Learning Symmetric Relational Markov Random Fields

A thesis submitted in partial fulfillment of the
requirements for the degree of Master of Science

by

Ofer Meshi

Supervised by Prof. Nir Friedman

December 2007

The School of Computer Science and Engineering
The Hebrew University of Jerusalem, Israel

Abstract

Relational Markov Random Fields (rMRF's) are a general and flexible framework for reasoning about the joint distribution over attributes of a large number of interacting entities, such as graphs, social networks or gene networks. When modeling such a network using an rMRF one of the major problems is choosing the set of features to include in the model and setting their weights. The main computational difficulty in learning such models from evidence is that the estimation of each set of features requires the use of a parameter estimation procedure. Even when dealing with complete data, where one can summarize a large domain by sufficient statistics, parameter estimation requires one to compute the expectation of the sufficient statistics given different parameter choices. This means that we run inference in the network for each step in the iterative algorithm used for parameter estimation. Since exact inference is usually intractable, the typical solution to this problem is to resort to approximate inference procedures, such as *loopy belief propagation*. Although these procedures are quite efficient, they still require computation that is on the order of the number of interactions (or features) in the model. When learning a large relational model over a complex domain even such approximations require unrealistic running time.

In this work we show that for a particular class of rMRFs, which have inherent symmetry, we can perform the inference needed for learning procedures using a lifted *template-level* belief propagation. This procedure's running time is proportional to the size of the relational model rather than the size of the domain. Moreover, we show that this computational procedure is equivalent to synchronous loopy belief propagation. This yields a dramatic speedup in inference time. We use this speedup to learn such symmetric rMRF's from evidence in an efficient way. This enables us to explore problem domains which were impossible to handle with existing methods.

Chapter 1

Introduction

1. Motivation

Complex networks are ubiquitous in many fields of science. Deciphering the network of interactions underlying the functionality of systems as a whole is a great challenge. If we succeed in doing so, then we might gain new insights to the behavior of such complex systems and better understand how individual nodes interact to perform complex tasks. This challenge is common to a plethora of domains including protein interaction networks (Figure 1.1), the Web, social networks, gene networks, power grids, information processing networks, and many more. One approach in this field attempts to find local rules of interactions between relatively small units that govern the global structure of the network. One of the main problems in handling such networks is their usually huge size. For example, in the *Protein-Protein interaction* network of budding yeast there are ~ 6000 proteins with $\sim 18,000,000$ possible interactions.

A notable work that tries to meet this challenge is *Network Motifs* by Milo *et al.* [Milo et al. 2002]. They search the network for basic units called motifs, which are over-represented subgraphs. To determine which subgraphs are over-represented the abundance of each subgraph in the real network is compared to its abundance in a random ensemble of networks. Other approaches to this problem emphasize the importance of measurable quantities of the network, such as the degrees of individual nodes [Barabasi and Albert 1999], shortest paths between nodes, and others. Each of these methods has its shortcomings (we give some more details in Chapter 7).

We take a different approach that uses a *Probabilistic Graphical Model* in order to model the complex network of interactions (see also [Jaimovich et al. 2006]). This is a *generative* approach in which we learn a model that describes the complex network at hand. We believe this approach is more elegant and overcomes some of the limitations of existing methods. More specifically, we use a sub-class of graphical models called *relational Markov Random Fields* (rMRFs) which are suitable for reasoning about complex networks of interactions. This framework is natural for describing complex relations between entities, in which the same local rules repeat throughout the model. In practice, such probabilistic models give a compact representation of the joint distribution of random

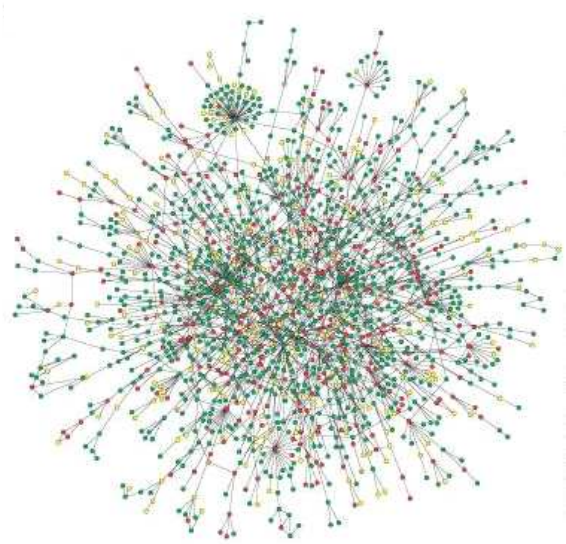


Figure 1.1: Example for an protein interaction network. Adopted from H Jeong et.al. Nature 411,41 2001

