly true. However, as the mean squared error approximation attempts to compensate for the edge deletion by adding a constant to the energy function, and attempting to deal with two different normalisation constants would add a great deal of complexity to our calculations, we assume that the ratio of the exact and approximate normalisation constants is roughly equal to one.

5.2 Lowest potential

Another potential way of judging the “impact” of an edge in a pairwise model is to look at the potential of the corresponding pairwise interaction. Consider a pairwise model written on the exponential form with an energy function

$$U(x) = \sum_{x_i \sim x_j} \beta_i x_i x_j,$$

where $x_i \sim x_j$ denotes adjacency. If the interaction parameter β is close to 0, the interaction is in a sense “almost not there”. In other words, that interaction has a lower impact on the model as a whole than do interactions with parameters differing strongly from 0. Intuitively, removing an edge corresponding to a pairwise interaction with a parameter close to 0 then has less of a negative impact on the accuracy of the final marginals than removing some other edge does.

Consider now instead a model with interactions of arbitrary order. Particularly if the model does not have any pairwise interactions, it no longer makes sense to use the pairwise interaction strength as a proxy for the importance of an edge. We can generalise the above idea into a general ranking method, in which the product of the potential functions the edge ends are involved in is used as a criterion for judging edge importance.

Consider in particular a binary Markov random field with an energy function that can be written on the form (2.10). A generalised score Z_e for ranking an edge e could then be written

$$Z_e = \mathrm{e}^{\sum_{\lambda \in C_e} \beta_\lambda}, \tag{5.1}$$

$$C_e = \{\lambda \in C : e \subseteq \lambda\}. \tag{5.2}$$

However, every interaction does not “matter” to the same degree. Let us consider a second-order interaction of variables x_1, x_2 and x_3. Using the above scoring method for an edge $\{x_1, x_2\}$, a second-order interaction with parameter $\frac{1}{2}$ will have the same impact on the score as a pairwise interaction with that parameter. Because of the fact that there is a third variable involved in the interaction, it will only be “on” half the time. We can take this into account in the ranking score by instead using the modified score

$$Z_e = \mathrm{e}^{\sum_{\lambda \in C_e} \frac{1}{2^{|\lambda|-2}} \beta_\lambda}, \tag{5.3}$$

$$C_e = \{\lambda \in C : e \subseteq \lambda\}. \tag{5.4}$$

where each additional variable in an interaction halves the impact of the interaction parameter on the edge’s score. This is the score we will use in our numerical experiments.

5.3 Minimising fill-in

The process of triangulating a graph will create a certain amount of *fill-in*, which is the term for the edges that are added to make the graph chordal. Generally, a greater number of fill-in edges leads to bigger cliques in the graph’s junction tree, which is something we want to avoid. If we could lessen the number of fill-in edges, we could reduce the size of the cliques in the junction tree and thus its width, rendering inference easier.

Consider the graph of some Markov random field, with two edges e_0 and e_1. Let us say that triangulating this graph results in n fill-in edges. If we delete e_0 and again triangulate the graph, we again get n fill-in edges. However, if we delete e_1, we only get $m < n$ fill-in edges when triangulating the graph. By consistently choosing to delete the edges that result in triangulations with the lowest number of fill-in edges, we may more quickly reduce the “complexity” and width of the triangulated graphs, and by that reach our goal junction tree width τ more quickly than if we chose to delete edges that resulted in triangulations with a high number of fill-in edges.

This strategy is different from the previous two strategies in that instead of attempting to choose edges to delete such that the impact of each edge deletion is as small as possible, we instead attempt to minimise the *number* of edges we need to delete in order to reach the goal. In other words, we posit that deleting a small number of edges where each may or may not cause a significant error may still lead to a better end result than deleting a large number of low-impact edges.

Let $G = (V, E)$ be the original graph, let $G_{-e} = (V_{-e}, E_{-e})$ denote the resulting graph after deleting an edge e using the mean squared error approximation, and denote the triangulation of a graph G by $G^{\text{tri}} = (V^{\text{tri}}, E^{\text{tri}})$. This ranking method may then be defined through the score

$$Z_e = \frac{1}{\left|E_{-e}^{\text{tri}}\right|}. \tag{5.5}$$

6 Experimental setup

The goal of this chapter is to apply our approximation algorithm to a number of example Markov random fields in order to quantify the quality of approximation and to compare our algorithm to other standard methods. We will use our algorithm to calculate the marginal probabilities of every variable in each of our example models.

6.1 Error function

To allow the comparison of our approximation algorithm with other methods, we need an error function that can quantify the quality of approximation. The output of each algorithm is the marginal probability of each state for every variable in the Markov random field, and the error function must distill this data into one number that represents the *overall quality* of the approximate inference algorithm as applied to that model.

We will use three error functions that take into account different aspects of the approximation quality. Both compare the output of our approximate inference algorithm to some *reference data*. The reference data is ideally the output of an exact inference algorithm, such as the junction tree algorithm. However, in many cases calculating the exact marginals is intractable. In those cases, we we will instead compare our results to a those of a long-running Gibbs sampler used for inference as described in Section 3.2.2.

The first error function is the mean relative approximation error (MRAE). Let x be some Markov random field with probability distribution $P(x)$ and sample space $\Omega = \{0,1\}$ for each variable $x_i \in x$. Furthermore, let $\rho_s^{x_i}$ denote the reference marginal state probabilities for a variable x_i in state s, and let ${}^*\rho_s^{x_i}$ denote the marginal state probabilities calculated by our approximate inference algorithm. If we let $\Delta\rho^{x_i} = \rho_0^{x_i} - \rho_1^{x_i}$ denote the difference between the marginal probabilities for the outcomes 0 and 1, then the mean relative approximation error can be written as

$$\varepsilon_{rel} = \frac{1}{|x|} \sum_{x_i \in x} \left| \frac{\Delta\rho^{x_i} - \Delta^*\rho^{x_i}}{\Delta\rho^{x_i}} \right|. \tag{6.1}$$

Our second error function measures the maximum difference between a reference marginal state probability $\rho_s^{x_i}$ and an approximate marginal state probability ${}^*\rho_s^{x_i}$. This is a useful measure to get an impression of the worst case performance of an approximate inference algorithm. We define this error function as

$$\varepsilon_{max} = \max_{x_i \in x, \omega \in \Omega} \left| \rho_s^{x_i} - {}^*\rho_s^{x_i} \right|. \tag{6.2}$$

The final error function is a "misclassification" rate, given by the expression

$$\varepsilon_{class} = \frac{1}{|x|} \sum_{x_i \in x} \left[I(\rho_0^{x_i} < 0.5, {}^*\rho_0^{x_i} > 0.5) + I(\rho_0^{x_i} > 0.5, {}^*\rho_0^{x_i} < 0.5) \right], \tag{6.3}$$

where I is the indicator function as defined in Section 3.1. This error function indicates the frequency that the approximate marginal probability distributions $P_{x_i}(x_i)$ satisfy $P^*_{x_i}(0) > P^*_{x_i}(1)$ when $P_{x_i}(0) < P_{x_i}(1)$, or that $P^*_{x_i}(1) > P^*_{x_i}(0)$ when $P_{x_i}(1) < P_{x_i}(0)$.

6.2 Example models

In the numerical experiments, we will apply our approximate inference algorithm to a number of example Markov random fields to study its performance compared to other approximate inference algorithms. We will now describe the classes of models that we will use in the experiments.

6.2.1 204 and 500 edge Boltzmann machines

Murray and Ghahramani (2004) presents two randomly generated Boltzmann machines, each with 100 nodes and 204 or 500 edges. A Boltzmann machine is a binary Markov random field with unary and pairwise interactions where the interaction parameters may be independent of each other, that is, a binary Markov random field with the energy function

$$U(x) = \sum_{x_i \in x} \alpha_i x_i + \sum_{x_i, x_j \in x} \beta_{ij} x_i x_j. \tag{6.4}$$

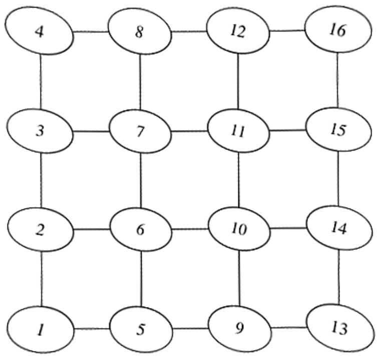


Figure 6.1: The graph of an Ising model of size 4×4.

Any interaction constant α_i or β_{ij} may be zero, and if a constant β_{ij} is non-zero then the corresponding graph of the Markov random field will contain an edge between the nodes corresponding to variables x_i and x_j. The sum $\sum_{x_i \in x} \alpha_i x_i$ is also known as the *bias term*.

Using the standard method in Figure 3.4 for triangulation, the 204 edge and 500 edge models have tree widths of 46 and 66, respectively. These are dense models that will likely require the deletion of a high number of edges in order to allow for tractable inference.

6.2.2 Ising model

An Ising model is a Boltzmann machine defined on a regular grid. Figure 6.1 displays an example of the graph of an Ising model. We briefly discussed Ising models in Chapter 2, and mentioned that they have their origin in physics. However, Ising models are also regularly used in the field of computer vision and in image reconstruction. Using our standard triangulation method, an Ising model with n^2 variables defined on a square grid will have a junction tree width of $n+1$, and will have $n^2 - 2n + 2$ cliques of maximum size. This means that performing inference on large Ising models rapidly becomes intractable.

In our experiments, we will look at Ising models both with and without bias terms. With identical interaction parameters, the energy functions for these models can be expressed on the form

$$U(x) = \beta \sum_{x_i, x_j \in x} x_i x_j, \tag{6.5}$$

$$U(x) = \alpha \sum_{x_i \in x} x_i + \beta \sum_{x_i, x_j \in x} x_i x_j. \tag{6.6}$$

In order to distinguish models with and without bias terms, we will refer to them as the *non-biased* and *biased* Ising models respectively. In addition to the above models with identical interaction parameters, we will also look at Ising models with energy functions on the form (6.4) where each parameter α_i and β_{ij} is selected from the uniform distribution on the interval $(0.0, 1.0)$. Finally, we will look at models where α and β are very large or very small in order to gauge if this has an impact on the performance of our algorithm.

6.2.3 Higher-order Markov random fields

Thus far, we have looked at Markov random fields with pairwise or *second-order* interactions, which means that the energy function of the Markov random field contains potentials that are functions of cliques of at most two variables. A Markov random field with *higher-order* interactions, conventionally called a higher-order MRF, has potentials that are functions of cliques of more than two variables. A Markov random field x of at most order N has an energy function that can be written on the form

$$U(x) = \sum_{n=1}^{N} \sum_{\lambda \in \Lambda : |\lambda| = n} f_\lambda(x_\lambda), \tag{6.7}$$

where the inner sum is over all cliques of size n.

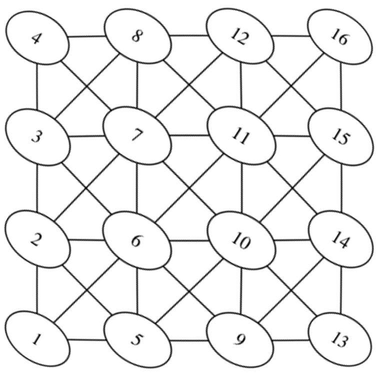


Figure 6.2: The graph of a third-order Ising model analogue on a 4×4 grid.

While pairwise Markov random fields are the most widely used, higher-order MRFs have been successfully applied to problems in image processing (Tjelmeland and Besag, 1998). Of particular interest are the higher-order analogues of the Ising model. Figure 6.2

of the graph of a higher-order Ising analogue on a grid where each interior vertex is adjacent to eight other vertices. The maximal cliques in this example contain four vertices, allowing for fourth-order interactions.

We will look at a third-order Ising model analog defined on a grid similar to that in Figure 6.2, but where the energy function only involves cliques of two and three variables as opposed to the maximum of four. Our third-order Ising model analog can be written as

$$U(x) = \sum_{x_i \in x} \alpha_i x_i + \sum_{x_i, x_j \in x} \beta_{ij} x_i x_j + \sum_{x_i, x_j, x_m \in x} \gamma_{ijm} x_i x_j x_m. \tag{6.8}$$

with each $x_i \in x$ taking binary values. As before, the interaction parameters α_i, β_{ij} and γ_{ijm} will be drawn from a uniform distribution on the interval $(0.0, 1.0)$.

In addition to the third-order grid model, we will look at a model based on a dataset of mortality rates for liver and gallbladder cancer in the United States by Riggan et al. (1987), inspired by the work done using such a model in Austad and Tjelmeland (2015). The model is defined on a grid that represents the eastern United States, where each vertex in the grid corresponds to a geographical region. The sample space for each region is

$$\Omega = \{\text{high cancer mortality rate, low cancer mortality rate}\},$$

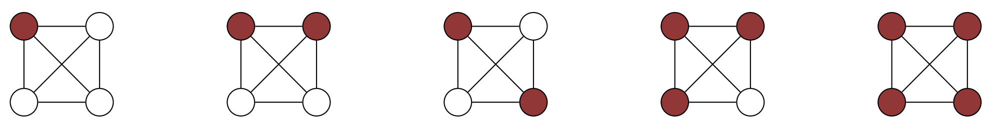
and we use a model with fourth-order interactions. With rotational symmetry, each interaction or clique has six different possible configurations, with interaction parameters θ_i. The energy function of our model may be written on the form

$$U(x \mid \theta) = \sum_{\Lambda \in C} \sum_{i=0}^{4} \theta_i I(x_\Lambda = c_i),$$

where the configurations c_i correspond to those displayed in Figure 6.3. When every variable in the clique is "off", the potential is set to zero. Compared to Austad and Tjelmeland (2015), which uses the full model with 66548 interactions, we will use a submodel with only 572 interactions. This is due to the limits put on our computational performance due to the choice of a lower performing programming language for our implementation and less effort devoted to optimizing the implementation for speed. The parameters chosen for this model are

$$\theta_0 = -0.50,\ \theta_1 = -0.62,\ \theta_2 = -0.60,\ \theta_3 = -0.77,\ \theta_4 = -0.32,$$

which roughly correspond to the simulated posterior values for these parameters given in Austad and Tjelmeland (2015). The exact values of these parameters is not important for our use.



(a) c_0 (b) c_1 (c) c_2 (d) c_3 (e) c_4

Figure 6.3: The distinct configurations c_i which determine the energy function for the US cancer model, where red indicates that a given variable is "on", taking the value 1, while white indicates that it is "off", taking the value 0. Each c_i has an associated parameter θ_i. The configurations are symmetric with regards to rotation and mirroring.

Our approximate inference algorithm	`our`
KL scoring (Section 5.1)	`our-kl`
Lowest potential (Section 5.2)	`our-potential`
Minimising fill-in (Section 5.3)	`our-fillin`
Junction tree algorithm	`junction`
Loopy belief propagation	`loopy`
Gibbs sampling	`gibbs`

Figure 6.4: Abbreviated names for the inference algorithms used in the following chapter.

6.3 Alternative inference algorithms

We wish to compare the performance of our inference algorithm to that of existing, standard approximate inference algorithms, which will be done by calculating the marginal probabilities of our example models using the algorithms in question. The algorithms that will be used in the comparison are loopy belief propagation and Gibbs sampling based inference, both of which have been described in Chapter 3.

As for the junction tree algorithm, we will use the implementations of the three alternative inference algorithms from the Matlab UGM library (Schmidt, 2007) to perform the computations. These implementations support only pairwise models. For the higher-order examples, we will restrict ourselves to comparing our results with those of the loopy belief propagation algorithm, using an implementation modified from that in the UGM library in order to support higher-order models.

Wherever there is not sufficient space to write the full name of an inference algorithm, we will use the abbreviations given in Figure 6.4. Whenever `gibbs` is followed by a number, that number indicates the number of samples used to perform inference. It does *not* indicate the number of burn-in samples, which will always be 1000. our followed by either `kl`, `potential` or `fillin` indicates which of the edge ranking methods are used.

6.4 Implementation

The software implementing the algorithms and procedures described in this thesis and used to perform the numerical experiments was written in the Matlab programming language, in order to allow for a greater amount of experimentation and more rapid changes in the code base. The implementations of the loopy belief propagation algorithm, Gibbs sampling-based inference and the junction tree algorithm were based on the code of the UGM toolbox for Matlab (Schmidt, 2007). However, as the UGM toolbox supports only Markov random fields with pairwise interactions, the implementations were rewritten to support higher-order Markov random fields.

The UGM toolbox used a representation of Markov random fields that was particularly tailored for Markov random fields with pairwise interactions, and also supported Markov random fields of variables that can take more than two values. However, as we were only interested in binary Markov random fields, and required support for higher-order interactions, we instead used a representation based on the representation of pseudo-Boolean functions (2.10). In this representation, a Markov random field x defined on a graph $G = (V, E)$ is represented as a set of pairs

$$\{(\lambda_1, \beta_1), (\lambda_2, \beta_2), \ldots, (\lambda_N, \beta_N)\},$$

where each $\lambda_i \subseteq x$ is a set of variables and β_i is an interaction parameter corresponding to that set of variables. The joint probability distribution of the Markov random field can then be written on the form

$$P(x) = \frac{1}{Z} \prod_{i=1}^{N} \mathrm{e}^{\beta_i \prod_{x_j \in \lambda_i} x_j}.$$

In Matlab, this was implemented using nested cell arrays. This representation, and the use of cell arrays in the implementation, allows for a high degree of flexibility. The downsides of this approach is that the representation must be converted to an adjacency matrix each time we want to do operations on the graph or junction tree of the Markov random field, and that the use of cell arrays can lead to lower performance due to less memory colocality. To ameliorate the latter issue, intermediate representations were sometimes used that offered superior performance in certain tasks.

For the mean squared error energy approximation, we used the implementation of Austad and Tjelmeland (2015), who first developed that approximation. Their implementation was written in C++, and glue libraries were written to connect our Matlab implementations with the C++ approximation code. The Markov random field representation used by Austad and Tjelmeland (2015), described in their paper, is similar to ours although more optimized, and converting between the two representations did not pose any problems.

7 Results

We will now present the results of applying our approximate inference algorithm to the example models introduced in the previous chapter. The results will be presented visually in the form of error plots, and quantitatively in the form of tables of errors and other measurements. When appropriate, the performance of our algorithm, both in terms of quality and in terms of speed, will be compared to that of the other approximate inference algorithms that have been discussed in this thesis. Less important plots and tables of data will be presented in Appendix A.

The results from performing approximate inference on the Ising model will be presented first. We will initially look at the non-biased 5×5 Ising model in order to demonstrate an example of the edge deletion procedure, and following that an 16×16 biased Ising model with parameters drawn from the uniform distribution on $(0.0, 1.0)$. Finally, we will see the results of inference in unbiased Ising models with all parameters set to the low value of $\beta = 0.05$ and the high value of $\beta = 2$.

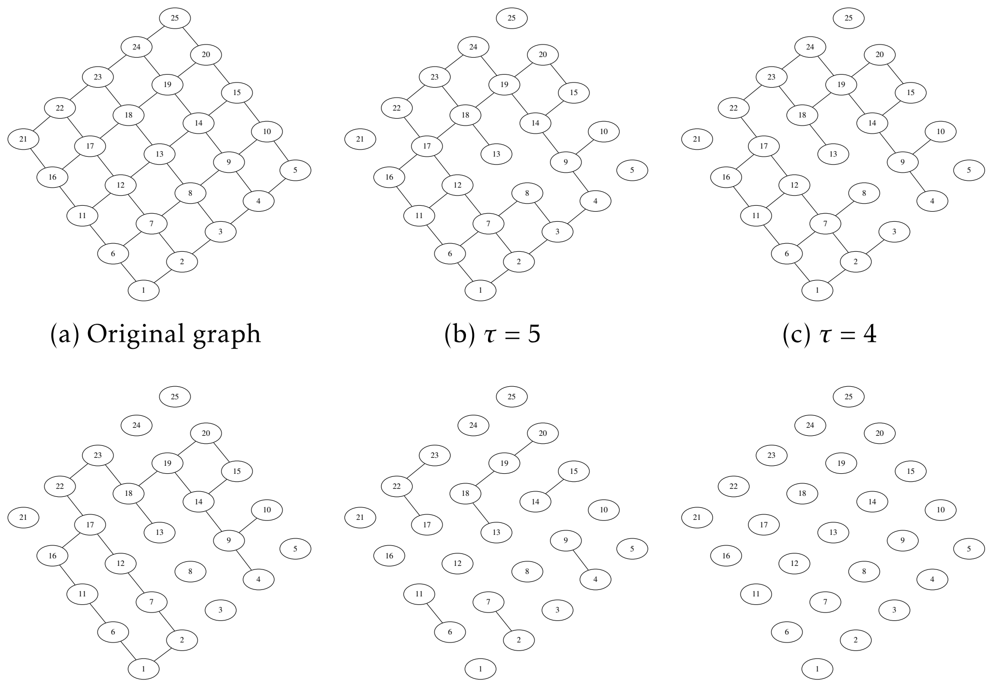
After the Ising model, we will apply our inference algorithm to the 204 and 500 edge Boltzmann machines from Murray and Ghahramani (2004). While these are still pairwise models, their graph structures are very chaotic compared to the regular structures of Ising models. Last, we will look at two higher-order models: a submodel of the U.S. cancer model from Austad and Tjelmeland (2015), and a 14×14 higher-order grid model. The graph structure of this grid model is displayed in Figure A.1 in the appendix.

For models with a low tree width, the marginals output by the exact junction tree algorithm will be used as the reference when calculating errors, but for bigger models in which exact inference is not tractable, we will use Gibbs sampling based inference based on 1000 burn-in iterations and 30000 samples. These numbers of burn-in iterations and samples were chosen by looking at the rate of convergence for each of the example models, and was found to work well for all of them.

7.1 Ising models

The first model we will approximate is the simple non-biased Ising model with 5×5 variables and a randomly chosen interaction strength β in the interval $(0.0, 1.0)$ on each of the pairwise interactions. This model has a total of 40 interactions and a junction tree width of 6, making it trivially tractable using the exact junction tree algorithm. The reason for including this model is to more easily be able to demonstrate the action of our inference algorithm, and we will not include the full set of data and plots for this model.

Figure 7.1 displays the original graph of this model as well as the graphs of the approximate models generated by applying our algorithm with $\tau \in \{5, 4, 3, 2, 1\}$. As expected, lower values of τ require the algorithm to delete increasing numbers of



(a) Original graph (b) $\tau = 5$ (c) $\tau = 4$

(d) $\tau = 3$ (e) $\tau = 2$ (f) $\tau = 1$

Figure 7.1: The original graph of the 5×5 non-biased Ising model, and the graphs of the approximate models generated by applying our approximation algorithm with with $\tau \in \{5, 4, 3, 2, 1\}$.

edges in order to reach the goal junction tree width. In this example, we selected edges according to the lowest potential criterion described in Section 5.2.

The table in Figure 7.2 displays the number of edges deleted as well as mean relative and maximum errors for the different values of τ. When $\tau = 6$, the algorithm does not delete any edges. At $\tau = 1$, on the other hand, every edge in the graph has been deleted. We see that the lowest potential edge ranking method in no way minimises the number of edges that are deleted: in d), the junction tree width could be reduced to 2 by deleting one edge in each of the two cycles, e.g. the edges $\{16, 17\}$ and $\{15, 20\}$. As we see in e), however, the algorithm instead deletes 11 edges before reaching a junction tree width of 2. While in general there is no guarantee that deleting two edges would necessarily be a better choice than deleting 11, as the two edges could be much more "impactful" than the 11.

Let us now look at a more interesting example, namely a biased 16×16 Ising model with pairwise interactions. All biases and pairwise interactions have parameters sampled from the uniform distribution on the interval $(0.0, 1.0)$. Calculating the exact marginal probabilities of this model is somewhat tractable, but very slow: on the author's computer, the exact computations took roughly one hour to finish. In other words, this is a prime target for an approximate inference algorithm such

τ	$\lvert E_{del}\rvert$	ε_{rel}	ε_{max}
1	40	0.0961	0.1067
2	31	0.0659	0.1067
3	20	0.0271	0.1067
4	14	0.0114	0.0749
5	11	0.0096	0.0749
6	0	0	0

Figure 7.2: Results of our applying our approximate inference algorithm to the 5×5 non-biased Ising model, using the lowest potential edge ranking criterion. Number of deleted edges $|E_{del}|$ as well as mean relative and maximum errors as a function of the parameter τ. At $\tau = 6$, the algorithm does not apply any approximations.

as ours.

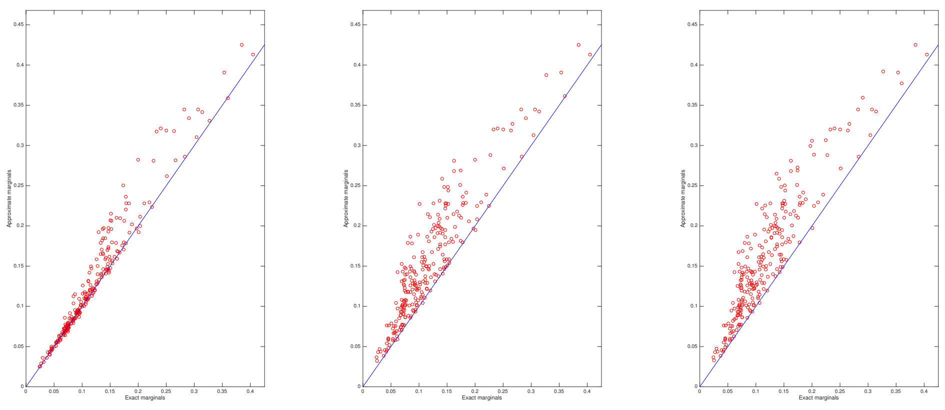
The results of running our algorithm on the biased 16×16 Ising model, using the lowest potential edge ranking method, are displayed in Figure 7.3. In addition to the number of edges deleted and the errors, we have also included the time taken to completion of the calculations. For models such as this one which are on the boundary of being tractable, even reducing the junction tree width by 1 allowed us to perform inference considerably faster. In this case, calculating the exact marginal probabilities took 3204 seconds, or just over 53 minutes, while calculating the approximate marginals with $\tau = 16$ took 318 seconds, more than 10 times quicker.

It is worth noting that the time taken to perform inference does not decrease monotonically with τ. Instead, it reaches a low point at $\tau = 12$ and $\tau = 13$, and is somewhat higher below that. This is due to the fact that at low junction tree widths, the junction tree algorithm's exponential growth in runtime has not yet overtaken the the cost of performing the approximations. In other words, the approximation algorithm has a higher constant factor, which leads to the approximate inference algorithm taking longer to complete than the exact algorithm when working with low junction tree widths. However, the effect is also partly due to inefficiencies in the implementation of the algorithm: the implementation of the approximation algorithm was written with an emphasis on code simplicity and readability, with performance concerns playing second fiddle, while the implementation of the exact junction tree algorithm originates in a proven software library and has been tuned for performance. For this reason, it is important not to read too much into the runtime of the approximation algorithm at low values of τ.

A useful tool in understanding the quality of the approximate marginals is the *correlation plot* between the exact and approximate marginals. Figure 7.4 displays correlation plots for the approximate marginal probabilities generated by our inference algorithm at $\tau \in \{16, 12, 9\}$. The marginals displayed in the plots are $\rho_0^{x_i}$ for all variables x_i in the model. The closer a point representing a marginal probability is to the diagonal, the lower is its difference from the true vaue. According to the error

τ	$\lvert E_{del}\rvert$	ε_{rel}	ε_{max}	ε_{class}	time (s)
9	307	0.1510	0.1256	0	152.8
10	301	0.1492	0.1210	0	150.3
11	267	0.1352	0.1202	0	140.9
12	228	0.1158	0.1202	0	130.2
13	204	0.1051	0.1143	0	130.2
14	174	0.0884	0.1143	0	135.2
15	160	0.0828	0.1085	0	153.4
16	88	0.0461	0.0951	0	318.0
17	0	0	0	0	3204

Figure 7.3: Results of our applying our approximate inference algorithm to the biased 16×16 Ising model, using the lowest potential edge ranking criterion. Number of deleted edges, mean relative errors, maximum errors, misclassification rate and time taken to completion of calculations as a function of the parameter τ.



(a) $\tau = 16$ (b) $\tau = 12$ (c) $\tau = 9$

Figure 7.4: Correlation plots between the exact marginal probabilities $\rho_0^{x_i}$ for the biased 16×16 Ising model and the approximate marginal probabilities generated by our approximate inference algorithm at $\tau \in \{16, 12, 9\}$.

data, most marginals should be close to diagonal in the plot for $\tau = 16$, while they should be considerably further away in the plots for $\tau = 12$ and $\tau = 9$. This appears to be true.

So far, we have only used the lowest potential method to select edges for deletion. We will now apply our algorithm to the 16×16 Ising model again, but this time we will use the Kullback-Leibler divergence and least fill-in methods in addition to the lowest potential method. Figure 7.5 displays a plot of the average relative error of the

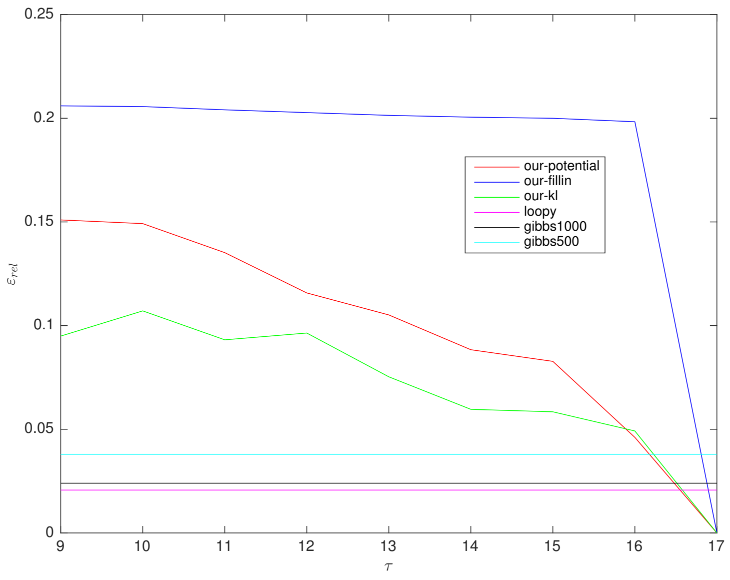


Figure 7.5: Mean relative error of the marginal probabilities for the biased 16×16 Ising model, as output by our approximate inference algorithm at different choices of τ and with different edge ranking methods. Also displayed are the mean relative errors of the marginal probabilities output by the loopy belief propagation algorithm (`loopy`) and sample-based inference based on 1000 (`gibbs1000`) and 500 (`gibbs500`) samples from a Gibbs sampler with 200 burn-in samples.

approximate marginals at different choices of τ for all three methods, in addition to lines representing the average relative errors of the approximate marginals generated by loopy belief propagation and Gibbs sampling-based inference. Tables of results for the Kullback-Leibler and least fill-in edge ranking methods can be found in Figures A.3 and A.4 in the appendix.