variables that describe properties of entities and the interactions between them. This representation assumes that the overall joint distribution can be described in terms of local, small joint distributions over groups of random variables in the model. Hence it is natural to use rMRFs in order to look for local rules governing the global properties of the network.

In the remaining of this chapter we explain about Markov Random Fields and rMRFs, then we show why running probabilistic inference on such models is both essential for our goal and difficult, and finally we talk about our contribution and related work.

2. Markov Random Fields

Markov Random Fields (MRFs), also known as *Markov Networks*, are a general way to model the joint probability of a group of random variables $\mathcal{X} = \{X_1, ..., X_n\}$. Such models were first introduced in the field of statistical mechanics to model certain physical phenomena and today they are used in a wide range of applications including computer vision, natural language, computational biology and digital communications. MRFs provide a compact representation of the distribution in terms of local *potentials* or *factors* which are defined over subsets of variables. These potential functions are defined as $\pi_c : x_c \rightarrow I\!R$ and can be viewed as representing preferences over local configurations (*not* to be confused with marginal probabilities). Such compact representation is achieved by *factorizing* the joint distribution into a product of the local factors. Of course, not every distribution can be factorized this way, but this family is still very expressive. An MRF is strongly related to an undirected graph $\mathcal{G} = (V, E)$ where each vertex $v \in V$ is associated with a random variable $X_i \in \mathcal{X}$ and each factor π is associated with a maximal clique c in $\mathcal{G}$.

Formally, the joint distribution represented by an MRF is given by:

$$P(\mathcal{X} = x) = \frac{1}{Z} \prod_c \pi_c(x_c) \tag{1.1}$$

where Z is a normalization constant known as the *Partition Function* and defined by:

$$Z = \sum_{x \in \Omega} \prod_c \pi_c(x_c)$$

where Ω is the set of all legal assignments to $\mathcal{X}$. This constant ensures that the model describes a legal probability distribution - all entries sum to 1.

A distribution of this form is called a *Gibbs distribution*.

It is often more convenient to specify potentials slightly differently and use the representation of a log-linear model:

$$P(\mathcal{X} = x) = \frac{1}{Z} \exp\left\{\sum_i \theta_i f_i(x)\right\} \tag{1.2}$$

and:

$$Z = \sum_{x \in \Omega} \exp\left\{\sum_i \theta_i f_i(x)\right\} \tag{1.3}$$

The log-linear model has a set of local feature functions $f : \Omega \rightarrow I\!R$ for each clique, and the parameters of the log-linear model are weights θ such that each $\theta_i \in I\!R$ corresponds to a feature f_i. If, for example, the potentials are represented as tabular CPDs, then each entry in the table is associated with one feature f_j and its value is exactly θ_j.

There exists an important connection between the structure of the undirected graph $\mathcal{G}$ and conditional independence in the probability distribution defined by the random field. Specifically, each group of variables $\mathbf{X} \subseteq \mathcal{X}$ is independent of all other variables $\{X_i \in \mathcal{X} \setminus \mathbf{X}\}$ given their neighbors in the graph - also called their *Markov blanket*. Formally: $(\mathbf{X} \perp \mathcal{X} \setminus \mathbf{X} | N_{\mathbf{X}})$, and we say that $\mathcal{X}$ is *locally Markov* with respect to $\mathcal{G}$. The Hammersley-Clifford theorem [Hammersley and Clifford 1971] states that if $\forall X_i \; p(X_i = x_i) > 0$ and $\mathcal{X}$ is locally Markov with respect to G, then $p(\mathcal{X})$ factorizes with respect to G (Eq. (1.1)). And the other direction is also true.