From the plot and result tables, we see that our algorithm appears to result in approximate marginals of lower quality than those output by both loopy belief propagation and the Gibbs sampling based method. This is true when using any of the edge ranking methods, although using the Kullback-Leibler and lowest potential edge ranking methods do result in better approximations than using the least fill-in edge ranking method. The plot of maximum error found in Figure A.2 in the appendix presents a similar picture. We do not include a plot of the misclassification rate as this was zero for every τ and edge ranking method.

In Figure 7.6, we see the time taken to completion of the calculations as a function of τ for the biased 16×16 Ising model. It is clear from the plot that the lowest potential method is much faster than the other two methods on this model, but whether this ultimately comes down to inherent slowness in the methods or an

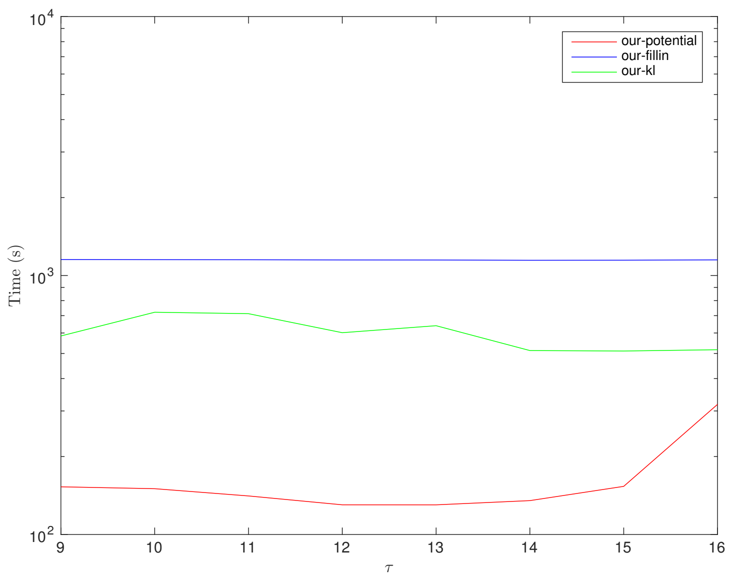


Figure 7.6: Time taken to completion of the algorithm as a function of τ for the biased 16×16 Ising model and each of the three edge ranking methods, plotted with a logarithmic scale on the vertical axis.

inefficient implementation is difficult to say. It is worth noting that before $\tau = 15$, the edge ranking and deletion procedures dominate the total time taken, and only at $\tau = 16$ does the exponential growth in time taken to perform the junction tree algorithm overtake the time spent selecting and deleting edges.

Finally, in Figure 7.7 we see a plot of the number of edges deleted as a function of τ using each edge ranking method. We see that in this model the least fill-in method deletes many more edges than the lowest potential method, which again deletes more edges than the Kullback-Leibler based selection method. In fact, the number of edges deleted using each method appears to correspond well to the quality of approximation for the methods, suggesting that the *number* of edges deleted has a close connection to the quality of the resulting approximate marginals.

7.2 Unbiased Ising models with low or high interaction parameters

In the previous section, we looked at biased Ising models with all parameters drawn from a uniform distribution between 0.0 and 1.0. We will now look at *unbiased* 16×16 Ising models with the parameter β set to either a value close to zero, $\beta = 0.05$, or a high value, $\beta = 2.0$, and see whether our inference algorithm performs differently here.

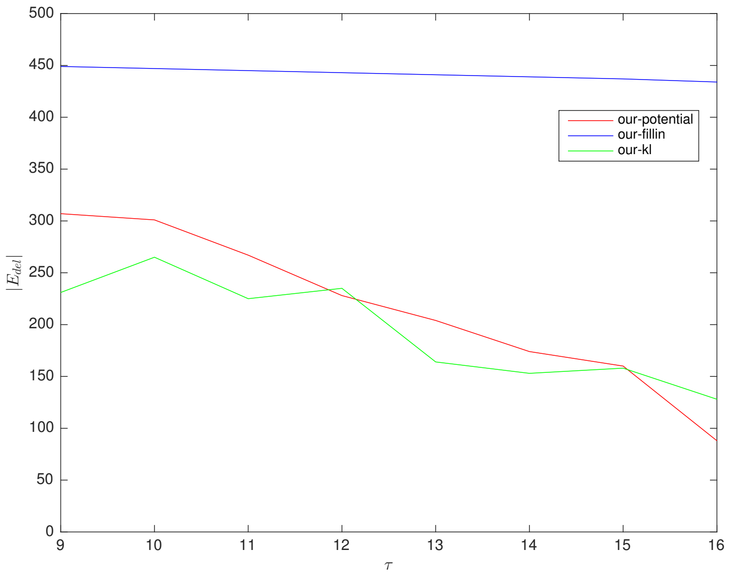


Figure 7.7: Number of edges deleted as a function of τ, for the biased 16×16 Ising model while using each of the three edge ranking methods.

Figures 7.8 and 7.9 contain plots of the mean relative error as a function of τ. It is clear that the performance of our algorithm relative to the other algorithms is very different between the two models. In the low-potential model, our algorithm performs very well, with low mean relative errors comparable to those of by loopy belief propagation regardless of the chosen edge ranking method. Meanwhile, Gibbs sampling based inference works poorly. In the high-potential model, on the other hand, our algorithm performs similarly well, but is now soundly beaten by both Gibbs sampling and loopy belief propagation.

Looking at the maximum errors in Figures A.5 and A.6, we see similar tendencies as in the relative error plots. The classification errors were zero regardless of choice of τ and edge ranking method, and for both models. The time plots in Figures A.7 and A.8 show similar results to earlier, namely that the Kullback-Leibler based edge ranking method is far slower than the other two methods. The plots of edges deleted as a function of τ in Figures A.9 and A.10 are also similar, except that the lowest potential edge ranking method now deletes as many edges as the least fill-in method at most τ. The tables of results for the low-potential model may be found in Figures A.11, A.12 and A.13, while the results for the high-potential model may be found in Figures A.14, A.15 and A.16.

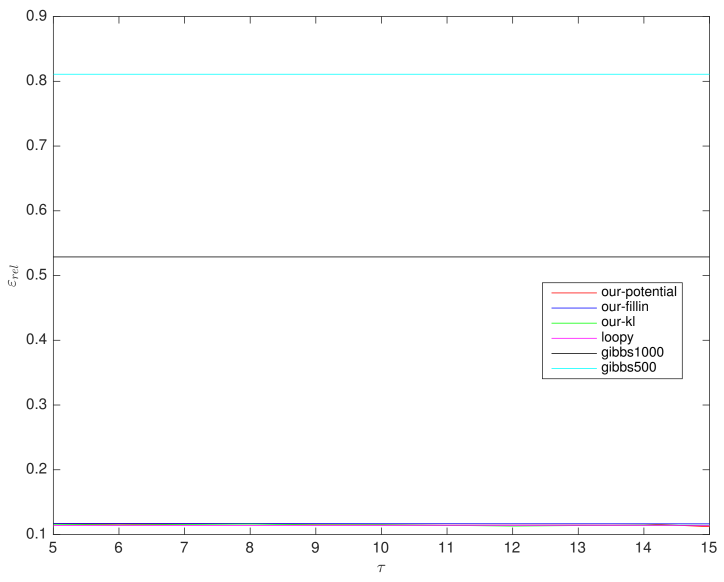


Figure 7.8: Mean relative error of the marginal probabilities for the unbiased 16×16 Ising model with $\beta = 0.05$, as output by our approximate inference algorithm at different choices of τ and with different edge ranking methods. Also displayed are the mean relative errors of the marginal probabilities output by the loopy belief propagation algorithm (`loopy`) and sample-based inference based on 1000 (`gibbs`1000) and 500 (`gibbs`500) samples from a Gibbs sampler with 200 burn-in samples.

7.3 204 and 500 edge Boltzmann machines

We now turn our attention to two other pairwise models, namely the 204 and 500 edge Boltzmann machines from MURRAY and GHAHRAMANI (2004). While Ising models are a widely used type of model, their regular structure means they are not necessarily a good representative for Markov random fields in general. The 204 and 500 edge Boltzmann machines are Markov random fields without any particular structure and may better represent the general performance of our algorithm.

Figure 7.10 displays the mean relative error for the 204 edge Boltzmann machine. As opposed to when we looked at the Ising models, none of the edge ranking methods here appear to perform consistently better than the others. As before, the other inference algorithms perform considerably better than does our algorithm, regardless of choice of edge ranking method. The results for the 500 edge Boltzmann machine are shown in Figure 7.11. For this model, loopy belief propagation did not converge, and the result shown for this algorithm is calculated based on the messages after 50 iterations. Our algorithm performs similarly to the "cut-off" loopy belief propagation algorithm, but again considerably worse than the results based

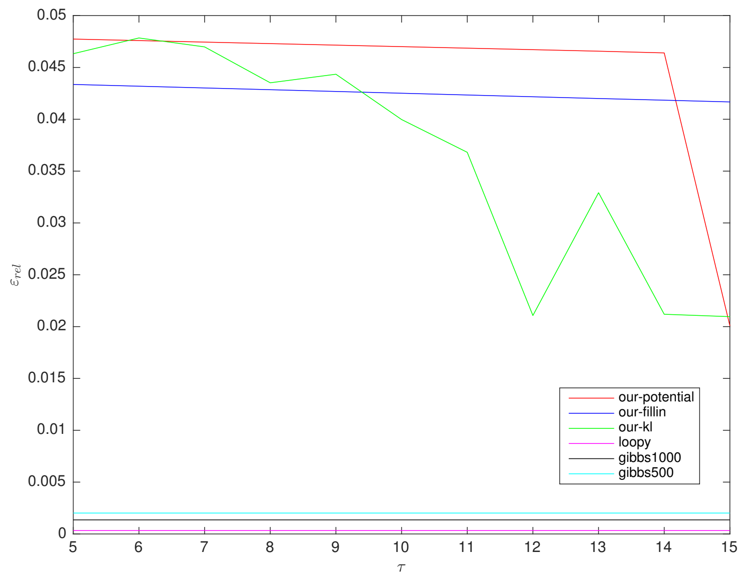


Figure 7.9: Mean relative error of the marginal probabilities for the unbiased 16×16 Ising model with $\beta = 2.0$, as output by our approximate inference algorithm at different choices of τ and with different edge ranking methods. Also displayed are the mean relative errors of the marginal probabilities output by the loopy belief propagation algorithm (`loopy`) and sample-based inference based on 1000 (`gibbs1000`) and 500 (`gibbs500`) samples from a Gibbs sampler with 200 burn-in samples.

on Gibbs sampling.

The two other error measures, the maximum error and the misclassification rate, lead to the same conclusions as the relative error, as seen in Figures A.17 and A.18 for the 204 edge model and Figures A.25 and A.26 for the 500 edge model.

Figure 7.12 shows the time taken to completion of the calculations as a functio of τ for the 204 edge Boltzmann machine. We see a similar behaviour as with the Ising models, except that the Kullback-Leibler based method now performs worse than the least fill-in method. A similar plot for the 500 edge model can be seen in Figure A.27.

As for the plot of number of edges deleted as a function of τ, this can be seen in Figure 7.13. The situation here is now reversed from what was the case for the Ising models, as the least fill-in edge ranking method now deletes the fewest edges. However, this does not necessarily rock the idea that quality of approximation is closely connected with the number of deleted edges, because the quality of approximation is much more similar between the three methods here than was the case for the Ising models. The same plot for the 500 edge model can be seen in

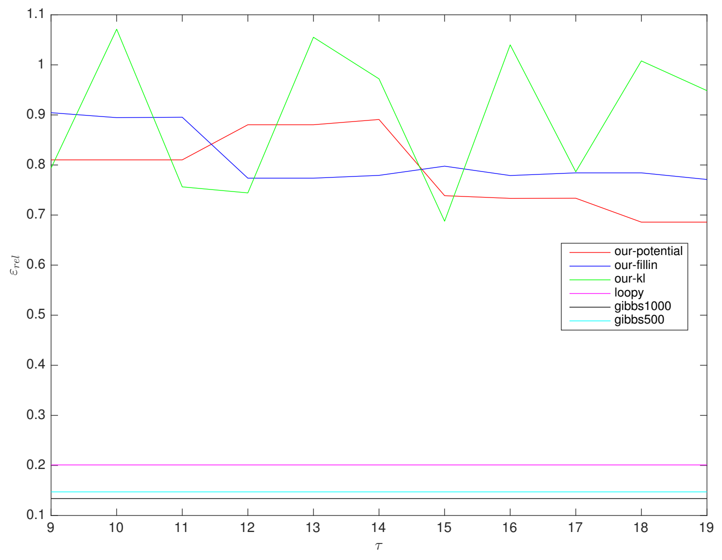


Figure 7.10: Mean relative approximation error of the marginal probabilities output by our approximate inference algorithm at different choices of τ, for the 204 edge Boltzmann machine, compared to the usual alternative approximate inference algorithms.

Figure A.28, and displays similar patterns.

Figures A.19, A.20 and A.21 in the appendix contain tables of the results for the 204 edge Boltzmann machine, while Figures A.22, A.23 and A.24 show the equivalent tables for the 500 edge model.

7.4 Higher-order models

While our inference algorithm did not perform particularly well on the previous models, we have thus far only looked at pairwise models. One of the barriers from putting higher-order models to greater use in fields such as computer vision and image restoration is their greater intractability, and it would be interesting to see if our algorithm could perform better on such models than traditional approximate inference algorithms.

We will first look at the U.S. cancer model. This model has a junction tree width of 24, rendering exact inference intractable on the author's hardware. Figure 7.14 displays the mean relative error, maximum error and misclassification rate for the approximate marginal probabilities at different values of τ. Loopy belief propagation did not converge for this model, and is not included for comparison. As the benchmark, Gibbs sampling based inference based on 25000 samples was used.

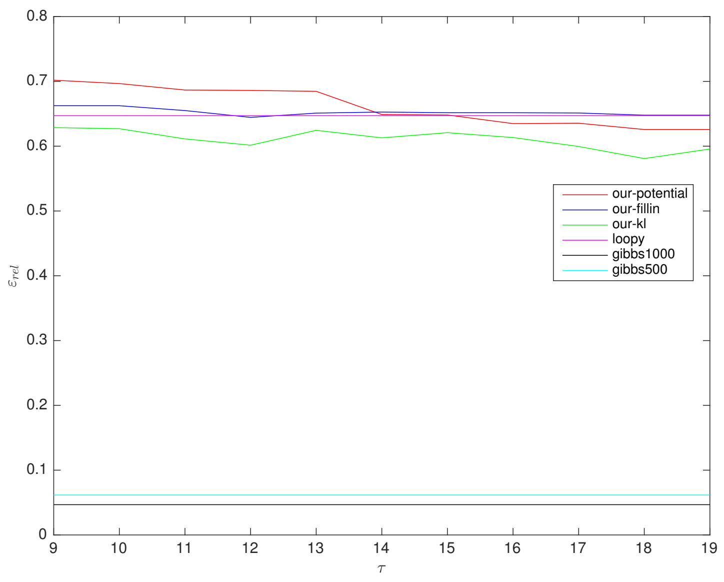


Figure 7.11: Mean relative approximation error of the marginal probabilities output by our approximate inference algorithm at different choices of τ, for the 500 edge Boltzmann machine, compared to the usual alternative approximate inference algorithms.

Compared to the pairwise models, our algorithm performs very well on the cancer model, largely outperforming the marginal probabilities generated by the Gibbs sampling based inference method. While each of the edge ranking methods performs well, the method based on least fill-in consistently gives among the lowest errors compared to the results from the other edge ranking methods and the Gibbs results.

Figures 7.15, 7.16 and 7.17 display tables of results for the U.S. cancer model using the lowest potential, Kullback-Leibler and least fill-in based edge ranking methods, respectively. The numbers confirm our intuition from looking at the graphs, namely that the least fill-in edge ranking methods is the best performing. This is particularly true when looking at the mean relative error. The downside, according to our results, is that the least fill-in method is more time-consuming than the lowest potential method. However, it is not clear whether this is due to an inefficient implementation or whether the method is inherently slower.

Figure 7.18 shows a plot of the time taken to completion as a function of τ for the U.S. cancer model and for each of the three edge ranking methods. Again, we see that there are wide differences in the efficiency of the edge ranking methods at low τ, but that this difference becomes less significant at high values of τ when time taken to perform the junction tree algorithm on the reduced model starts to dominate.

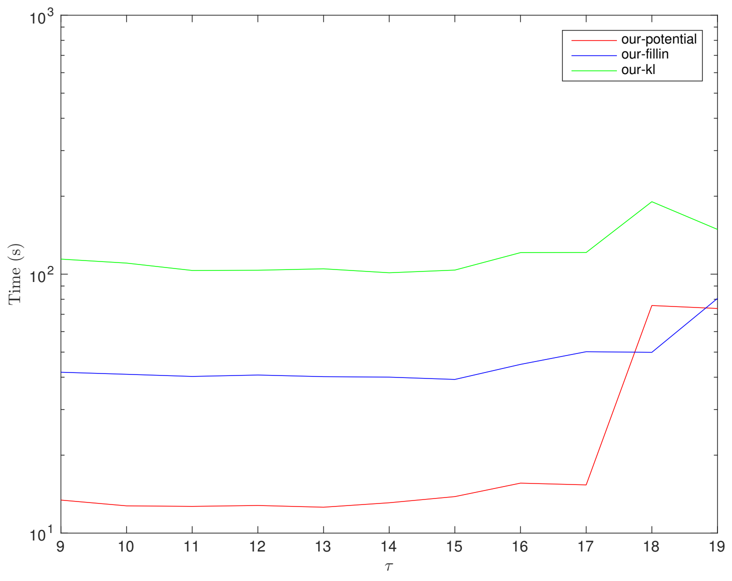


Figure 7.12: Time taken to completion of the algorithm as a function of τ for the 204 edge Boltzmann machine and using each of the three edge ranking methods, plotted with a logarithmic scale on the vertical axis.

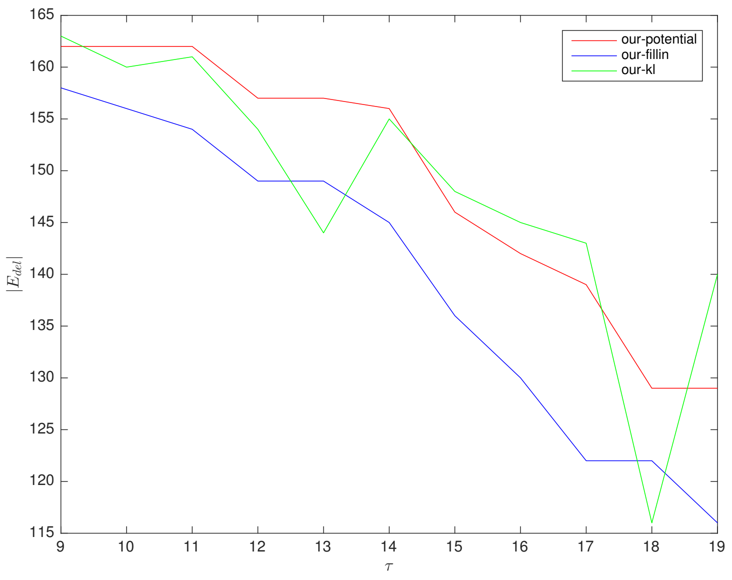
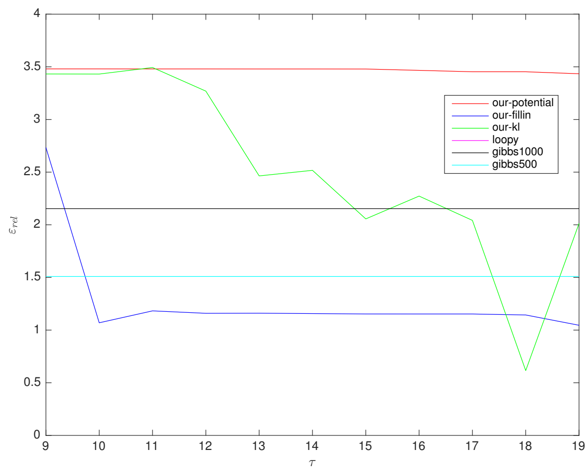
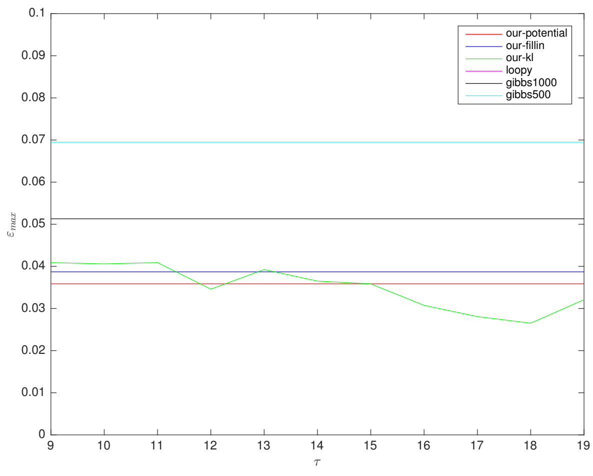


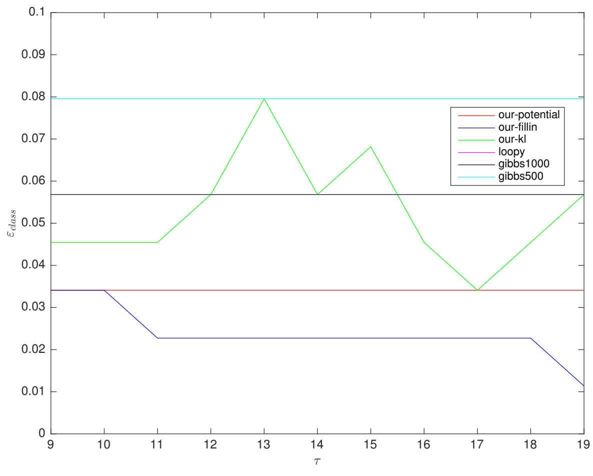
Figure 7.13: Number of edges deleted as a function of τ, for the 204 edge Boltzmann machine while using each of the three edge ranking methods.



(a) Mean relative error



(b) Maximum error



(c) Misclassification rate

Figure 7.14: Mean relative error, maximum error and misclassification rate for the U.S. cancer model, as a function of τ for our approximate inference algorithm and compared to Gibbs sampling based inference with 500 and 1000 samples. Loopy belief propagation did not converge, and is not included.

τ	$\lvert E_{del}\rvert$	ε_{rel}	ε_{max}	ε_{class}	time (s)
9	238	3.4805	0.0359	0.0341	15.9
10	237	3.4805	0.0359	0.0341	15.2
11	235	3.4801	0.0359	0.0341	15.0
12	234	3.4801	0.0359	0.0341	14.9
13	232	3.4797	0.0359	0.0341	15.2
14	231	3.4797	0.0359	0.0341	15.3
15	229	3.4792	0.0359	0.0341	15.9
16	200	3.4665	0.0359	0.0341	18.5
17	165	3.4531	0.0359	0.0341	92.4
18	165	3.4531	0.0359	0.0341	92.6
19	128	3.4341	0.0359	0.0341	479.3
24	0	0	0	0	—

Figure 7.15: Results of our applying our approximate inference algorithm to the U.S. cancer model, using the lowest potential edge ranking method. Number of deleted edges, mean relative errors, maximum errors, misclassification rate and time taken to completion of calculations as a function of the parameter τ.

τ	$\lvert E_{del}\rvert$	ε_{rel}	ε_{max}	ε_{class}	time (s)
9	257	3.4322	0.0409	0.0455	316.2
10	263	3.4314	0.0406	0.0455	399.0
11	237	3.4935	0.0409	0.0455	303.9
12	199	3.2693	0.0346	0.0568	391.5
13	211	2.4642	0.0393	0.0795	238.1
14	141	2.5171	0.0365	0.0568	340.5
15	137	2.0560	0.0359	0.0682	398.0
16	97	2.2725	0.0308	0.0455	395.4
17	78	2.0418	0.0281	0.0341	390.3
18	53	0.6156	0.0265	0.0455	463.8
19	54	2.0123	0.0321	0.0568	376.0
24	0	0	0	0	—

Figure 7.16: Results of our applying our approximate inference algorithm to the U.S. cancer model, using the Kullback-Leibler edge ranking method. Number of deleted edges, mean relative errors, maximum errors, misclassification rate and time taken to completion of calculations as a function of the parameter τ.

τ	$\lvert E_{del}\rvert$	ε_{rel}	ε_{max}	ε_{class}	time (s)
9	264	2.7340	0.0387	0.0341	74.6
10	252	1.0694	0.0387	0.0341	73.6
11	248	1.1825	0.0387	0.0227	73.2
12	244	1.1594	0.0387	0.0227	72.2
13	240	1.1606	0.0387	0.0227	72.5
14	236	1.1568	0.0387	0.0227	72.5
15	232	1.1531	0.0387	0.0227	74.7
16	229	1.1528	0.0387	0.0227	73.9
17	227	1.1528	0.0387	0.0227	81.9
18	222	1.1436	0.0387	0.0227	88.8
19	188	1.0458	0.0387	0.0114	455.5
24	0	0	0	0	—

Figure 7.17: Results of our applying our approximate inference algorithm to the U.S. cancer model, using the least fill-in edge ranking method. Number of deleted edges, mean relative errors, maximum errors, misclassification rate and time taken to completion of calculations as a function of the parameter τ.

Figure 7.19 displays the number of edges deleted as a function of τ. We see that the fill-in method deletes the most edges at almost every value of τ, despite performing well in terms of the error functions. This demonstrates that a low number of edges deleted does not necessarily correlate with small errors.

Our other higher-order model is a higher-order analog of the Ising model with interactions defined on all cliques of one, two and three variables, and each interaction parameter chosen from the uniform distribution on $(0.0, 1.0)$. In Figure 7.20, we see the mean relative and maximum errors for this model. While both Gibbs sampling based inference and loopy belief propagation perform better than our algorithm, the mean relative error and maximum error are still quite low at every tested value of τ. In situations where stochastic methods such as Gibbs sampling are to be avoided and loopy belief propagation does not converge, our method may be useful.

Figure A.29 shows the number of edges deleted as a function of τ, which displays the same pattern we have seen previously in which the least fill-in method deletes more edges than the other methods. Figure 7.21 displays the time to completion as a function of τ. For every one of the three edge ranking methods, the time curve is roughly flat because the time taken to perform the junction tree algorithm has not overtaken the time needed to perform the edge ranking and deletion even at $\tau = 12$. We again see that the Kullback-Leibler edge ranking method is very slow – in this case, slower than the lowest potential method by a factor of 36. Figures A.30, A.31 and A.32 in the appendix show full tables of data for each of the three edge ranking methods.

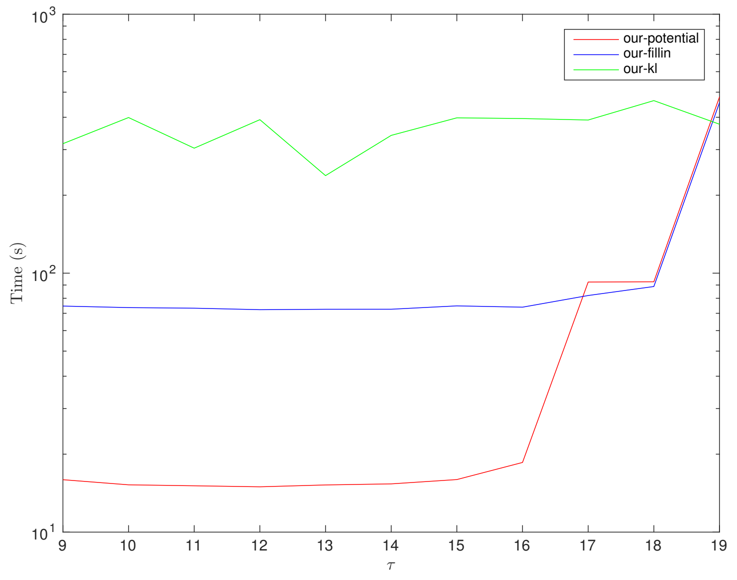


Figure 7.18: Time taken to completion of the algorithm as a function of τ with the vertical axis on a logarithmic scale, for the U.S. cancer model and each of the three edge ranking methods.