In this work we focus on MRFs for structured domains, that are naturally represented under the Entity-Relation paradigm [Getoor et al. 2001; Friedman et al. 1999]. These *Probabilistic Relational Models* (PRMs), also called *Template Models*, specify a recipe with which a concrete MRF can be constructed for a specific set of entities. Such relational MRFs (rMRFs) may reuse the same potential function for many factors in the instantiated model. This means that the model uses *shared parameters* that allow reasoning about a set of variables as a group. rMRFs are used to model many types of domains like the web [Taskar et al. 2004], gene expression measurements [Segal et al. 2003] and protein-protein

interaction networks [Jaimovich et al. 2006]. In these domains, they can be used for diverse tasks, such as prediction of missing values given some observations [Jaimovich et al. 2006], classification [Taskar et al. 2004], and model selection [Segal et al. 2003]. All of these tasks require the ability to perform inference in these models. In this work we build on the fact that such models contain many repetitions of the same local structure in the instantiated level. We use it to devise an extremely efficient approximate inference algorithm that takes advantage of this symmetry. Furthermore, we use the same property to perform model selection tasks much more efficiently.

3. Model Selection

In the task of learning an MRF from empirical evidence we are given a set of training samples $\boldsymbol{D} = \{x[1], ..., x[M]\}$, each is an assignment to the variables $\mathcal{X}$ (In this work we focus on the case of fully observed data, which means that in each sample values are assigned to all the variables in $\mathcal{X}$). Our goal is to learn an appropriate set of features $F = \{f_1, ..., f_k\}$ (*Feature Selection*) and their corresponding parameters $\theta = \{\theta_1, ..., \theta_k\}$ (*Parameter Estimation*). In other words, we want to construct the best generative model for the given evidence. This task turns out to be very difficult as the number of the feature sets we have to consider is usually prohibitively large, and even if we have the correct set of features finding values for their parameters cannot be done effectively in general [Parise and Welling 2005]. Instead, we normally have to resort to iterative methods for optimizing over parameter space. Unfortunately, every step in the iterative algorithm requires that we run inference on the model. So, inference turns out to be the main computational bottleneck in the learning procedure.

4. Inference in MRFs

Inference in MRFs is the computation needed to answer probabilistic queries about the joint distribution defined by the model. Notable queries include finding the marginal probability or the most probable assignment of a subset of the variables (possibly given the values of other variables). A naive solution to such queries is achieved by summing over some (or all) of the possible assignments, which generally requires computational time that is exponential in the number of variables. This makes exact inference infeasible in most interesting cases. In fact, the problem of inference in such probabilistic models is $\#P - complete$. Instead, a common practice is to trade-off accuracy for feasibility and resort to approximate inference methods.

One approach to the design of approximate inference uses instantiations to all or some of the variables. This approach involves a stochastic process, such as *Markov Chain Monte Carlo* (MCMC) [Geman and Geman 1984], to produce the instantiations, from which the joint distribution can be approximated. In another approach to approximate inference, termed *Variational Methods* [Jordan et al. 1998], we attempt to approximate the target distribution P by a simpler distribution Q. In practice we define a family of simpler distri-

butions $\mathcal{Q}$ and look for a particular instance $Q \in \mathcal{Q}$ that best approximates P. This simplification is achieved by expanding the problem to include additional parameters, known as *variational parameters*. Generally speaking, the algorithms in this class can be viewed as optimizing a target function that measures the quality of the approximation. In this work we focus on one variational method called *Belief Propagation*.

5. Our Contribution

In this work we show how to perform model selection in a special type of rMRFs that have inherent symmetry properties. In such tasks we have to run inference for many different models. Our basic observation is that when the model has such symmetry properties it is possible to run approximate inference very efficiently. In particular, we show that many of the intermediate results of approximate inference procedures, such as loopy belief propagation, are identical. Thus, instead of recalculating the same terms over and over, we can perform inference at the template level. We define formally a large class of relational models that have these symmetry properties, show how we can use them to perform efficient approximate inference and compare our results with other methods. This is, to the best of our knowledge, the first *lifted* approximate inference algorithm that works on the template level of the model. Using the efficient inference algorithm we perform model selection for both synthetic and real-life problems. The efficient learning procedure allows us to explore domains that were intractable using previous methods.

6. Related Work

Other works attempted to exploit relational structure for more efficient inference. For example, Pfeffer *et al.* [Pfeffer et al. 1999] used the relational structure to cache repeated computations of intermediate terms that are identical in different instances of the same template. Several recent works derive rules as to when variable elimination can be performed at the template level rather than the instance level, which saves duplicate computations [Poole 2003; de Salvo Braz et al. 2005]. These methods focus on speeding exact inference, and are relevant in models where the intermediate calculations of exact inference have tractable representations. These approaches cannot be applied to models, such as the ones we consider, where intermediate results of variable elimination are exponential. In contrast, our method focuses on template level inference for lifted approximate inference in such intractable models.

This document is organized as follows: in Chapter 2 we define a class of rMRFs and the way to construct them. In Chapter 3 we show how symmetry properties of such models can be used for efficient inference. In the following chapters we study model selection for these kind of models, including parameter estimation (Chapter 4) and feature selection (Chapter 5). Then in Chapter 6 we use our efficient algorithm to learn a generative model for a large scale real-life problem from the Protein-Protein-Interaction domain. Finally, we conclude with a discussion.

Chapter 2

Symmetric relational models

In this chapter we define a class of rMRFs. We will later show how to exploit symmetry properties of models in this class in order to run extremely efficient approximate inference algorithm.

As mentioned in Chapter 1, *Probabilistic Relational Models* (PRMs) provide a language for defining how to construct models from reoccurring sub-components [Friedman et al. 1999; Getoor et al. 2001; Taskar et al. 2002; Poole 2003]. Depending on the specific *instantiation*, these sub-components are duplicated to create the actual probabilistic model. We are interested in models that can be applied for reasoning about the relations between entities. Our motivating example will be reasoning about the structure of interaction networks. We now define a class of relational models that will be convenient for reasoning about these domains. We use a language that is similar to ones previously defined [Richardson and Domingos 2006], but also somewhat different, in order to make our claims in the following chapter more simple and clear.

As with most relational models in the literature we distinguish the *template-level* model that describes the types of objects and components of the model and how they can be applied, from the *instantiation-level* that describes a particular model which is an instantiation of the template to a specific set of entities.

To define a template-level model we first set up the different types of entities we reason about in the model. We distinguish *basic entity types* that describe atomic entities from *complex types* that describe composite entities.

Definition 1 *Given a set $\mathcal{T}_{\text{basic}} = (T_1, \ldots, T_n)$ of* basic entity types *we define two kinds of* **complex types***:*

- *If $T_1, \ldots, T_k$ are basic types, then $T_1 \times \cdots \times T_k$ denotes the type of* ordered tuples *of entities of these types. If $e_1, \ldots, e_k$ are entities of types $T_1, \ldots, T_k$, respectively, then $\langle e_1, \ldots, e_k \rangle$ is of type $T_1 \times \cdots \times T_k$.*
- *If T is a basic type, then T^k denotes the type of* unordered tuples *of entities of type T. If $e_1, \ldots, e_k$ are entities of type T, then $[e_1, \ldots, e_k]$ is of type T^k. When considering unordered tuples, permutations of the basic elements still refer to the same complex*

entity. Thus, if e_1, e_2 are of type T, then both $[e_1, e_2]$ and $[e_2, e_1]$ refer to the same complex entity of type T^2.

∎

For example, suppose we want to reason about undirected graphs. If we define a type T_v for vertices then an undirected edge is of type $T_e \equiv T_v^2$ since an edge is a composite object that consists of two vertices. Note that we use unordered tuples since the edge does not have a direction. That is, both $[v_1, v_2]$ and $[v_2, v_1]$ refer to the same relationship between the two vertices. If we want to model directed edges, we need to reason about ordered tuples $T_e \equiv T_v \times T_v$. Now $\langle v_1, v_2 \rangle$ and $\langle v_2, v_1 \rangle$ refer to two distinct edges. This forms a rich language which enables the representation of complex domains. For example, We can consider social networks, where vertices correspond to people. Now we might also add a type T_l of physical locations. In order to reason about relationships between vertices (people) and locations we need to define pairs of type $T_p \equiv T_v \times T_l$. Note that tuples that relate between different types are by definition ordered.