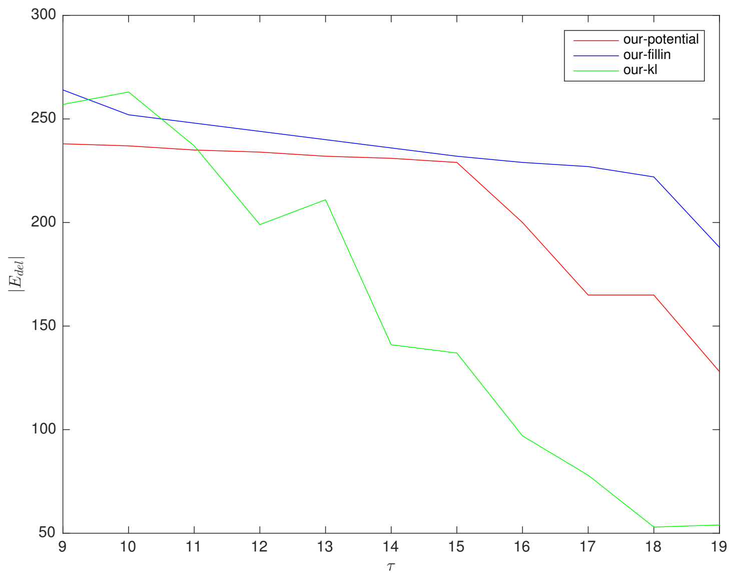
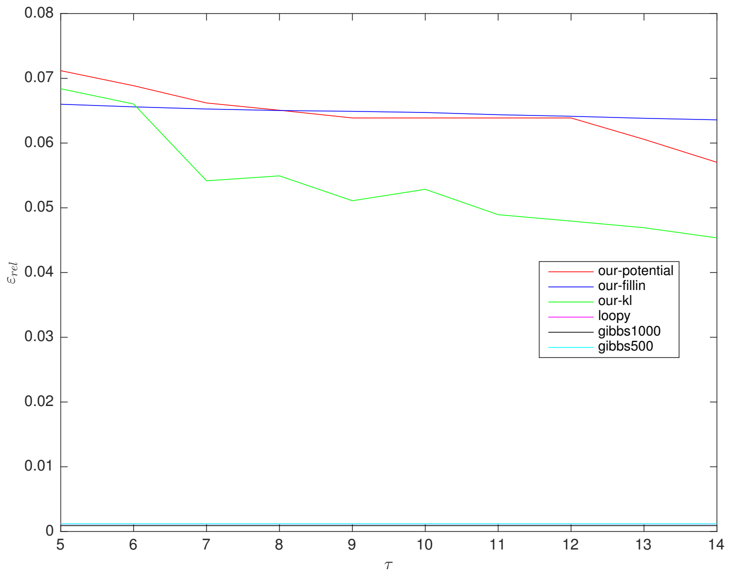
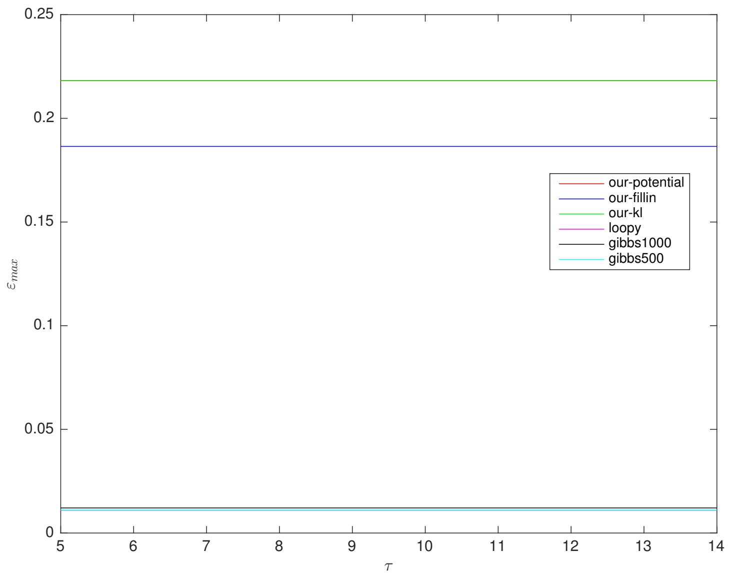


Figure 7.19: Number of edges deleted as a function of τ, for the U.S. cancer model and each of the three edge ranking methods.



(a) Mean relative error



(b) Maximum error

Figure 7.20: Mean relative error and maximum error for the 14×14 third-order grid-model, as a function of τ for our approximate inference algorithm and compared to Gibbs sampling based inference with 500 and 1000 samples as well as loopy belief propagation. The misclassification rate was zero at all values of τ and for all algorithms and methods. The lines for `our-potential` and `our-kl` overlap in the second plot.

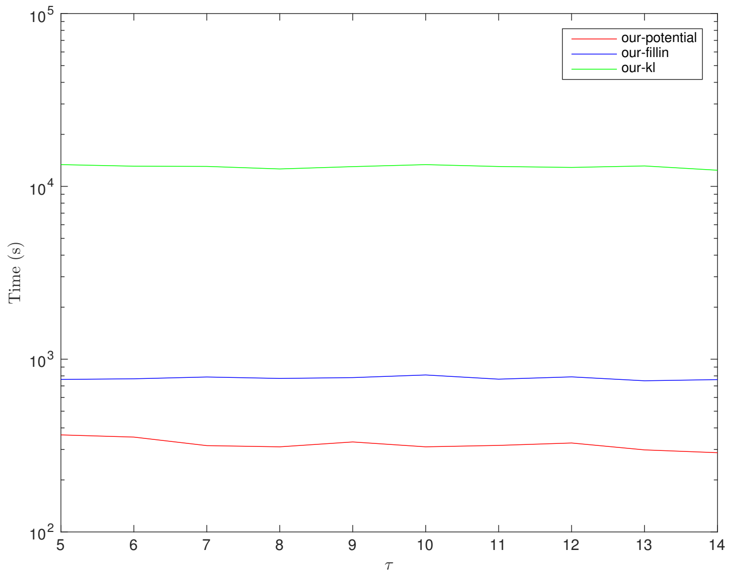


Figure 7.21: Time taken to completion of the algorithm as a function of τ with the vertical axis on a logarithmic scale, for the 14×14 third-order grid model and each of the three edge ranking methods.

7.5 Discussion

We have now seen the results of applying our approximate inference algorithm to four different Ising models, two Boltzmann machines of different complexity, and two higher-order grid models. It is clear that for most of the models, our algorithm does not outperform loopy belief propagation and Gibbs sampling-based inference. For the biased 16×16 Ising model with parameters drawn from $(0.0, 1.0)$, our algorithm resulted in considerably higher mean relative and maximum errors than loopy belief propagation and Gibbs sampling-basde inference, and the same is true for the 204 and 500 edge Boltzmann machines.

For the unbiased $16{\times}16$ Ising models with low or high parameters, our algorithm performs well. On the model with $\beta = 0.05$, it performs similarly to loopy belief propagation and outperforms Gibbs sampling-based inference with 500 and 1000 samples. While it performs worse than the comparison methods on the $\beta = 2.0$ model, the mean relative error is consistently below 0.05 at each τ between 5 and 15, and regardless of the edge ranking method used.

On the higher-order models, our algorithm generally performs well, performing similarly to the comparison methods for the U.S. cancer model and with mean relative errors below 0.08 in the 14×14 third-order grid model. Higher order models are also a category of models where loopy belief propagation can be slow due to that algorithm's exponential time complexity in the highest interaction order, and loopy belief propagation did not converge on the U.S. cancer model.

Of the three edge ranking methods, the performance of each varied considerably between each of the example models. The Kullback-Leibler edge selection method generally resulted in the lowest mean relative errors in the pairwise models, with the lowest potential and least fill-in methods tying for second best. In the US cancer model, the least fill-in method performed best in terms of mean relative error and misclassification rate, while also performing well in terms of maximum error. In the 14×14 third-order model, least fill-in gives the best result in terms of maximum error, while the Kullback-Leibler based method performs best in terms of mean relative error.

While the Kullback-Leibler method often performed well, we note that due to its stochasticity and the fact that we were forced to use a low number of samples to calculate it in order to ensure tractability, its performance was highly unstable. Indeed, in many of the plots, we see that we get a higher quality of approximation at a lower τ. While this is possible even with the lowest potential and least fill-in ranking methods, for the Kullback-Leibler method it is likely due to different edge choices in different runs depending on the samples generated using Gibbs sampling.

There appears to be a connection between the size of the error and the total number of edges deleted throughout most of the results. For the biased 16×16 model, the performance of each edge ranking method in terms of mean relative error closely follows the number of edges deleted. The by-far worst-performing method in this model, the least fill-in method, also deleted by-far the most edges. This connection, although weaker, is also present in the 14×14 third-order model. In the US cancer model, however, we see that the best performing method, least fill-in, also deletes the most edges, showing that there is not a simple connection between number of deleted edges and quality of approximation.

In terms of speed, the three edge ranking methods perform very differently. The lowest-potential edge selection method is consistently faster than the other two methods, with the least fill-in and Kullback-Leibler based methods being slower by a high factor when applied to most of the example models. In particular, the speed of the Kullback-Leibler based method is extremely slow in the 14×14 third-order model. The cause of this is unclear, but the reason may be inefficiencies in the implementation of Gibbs sampling which causes the runtime to blow up for certain kinds of models. Notably, the 14×14 model contains 1574 interactions, a considerably larger number than the other example models. Plotting the total runtime using the Kullback-Leibler edge ranking method as a function of the total number of interactions, as in Figure 7.22, does not present us with any obvious type of connection between runtime and number of interactions, however.

While our algorithm generally performs worse than the comparison methods, the results are generally decent to good for most of the example models we have looked at. This is particularly true with regards to the misclassification rate, which is zero much of the time. In the instances where loopy belief propagation may be unusable, either due to its exponential runtime as a function of the model order or due to the

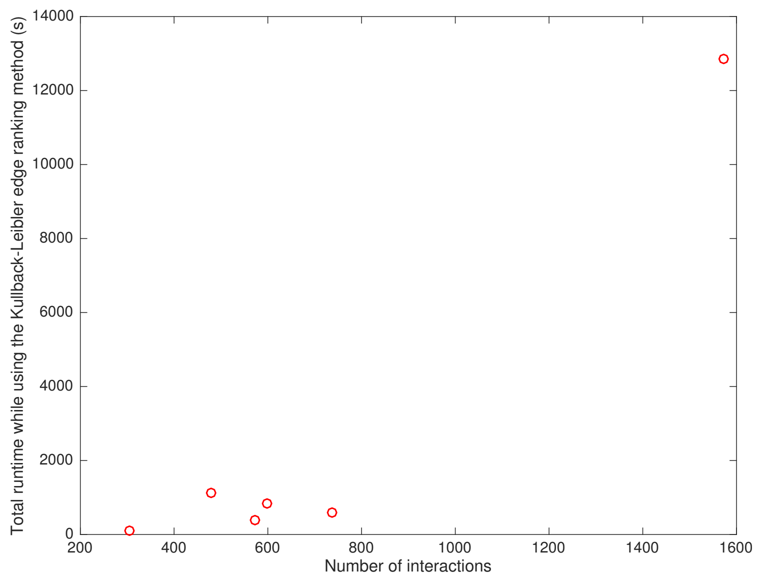


Figure 7.22: Scatterplot of the total runtime of our approximation algorithm as a function of the total number of interactions in the model, using the Kullback-Leibler edge ranking method and $\tau = 12$.

fact that it does not always converge, and Gibbs sampling-based inference is not an option due to its stochasticity, our algorithm may present a useful alternative. However, there exist several other approximate inference algorithms, including the entire field of variational methods, and the choice of inference algorithms should be made based on the properties of the specific model one is interested in.

8 Closing remarks

In this thesis, we have developed an approximate marginal inference algorithm for binary Markov random fields, based on deleting edges using a mean squared error energy approximation and running the junction tree algorithm on the reduced model. To select which edges to delete, we have used a strategy where we regard each clique in the junction tree in order of width and delete one edge from each clique until either the junction tree width or the number of cliques of maximum size has been reduced. We have proposed three different methods for choosing which edge to delete within each clique, based on lowest potential, the Kullback-Leibler divergence and least fill-in.

We have applied our approximate inference algorithm to a wide range of example models. For each model, we have calculated the mean relative error, the maximum error and a "misclassification" rate of the results generated by our algorithm paired with each of the three edge ranking methods. We have also performed approximate inference using the loopy belief propagation algorithm as well as sample-based inference, and compared the results from these methods with the results from our algorithm.

For the pairwise models included in our numerical experiments, the quality of approximation when using our inference algorithm is typically lower than when using loopy belief propagation or sample-based inference. When the loopy belief propagation algorithm converges, there appears to be little reason to prefer our algorithm. Even when loopy belief propagation does not converge, sample-based inference with a moderate number of samples performs better than our algorithm on most tested pairwise models, with the exception of models with interaction parameters particularly close to zero. However, our algorithm may be useful in cases where a deterministic inference algorithm is required and the loopy belief propagation algorithm does not converge.

In higher-order models, our algorithm shows greater promise, performing on par with or better than the traditional approximate inference methods in many cases. In particular, combining our algorithm with the least fill-in or Kullback-Leibler based edge ranking methods results in approximations of high quality in such models. However, both our example higher-order models are regular grid models, so it is not necessarily the case that our model performs as well on less rigidly structured higher-order Markov random fields.

In terms of speed, our algorithm tends to be considerably slower than loopy belief propagation or sample-based inference with a moderate number of samples when applied to pairwise models. A major cause of this is the method by which we choose edges to deletion, and the algorithm is particularly slow when the Kullback-Leibler or least fill-in based edge ranking methods are used. However, it is unclear whether these methods are inherently slow or whether their poor performance in our numerical experiments is due to insufficiently optimized implementations.

8.1 Future work

We have used an edge deletion strategy which visits each junction tree clique in turn, deleting only one edge in each clique before proceeding to the next clique. This is not necessarily an optimal strategy, and experimenting with alternative edge deletion strategies could lead to better approximations. This also applies to the edge ranking methods used to choose which edge to delete within a clique. There is no guarantee that any of the edge ranking methods we have used are close to optimal. With regards to the Kullback-Leibler edge ranking method, more work could be done on alternative ways of calculating the divergence, as the Gibbs sampling based method used here has performed poorly.

The range of models tested in this text is limited, and mainly includes grid-structured Markov random fields with interactions of order one to four. It is not necessarily true that our algorithm performs poorly for all pairwise models beyond those which we have tested, or that it performs well for all higher-order models beyond those which we have tested. A natural extension of our work could be to apply our algorithm to new classes of models.

While we have only looked at marginal inference in this text, it would be simple to extend our method to the maximal a posteriori (MAP) problem, as this requires only an alteration of the junction tree algorithm. The "misclassification" rate calculated for the approximate marginal probabilities in this text could be used as a indication of which classes of model MAP inference using our algorithm could perform well on.

Finally, we have only concerned ourselves with *binary* Markov random fields, as Austad and Tjelmeland (2015) only developed the theory of the mean squared error energy approximation in the case of binary models. If possible, it would be interesting to extend the approximation to general Markov random fields, and see how our algorithm performs on non-binary models.

A Additional plots and data

In this appendix, we present plots and tables of data that were not essential to the exposition of the results in Chapter 7.

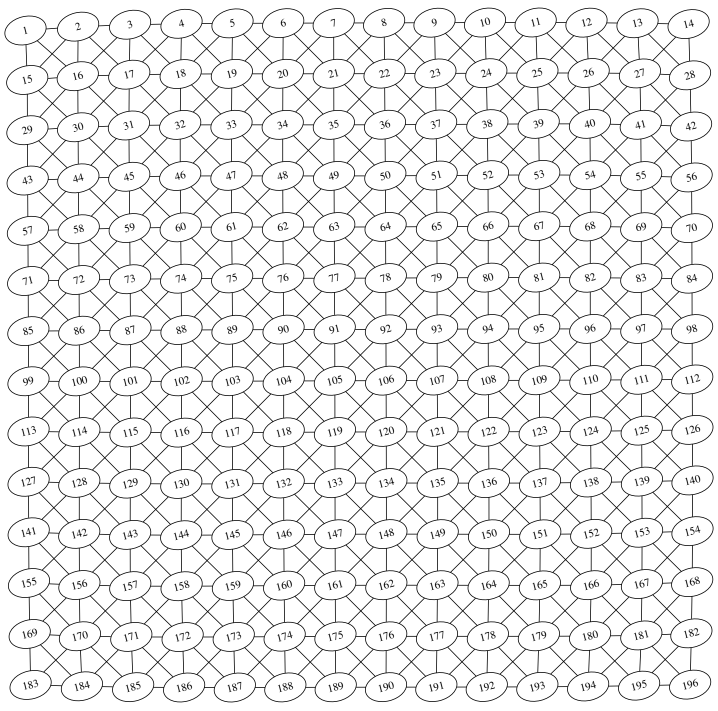


Figure A.1: The graph of the third-order Ising model analogue on an 14×14 grid used in the numerical experiments.

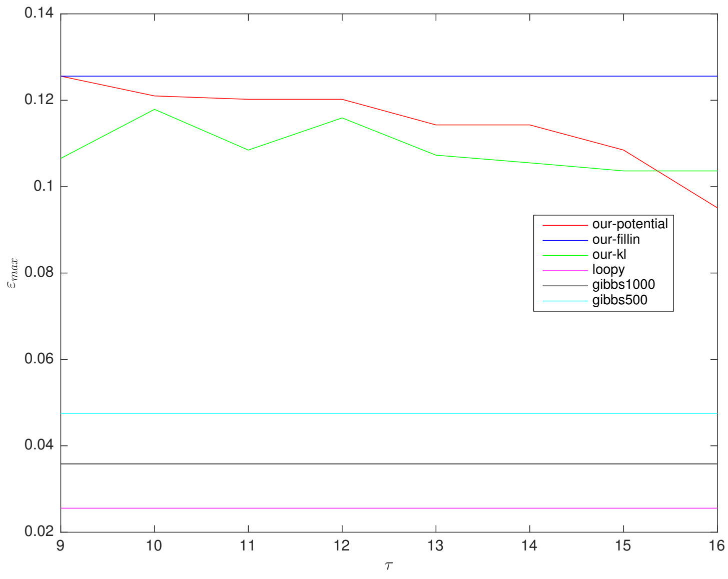


Figure A.2: Maximum error of the marginal probabilities for the biased 16 × 16 Ising model, as output by our approximate inference algorithm at different choices of τ and with different edge ranking methods. Also displayed are the mean relative errors of the marginal probabilities output by the loopy belief propagation algorithm (loopy) and sample-based inference based on 1000 (gibbs1000) and 500 (gibbs500) samples from a Gibbs samplingr with 200 burn-in samples. Note that the lines for all our edge selection methods overlap.

τ	$\lvert E_{del}\rvert$	ε_{rel}	ε_{max}	ε_{class}	time (s)
9	231	0.094 945	0.106 54	0	584.3
10	265	0.1071	0.117 89	0	721.55
11	225	0.093 148	0.108 47	0	712.36
12	235	0.096 416	0.115 91	0	601.52
13	164	0.075 308	0.107 29	0	640.11
14	153	0.059 62	0.105 51	0	513.24
15	158	0.058 426	0.103 64	0	511.11
16	128	0.049 18	0.103 64	0	517.01
17	0	0	0	0	—

Figure A.3: Results of applying our approximate inference algorithm to the biased 16 × 16 Ising model, using the Kullback-Leibler edge ranking method. Number of deleted edges, mean relative errors, maximum errors, misclassification rate and time taken to completion of calculations as a function of the parameter τ.

τ	$\lvert E_{del}\rvert$	ε_{rel}	ε_{max}	ε_{class}	time (s)
9	449	0.20599	0.12558	0	1152.5
10	447	0.20564	0.12558	0	1151.5
11	445	0.20407	0.12558	0	1150.9
12	443	0.20274	0.12558	0	1148.7
13	441	0.2014	0.12558	0	1148
14	439	0.20053	0.12558	0	1145.4
15	437	0.2	0.12558	0	1146.1
16	434	0.19832	0.12558	0	1149.5
17	0	0	0	0	—

Figure A.4: Results of applying our approximate inference algorithm to the biased 16×16 Ising model, using the least fill-in edge ranking method. Number of deleted edges, mean relative errors, maximum errors, misclassification rate and time taken to completion of calculations as a function of the parameter τ.

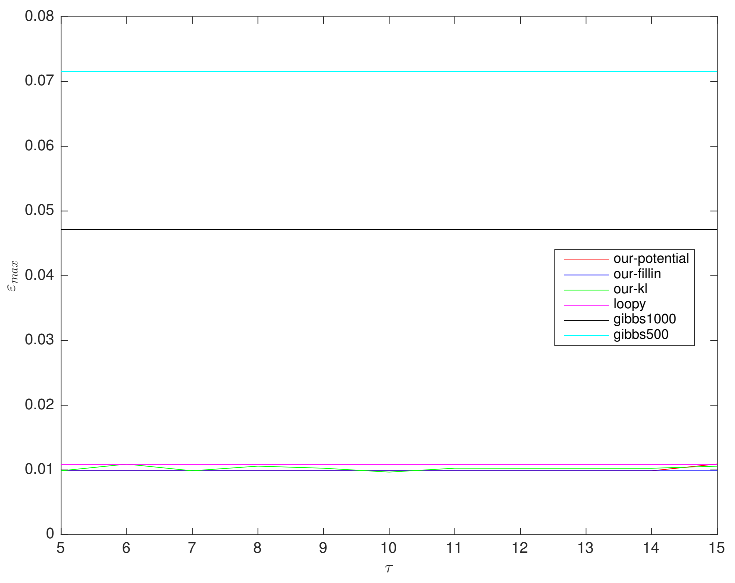


Figure A.5: Maximum error of the marginal probabilities for the unbiased 16×16 Ising model with $\beta = 0.05$, as output by our approximate inference algorithm at different choices of τ and with different edge ranking methods. Also displayed are the mean relative errors of the marginal probabilities output by the loopy belief propagation algorithm (`loopy`) and sample-based inference based on 1000 (`gibbs1000`) and 500 (`gibbs500`) samples from a Gibbs samplingr with 200 burn-in samples.

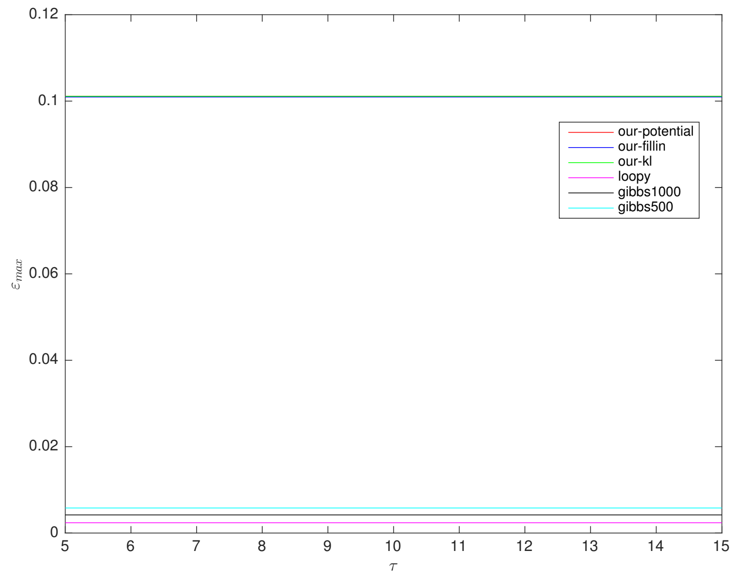


Figure A.6: Maximum error of the marginal probabilities for the unbiased 16×16 Ising model with $\beta = 2.0$, as output by our approximate inference algorithm at different choices of τ and with different edge ranking methods. Also displayed are the mean relative errors of the marginal probabilities output by the loopy belief propagation algorithm (`loopy`) and sample-based inference based on 1000 (`gibbs1000`) and 500 (`gibbs500`) samples from a Gibbs samplingr with 200 burn-in samples.

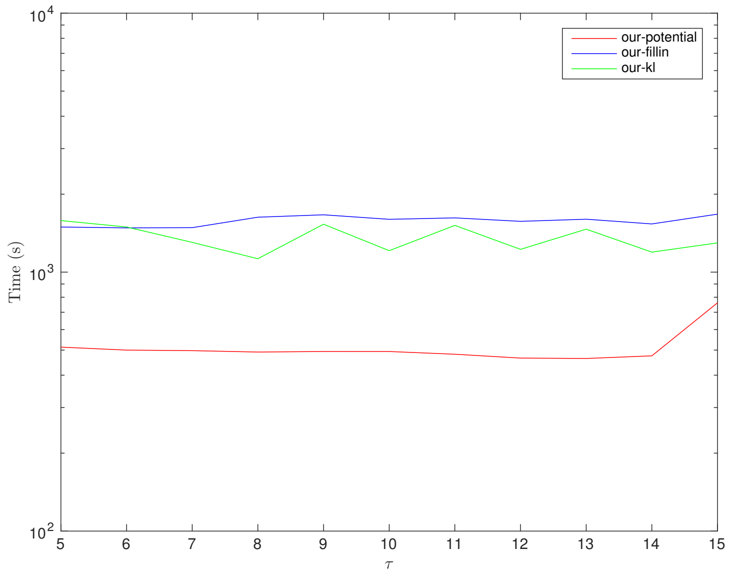


Figure A.7: Time taken to completion of the algorithm as a function of τ for the unbiased 16×16 Ising model with $\beta = 0.05$ and using each of the three edge ranking methods, plotted with a logarithmic scale on the vertical axis.

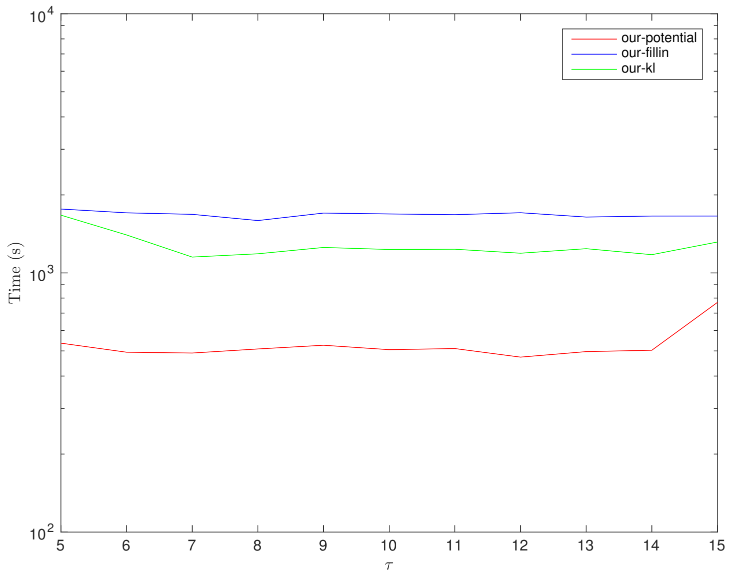


Figure A.8: Time taken to completion of the algorithm as a function of τ for the unbiased 16×16 Ising model with $\beta = 2.0$ and using each of the three edge ranking methods, plotted with a logarithmic scale on the vertical axis.

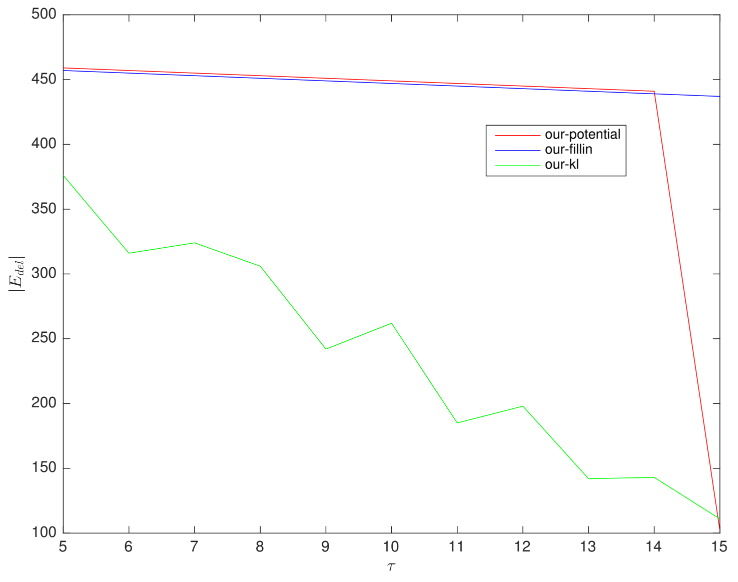


Figure A.9: Number of edges deleted as a function of τ, for the unbiased 16×16 Ising model with $\beta = 0.05$ and using each of the three edge ranking methods.

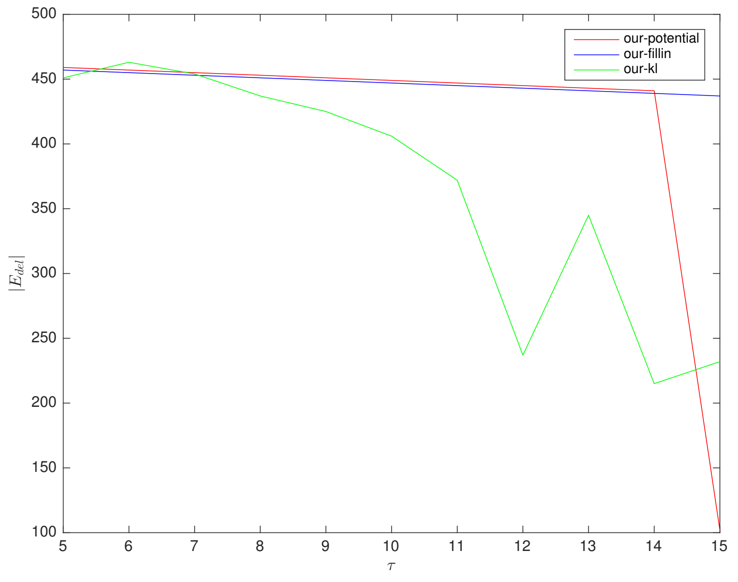


Figure A.10: Number of edges deleted as a function of τ, for the unbiased 16×16 Ising model with $\beta = 2.0$ and using each of the three edge ranking methods.

τ	$\lvert E_{del}\rvert$	ε_{rel}	ε_{max}	ε_{class}	time (s)
5	459	0.116 31	0.009 870 8	0	513.42
6	457	0.116 33	0.009 870 8	0	499.97
7	455	0.116 35	0.009 870 8	0	497.54
8	453	0.116 38	0.009 870 8	0	491.41
9	451	0.116 39	0.009 870 8	0	493.9
10	449	0.116 41	0.009 870 8	0	493.75
11	447	0.116 43	0.009 870 8	0	481.93
12	445	0.116 35	0.009 870 8	0	465.75
13	443	0.116 28	0.009 870 8	0	464.11
14	441	0.1164	0.009 870 8	0	475.11
15	102	0.112 04	0.010 923	0	761.29
17	0	0	0	0	—

Figure A.11: Results of applying our approximate inference algorithm to the unbiased 16×16 Ising model with $\beta = 0.05$, using the lowest potential edge ranking method. Number of deleted edges, mean relative errors, maximum errors, misclassification rate and time taken to completion of calculations as a function of the parameter τ.

τ	$\lvert E_{del}\rvert$	ε_{rel}	ε_{max}	ε_{class}	time (s)
5	376	0.1158	0.009 870 8	0	1581.7
6	316	0.114 67	0.010 903	0	1494.1
7	324	0.115 49	0.009 870 8	0	1303
8	306	0.116 14	0.010 591	0	1126.8
9	242	0.1148	0.010 268	0	1531.5
10	262	0.1148	0.009 670 8	0	1210.4
11	185	0.114 05	0.010 272	0	1515.3
12	198	0.112 99	0.010 264	0	1224.6
13	142	0.113 76	0.010 264	0	1465.4
14	143	0.114 37	0.010 26	0	1194.8
15	111	0.114 22	0.010 587	0	1297.1
17	0	0	0	0	—

Figure A.12: Results of applying our approximate inference algorithm to the unbiased 16×16 Ising model with $\beta = 0.05$, using the Kullback-Leibler based ranking method. Number of deleted edges, mean relative errors, maximum errors, misclassification rate and time taken to completion of calculations as a function of the parameter τ.

τ	$\lvert E_{del}\rvert$	ε_{rel}	ε_{max}	ε_{class}	time (s)
5	457	0.117 18	0.009 870 8	0	1493.4
6	455	0.117 15	0.009 870 8	0	1482.5
7	453	0.117 09	0.009 870 8	0	1485.3
8	451	0.117 05	0.009 870 8	0	1629.9
9	449	0.116 87	0.009 870 8	0	1665.1
10	447	0.116 69	0.009 870 8	0	1599.6
11	445	0.116 64	0.009 870 8	0	1620
12	443	0.116 57	0.009 870 8	0	1571
13	441	0.116 57	0.009 870 8	0	1601.1
14	439	0.116 55	0.009 870 8	0	1535.3
15	437	0.116 47	0.009 870 8	0	1674.6
17	0	0	0	0	—

Figure A.13: Results of applying our approximate inference algorithm to the unbiased 16×16 Ising model with $\beta = 0.05$, using the fill-in based ranking method. Number of deleted edges, mean relative errors, maximum errors, misclassification rate and time taken to completion of calculations as a function of the parameter τ.

τ	$\lvert E_{del}\rvert$	ε_{rel}	ε_{max}	ε_{class}	time (s)
5	459	0.047 736	0.1011	0	535.88
6	457	0.047 59	0.1011	0	494.43
7	455	0.047 444	0.1011	0	490.83
8	453	0.047 298	0.1011	0	508.9
9	451	0.047 152	0.1011	0	525.99
10	449	0.047 006	0.1011	0	505.76
11	447	0.046 86	0.1011	0	510.72
12	445	0.046 714	0.1011	0	473.33
13	443	0.046 567	0.1011	0	497.04
14	441	0.046 401	0.1011	0	503.18
15	102	0.020 079	0.1011	0	770.01
17	0	0	0	0	—

Figure A.14: Results of applying our approximate inference algorithm to the unbiased 16×16 Ising model with $\beta = 2.0$, using the lowest potential edge ranking method. Number of deleted edges, mean relative errors, maximum errors, misclassification rate and time taken to completion of calculations as a function of the parameter τ.

τ	$\lvert E_{del} \rvert$	ε_{rel}	ε_{max}	ε_{class}	time (s)
5	451	0.046 324	0.1011	0	1671.1
6	463	0.047 842	0.1011	0	1401.5
7	454	0.046 981	0.1011	0	1151.4
8	437	0.043 518	0.1011	0	1185.5
9	425	0.044 345	0.1011	0	1254
10	406	0.039 976	0.1011	0	1231.7
11	372	0.036 814	0.1011	0	1234.4
12	237	0.021 072	0.1011	0	1191.5
13	345	0.032 916	0.1011	0	1240.4
14	215	0.021 191	0.1011	0	1176.1
15	232	0.020 962	0.1011	0	1316.3
17	0	0	0	0	—

Figure A.15: Results of applying our approximate inference algorithm to the unbiased 16×16 Ising model with $\beta = 2.0$, using the Kullback-Leibler based ranking method. Number of deleted edges, mean relative errors, maximum errors, misclassification rate and time taken to completion of calculations as a function of the parameter τ.

τ	$\lvert E_{del}\rvert$	ε_{rel}	ε_{max}	ε_{class}	time (s)
5	457	0.043 359	0.100 95	0	1764.3
6	455	0.043 19	0.100 95	0	1705.8
7	453	0.043 02	0.100 95	0	1683.1
8	451	0.042 851	0.100 95	0	1592.7
9	449	0.042 682	0.100 95	0	1702.2
10	447	0.042 512	0.100 95	0	1688.9
11	445	0.042 341	0.100 95	0	1678
12	443	0.042 171	0.100 95	0	1707.3
13	441	0.042 002	0.100 95	0	1642.9
14	439	0.041 841	0.100 95	0	1657.8
15	437	0.041 673	0.100 95	0	1658
17	0	0	0	0	—

Figure A.16: Results of applying our approximate inference algorithm to the unbiased 16×16 Ising model with $\beta = 2.0$, using the fill-in based ranking method. Number of deleted edges, mean relative errors, maximum errors, misclassification rate and time taken to completion of calculations as a function of the parameter τ.

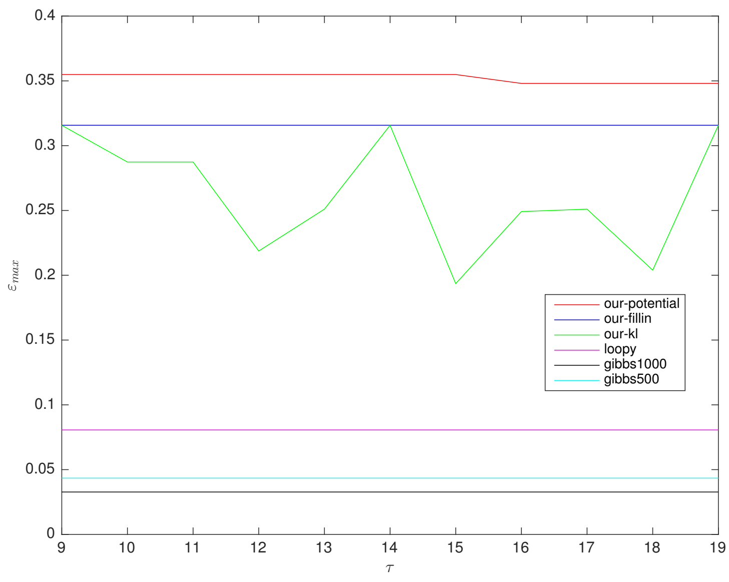


Figure A.17: Maximum error of the marginal probabilities output by our approximate inference algorithm at different choices of τ, for the 204 edge Boltzmann machine, compared to the usual alternative approximate inference algorithms.

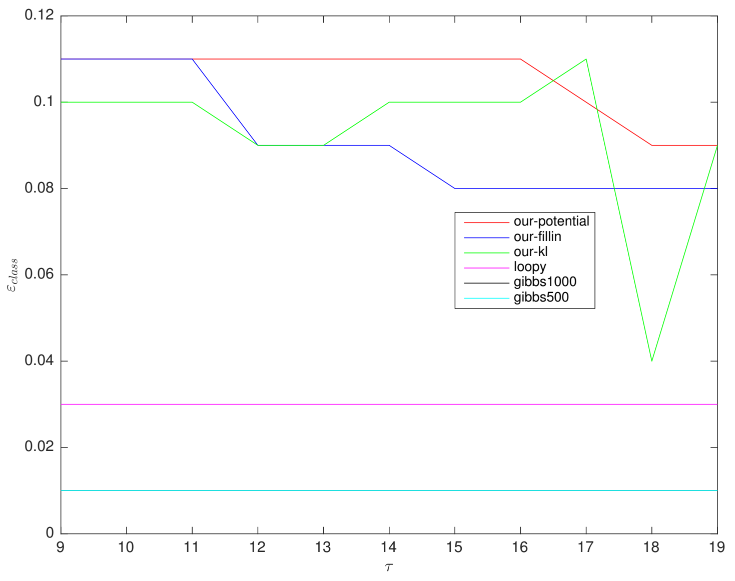


Figure A.18: Misclassification error of the marginal probabilities output by our approximate inference algorithm at different choices of τ, for the 204 edge Boltzmann machine, compared to the usual alternative approximate inference algorithms.

τ	$\lvert E_{del} \rvert$	ε_{rel}	ε_{max}	ε_{class}	time (s)
9	162	0.810 25	0.3549	0.11	13.417
10	162	0.810 25	0.3549	0.11	12.745
11	162	0.810 25	0.3549	0.11	12.685
12	157	0.880 36	0.3549	0.11	12.787
13	157	0.880 36	0.3549	0.11	12.592
14	156	0.890 87	0.3549	0.11	13.1
15	146	0.7387	0.3549	0.11	13.838
16	142	0.733 37	0.348 07	0.11	15.599
17	139	0.733 61	0.348 07	0.1	15.355
18	129	0.685 85	0.348 07	0.09	75.679
19	129	0.685 85	0.348 07	0.09	73.717
47	0	0	0	0	—

Figure A.19: Results of applying our approximate inference algorithm to the 204 edge Boltzmann machine, using the lowest potential edge ranking method. Number of deleted edges, mean relative errors, maximum errors, misclassification rate and time taken to completion of calculations as a function of the parameter τ.

τ	$\lvert E_{del} \rvert$	ε_{rel}	ε_{max}	ε_{class}	time (s)
9	163	0.793 79	0.315 75	0.1	114.35
10	160	1.0712	0.2874	0.1	110.36
11	161	0.756 33	0.2874	0.1	103.36
12	154	0.744 23	0.218 73	0.09	103.56
13	144	1.0551	0.251 02	0.09	104.89
14	155	0.972 02	0.315 75	0.1	101.26
15	148	0.687 78	0.193 56	0.1	103.71
16	145	1.0399	0.249 16	0.1	121.13
17	143	0.786 43	0.251 02	0.11	121.26
18	116	1.0078	0.203 95	0.04	190.51
19	140	0.948 49	0.315 75	0.09	148.89
47	0	0	0	0	—

Figure A.20: Results of applying our approximate inference algorithm to the 204 edge Boltzmann machine, using the Kullback-Leibler based edge ranking method. Number of deleted edges, mean relative errors, maximum errors, misclassification rate and time taken to completion of calculations as a function of the parameter τ. Gibbs sampling-based inference based on 30000 iterations was used as the reference.

τ	$\lvert E_{del} \rvert$	ε_{rel}	ε_{max}	ε_{class}	time (s)
9	158	0.904 42	0.315 75	0.11	41.819
10	156	0.894 57	0.315 75	0.11	41.061
11	154	0.895 39	0.315 75	0.11	40.281
12	149	0.773 66	0.315 75	0.09	40.791
13	149	0.773 66	0.315 75	0.09	40.202
14	145	0.779 27	0.315 75	0.09	40.032
15	136	0.797 65	0.315 75	0.08	39.223
16	130	0.778 92	0.315 75	0.08	44.84
17	122	0.784 29	0.315 75	0.08	50.188
18	122	0.784 29	0.315 75	0.08	49.912
19	116	0.771 08	0.315 75	0.08	80.634
47	0	0	0	0	—

Figure A.21: Results of applying our approximate inference algorithm to the 204 edge Boltzmann machine, using the least fill-in based edge ranking method. Number of deleted edges, mean relative errors, maximum errors, misclassification rate and time taken to completion of calculations as a function of the parameter τ. Gibbs sampling-based inference based on 30000 iterations was used as the reference.

τ	$\|E_{del}\|$	ε_{rel}	ε_{max}	ε_{class}	time (s)
9	479	0.701 86	0.508 28	0.15	42.864
10	468	0.696 64	0.508 28	0.15	42.138
11	463	0.686 69	0.508 28	0.15	41.752
12	462	0.686 06	0.508 28	0.15	42.026
13	460	0.684 69	0.508 28	0.15	41.902
14	456	0.648 92	0.508 28	0.15	43.838
15	455	0.648 21	0.508 28	0.15	46.18
16	447	0.634 85	0.508 28	0.14	51.968
17	445	0.635 47	0.508 28	0.14	64.432
18	425	0.625 76	0.508 28	0.14	114.89
19	425	0.625 76	0.508 28	0.14	113.72
67	0	0	0	0	—

Figure A.22: Results of applying our approximate inference algorithm to the 500 edge Boltzmann machine, using the lowest potential edge ranking method. Number of deleted edges, mean relative errors, maximum errors, misclassification rate and time taken to completion of calculations as a function of the parameter τ. Gibbs sampling-based inference based on 30000 iterations was used as the reference.

τ	$\|E_{del}\|$	ε_{rel}	ε_{max}	ε_{class}	time (s)
9	464	0.628 65	0.508 28	0.12	950.03
10	455	0.627 01	0.508 28	0.13	865.45
11	458	0.611 28	0.508 28	0.12	846.66
12	450	0.601 56	0.508 28	0.11	845.86
13	454	0.6245	0.481 88	0.13	808.9
14	452	0.612 91	0.508 28	0.11	704.5
15	451	0.620 86	0.495 02	0.12	810.73
16	441	0.613 36	0.489 93	0.12	741.58
17	450	0.5996	0.508 28	0.11	717.54
18	448	0.580 92	0.480 13	0.12	825.93
19	440	0.595 64	0.489 92	0.11	782.66
67	0	0	0	0	—

Figure A.23: Results of applying our approximate inference algorithm to the 500 edge Boltzmann machine, using the Kullback-Leibler based edge ranking method. Number of deleted edges, mean relative errors, maximum errors, misclassification rate and time taken to completion of calculations as a function of the parameter τ. Gibbs sampling-based inference based on 30000 iterations was used as the reference.

τ	$\lvert E_{del}\rvert$	ε_{rel}	ε_{max}	ε_{class}	time (s)
9	451	0.662 46	0.508 28	0.12	191.88
10	451	0.662 46	0.508 28	0.12	190.91
11	448	0.654 99	0.508 28	0.12	191.84
12	437	0.644 46	0.508 28	0.12	191.19
13	436	0.651 07	0.508 28	0.13	191.42
14	432	0.652 71	0.508 28	0.13	193.23
15	429	0.651 63	0.508 28	0.12	190.94
16	429	0.651 63	0.508 28	0.12	191.04
17	428	0.651 25	0.508 28	0.12	239.9
18	426	0.647 88	0.508 28	0.12	272.46
19	426	0.647 88	0.508 28	0.12	270.72
67	0	0	0	0	—

Figure A.24: Results of applying our approximate inference algorithm to the 500 edge Boltzmann machine, using the least fill-in based edge ranking method. Number of deleted edges, mean relative errors, maximum errors, misclassification rate and time taken to completion of calculations as a function of the parameter τ. Gibbs sampling-based inference based on 30000 iterations was used as the reference.

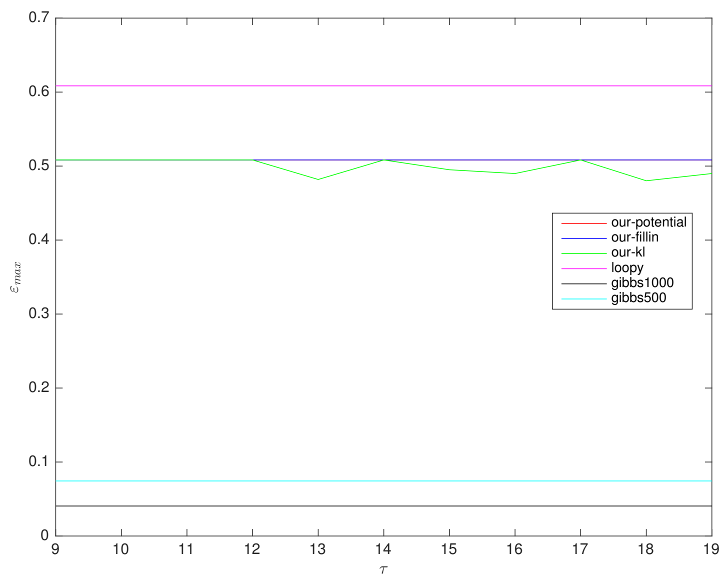


Figure A.25: Maximum error of the marginal probabilities output by our approximate inference algorithm at different choices of τ, for the 500 edge Boltzmann machine, compared to the usual alternative approximate inference algorithms.

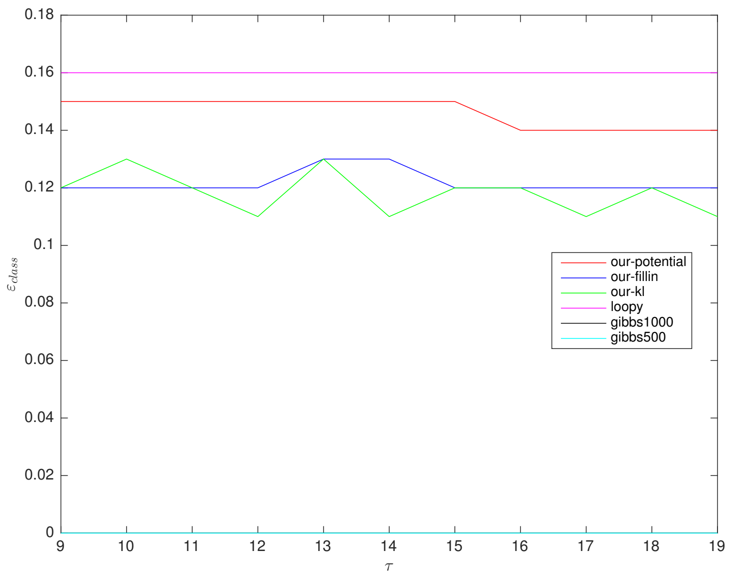


Figure A.26: Misclassification error of the marginal probabilities output by our approximate inference algorithm at different choices of τ, for the 500 edge Boltzmann machine, compared to the usual alternative approximate inference algorithms.

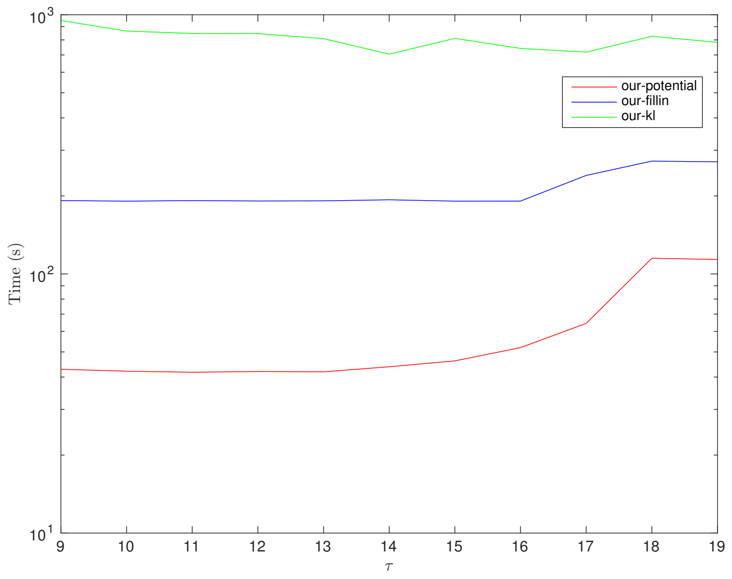


Figure A.27: Time taken to completion of the algorithm as a function of τ for the 500 edge Boltzmann machine and using each of the three edge ranking methods, plotted with a logarithmic scale on the vertical axis.

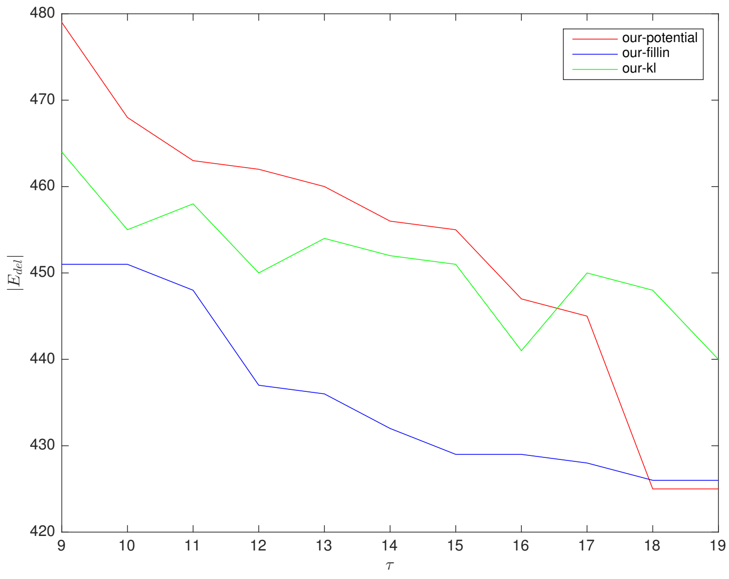


Figure A.28: Number of edges deleted as a function of τ, for the 500 edge Boltzmann machine while using each of the three edge ranking methods.

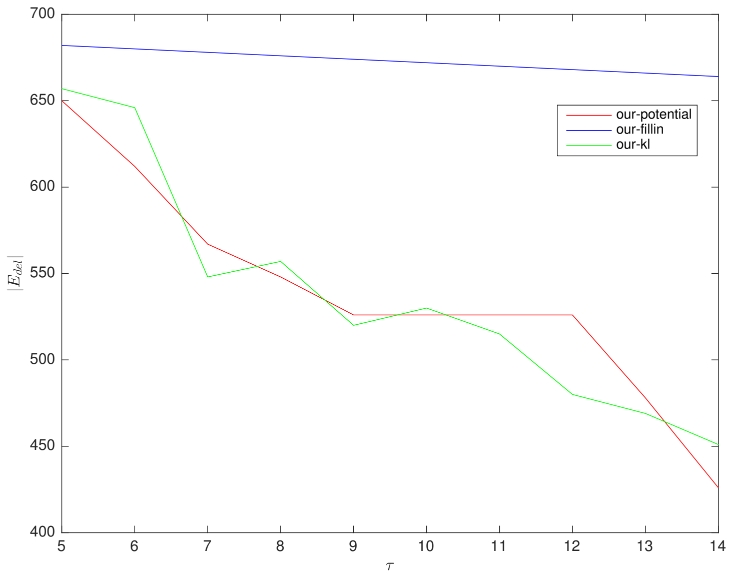


Figure A.29: Number of edges deleted as a function of τ, for the 14×14 third order grid model and each of the three edge ranking methods.

τ	$\lvert E_{del} \rvert$	ε_{rel}	ε_{max}	ε_{class}	time (s)
5	650	0.071 18	0.218 22	0	364.47
6	612	0.068 858	0.218 22	0	353.98
7	567	0.066 193	0.218 22	0	315.96
8	548	0.065 069	0.218 22	0	310.9
9	526	0.063 879	0.218 22	0	331.79
10	526	0.063 879	0.218 22	0	310.92
11	526	0.063 879	0.218 22	0	316.87
12	526	0.063 879	0.218 22	0	327.21
13	478	0.060 576	0.218 22	0	298.5
14	426	0.057 028	0.218 22	0	287.51
15	367	0.053 37	0.218 22	0	360.22
16	0	0	0	0	—

Figure A.30: Results of applying our approximate inference algorithm to the 14×14 third-order grid model, using the lowest potential edge ranking method. Number of deleted edges, mean relative errors, maximum errors, misclassification rate and time taken to completion of calculations as a function of the parameter τ. Gibbs sampling-based inference based on 30000 iterations was used as the reference.

τ	$\lvert E_{del} \rvert$	ε_{rel}	ε_{max}	ε_{class}	time (s)
5	657	0.068 39	0.218 22	0	13365
6	646	0.066 031	0.218 22	0	13087
7	548	0.054 169	0.218 22	0	13049
8	557	0.054 934	0.218 22	0	12615
9	520	0.051 091	0.218 22	0	13011
10	530	0.052 854	0.218 22	0	13361
11	515	0.048 937	0.218 22	0	13033
12	480	0.047 947	0.218 22	0	12873
13	469	0.046 923	0.218 22	0	13131
14	451	0.045 355	0.218 22	0	12402
16	0	0	0	0	—

Figure A.31: Results of applying our approximate inference algorithm to the 14×14 third-order grid model, using the Kullback-Leibler based edge ranking method. Number of deleted edges, mean relative errors, maximum errors, misclassification rate and time taken to completion of calculations as a function of the parameter τ. Gibbs sampling-based inference based on 30000 iterations was used as the reference.

τ	$\lvert E_{del}\rvert$	ε_{rel}	ε_{max}	ε_{class}	time (s)
5	682	0.065 998	0.186 48	0	763.33
6	680	0.065 592	0.186 48	0	769.86
7	678	0.065 259	0.186 48	0	788.49
8	676	0.065 027	0.186 48	0	773.48
9	674	0.064 902	0.186 48	0	782.08
10	672	0.064 721	0.186 48	0	810.07
11	670	0.064 376	0.186 48	0	766.08
12	668	0.064 139	0.186 48	0	789.82
13	666	0.063 826	0.186 48	0	749.7
14	664	0.063 585	0.186 48	0	761.55
16	0	0	0	0	—

Figure A.32: Results of applying our approximate inference algorithm to the 14×14 third-order grid model, using the least fill-in edge ranking method. Number of deleted edges, mean relative errors, maximum errors, misclassification rate and time taken to completion of calculations as a function of the parameter τ. Gibbs sampling-based inference based on 30000 iterations was used as the reference.

Bibliography

Ajroud, A., M. N. Omri, H. Youssef, and S. Benferhat (2012). “Loopy belief propagation in Bayesian networks: Origin and possibilistic perspectives”. In: *CoRR* abs/1206.0976 (cit. on p. 40).

Arnborg, S., D. G. Corneil, and A. Proskurowski (1987). “Complexity of finding embeddings in a k-tree”. In: *SIAM Journal on Algebraic Discrete Methods* 8.2, pp. 277–284. doi: 10.1137/0608024 (cit. on p. 8).

Austad, H. M. and H. Tjelmeland (2015). “Approximations and bounds for binary Markov random fields”. In: *ArXiv e-prints*. arXiv: 1501.07414 [stat.CO] (cit. on pp. 1, 2, 14, 34–36, 48, 50, 51, 72).

Boros, E. and P. L. Hammer (2002). “Pseudo-boolean optimization”. In: *Discrete Appl. Math.* 123.1-3, pp. 155–225. doi: 10.1016/S0166-218X(01)00341-9 (cit. on p. 14).

Budka, M., B. Gabrys, and K. Musial (2011). “On accuracy of PDF divergence estimators and their applicability to representative data sampling”. In: *Entropy* 13.7, p. 1229 (cit. on p. 42).

Cano, A. and S. Moral (1995). “Heuristic algorithms for the triangulation of graphs”. In: *Selected Papers from the 5th International Conference on Processing and Management of Uncertainty in Knowledge-Based Systems, Advances in Intelligent Computing*. IPMU’94. London, UK, UK: Springer-Verlag, pp. 98–107. isbn: 3-540-60116-3 (cit. on p. 23).

Elidan, G., I. McGraw, and D. Koller (2006). “Residual belief propagation: Informed scheduling for asynchronous message passing”. In: *Proceedings of the twenty-second conference on uncertainty in AI (UAI)*. Boston, Massachussetts (cit. on p. 28).

Hammersley, J. M. and P. Clifford (1971). *Markov fields on finite graphs and lattices*. http://www.statslab.cam.ac.uk/~grg/books/hammfest/hamm-cliff.pdf (cit. on p. 10).

Heggernes, P. (2006). “Minimal triangulations of graphs: A survey”. In: *Discrete Mathematics* 306.3. Minimal Separation and Minimal Triangulation, pp. 297 –317. doi: http://dx.doi.org/10.1016/j.disc.2005.12.003 (cit. on p. 24).

Jensen, F. and S. K. Anderson (1990). “Approximations in Bayesian belief universe for knowledge based systems”. In: *Proceedings of the sixth conference on uncertainty in artificial intelligence (UAI)*, pp. 162–169 (cit. on p. 1).

Khosla, M. (2009). “Message passing algorithms”. MA thesis. Universität des Saarlandes (cit. on p. 20).

Kjærulff, U. (1994). “Reduction of computational complexity in Bayesian networks through removal of weak dependencies”. In: *Proceedings of the tenth conference on uncertainty in artificial intelligence (UAI)*, pp. 374–382 (cit. on p. 1).

Koller, D. and N. Friedman (2009). *Probabilistic graphical models: Principles and techniques*. MIT Press (cit. on pp. 3, 19, 24, 25, 29–31).

Kruskal, J. B. (1956). “On the shortest spanning subtree of a graph and the traveling salesman problem”. In: *Proceedings of the American Mathematical Society* 7.1, pp. 48–50 (cit. on p. 25).

Murphy, K. P., Y. Weiss, and M. I. Jordan (1999). “Loopy belief propagation for approximate inference: An empirical study”. In: *Proceedings of the fifteenth conference on uncertainty in artificial intelligence (UAI)*. UAI’99. Morgan Kaufmann Publishers Inc., pp. 467–475 (cit. on p. 28).

Murray, I. and Z. Ghahramani (2004). “Bayesian learning in undirected graphical models: Approximate MCMC algorithms”. In: (cit. on pp. 46, 51, 58).

Pattabiraman, B., M. Patwary, A. Gebremedhin, W.-k. Liao, and A. Choudhary (2013). “Fast algorithms for the maximum clique problem on massive sparse graphs”. English. In: *Algorithms and models for the web graph*. Ed. by A. Bonato, M. Mitzenmacher, and P. Prałat. Vol. 8305. Lecture Notes in Computer Science. Springer International Publishing, pp. 156–169. doi: 10.1007/978-3-319-03536-9_13 (cit. on p. 4).

Prim, R. C. (1957). “Shortest connection networks and some generalizations”. In: *Bell System Technical Journal* 36.6, pp. 1389–1401. doi: 10.1002/j.1538-7305.1957.tb01515.x (cit. on p. 25).

Riedel, S., D. Smith, and A. McCallum (2010). “Inference by minimizing Size, divergence, or their sum”. In: *Proceedings of the 26th Annual Conference on Uncertainty in AI (UAI ’10)* (cit. on pp. 1, 2).

Riggan, W. B. et al. (1987). *U.S. cancer mortality rates and trends, 1950-1979*. Vol. 4. US Environmental Protection Agency (cit. on p. 48).

Schmidt, M. (2007). *UGM: Matlab code for undirected graphical models*. https://www.cs.ubc.ca/~schmidtm/Software/UGM.html (cit. on pp. 49, 50).

Shannon, C. (1948). “A mathematical theory of communication”. In: *Bell System Technical Journal, The* 27.3, pp. 379–423. doi: 10.1002/j.1538-7305.1948.tb01338.x (cit. on p. 13).

Sinoquet, C. and R. Mourad (2014). *Probabilistic graphical models for genetics, genomics, and postgenomics*. Oxford University Press (cit. on p. 26).

Stoop, N., T. Ott, and R. Stoop (2006). “Loopy belief propagation: Benefits and pitfalls on Ising-like systems”.

In: *International symposium on nonlinear theory and its applications*. Bologna: International symposium on nonlinear theory and its applications (cit. on p. 28).

Sutton, C. A. and A. McCallum (2007). "Improved dynamic schedules for belief propagation". In: *Conference on uncertainty in AI (UAI)* (cit. on p. 28).

T. S. Levitt L. N. Kanal, J. F. L. and R. D. Shachter (2014). *Uncertainty in artificial intelligence 4*. Elsevier, pp. 169–198 (cit. on p. 25).

Thornton, J. A. (2003). "Approximate inference of Bayesian networks through edge deletion". MA thesis. Kansas State University (cit. on p. 1).

Tjelmeland, H. and J. Besag (1998). "Markov random fields with higher-order interactions". In: *Scandinavian Journal of Statistics* 25.3, pp. 415–433. doi: 10.1111/1467-9469.00113 (cit. on p. 47).