Once we define the template-level set of types $\mathcal{T}$ over some set of basic types $\mathcal{T}_{\text{basic}}$, we can consider particular instantiations in terms of entities.

Definition 2 *An* **entity instantiation** $\mathcal{I}$ *for* $(\mathcal{T}_{\text{basic}}, \mathcal{T})$ *consists of a set of* basic entities $\mathcal{E}$ *and a mapping* $\sigma : \mathcal{E} \mapsto \mathcal{T}_{\text{basic}}$ *that assigns a basic type to each basic entity.* ∎

Based on an instantiation, we create all possible instantiations of each type in $\mathcal{T}$:

- if $T \in \mathcal{T}_{\text{basic}}$ then $\mathcal{I}(T) = \{e \in \mathcal{E} : \sigma(e) = T\}$
- If $T = T_1 \times \cdots \times T_k$ then $\mathcal{I}(T) = \mathcal{I}(T_1) \times \cdots \times \mathcal{I}(T_k)$.
- If $T = T_1^k$ then $\mathcal{I}(T) = \{[e_1, \ldots, e_k] : e_1, \ldots e_k \in \mathcal{I}(T_1), e_1 \leq \cdots \leq e_k\}$ where $\leq$ is some (arbitrary) order over $\mathcal{I}(T)$ [1].

Once we define a set of basic entities, we assume that all possible complex entities of the given type are defined (see Figure 2.1 for an instantiation of the undirected graph example).

The basic and complex entities define the structure of our domain of interest. Our goal, however, is to reason about the properties of these entities. We refer to these properties as *attributes*. Again, we start by the definition at the template level, and proceed to examine their application to a specific instantiation:

Definition 3 *A* **template attribute** $\mathsf{A}(T)$ *defines a property of entities of type T. The set of values the attribute can take is denoted $Val(\mathsf{A}(T))$.* ∎

1. For example, considering undirected edges again, we think of $[v_1, v_2]$ and $[v_2, v_1]$ as two different names of the same entity. Our definition ensures that only one of these two objects is in the set of entities and we view the other as an alternative reference to the same entity.

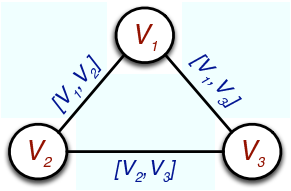


Figure 2.1: An instantiation of an undirected graph scheme over a domain of three vertices.

A template attribute denotes a specific property we expect each object of the given type to have. In general, we can consider attributes of basic objects or attributes of complex objects. In our example, we can reason about the color of a vertex, by having an attribute $\mathsf{Color}(T_v)$. We can also create an attribute $\mathsf{Exist}(T_e)$ that denotes whether the edge between two vertices exists. We can consider other attributes such as the weight of an edge and so on. All these template attributes are defined at the level of the scheme and we will denote by $\mathcal{A}$ the set of template attributes in our model.

Given a concrete entity instance $\mathcal{I}$ we consider all the attributes of each instantiated type. We view the attributes of objects as random variables. Thus, each template attribute in $\mathcal{A}$ defines a set of random variables:

$$\mathcal{X}_{\mathcal{I}}(\mathsf{A}(T)) = \{X_{\mathsf{A}}(e) : e \in \mathcal{I}(T)\}$$

We define $\mathcal{X}_{\mathcal{I}} = \cup_{\mathsf{A}(T)\in\mathcal{A}}\mathcal{X}_{\mathcal{I}}(\mathsf{A}(T))$ to be the set of all random variables that are defined over the instantiation $\mathcal{I}$. For example, if we consider the attributes Color over vertices and Exist over unordered pairs of vertices, and suppose that $\mathcal{E} = \{v_1, v_2, v_3\}$ are all of type T_v, then we have three random variables in $\mathcal{X}(\mathsf{Color}(T_v))$ which are $X_{\mathrm{Color}}(v_1)$, $X_{\mathrm{Color}}(v_2)$, $X_{\mathrm{Color}}(v_3)$, and three random variables in $\mathcal{X}(\mathsf{Exist}(T_e))$ which are $X_{\mathrm{Exist}}([v_1, v_2])$, $X_{\mathrm{Exist}}([v_1, v_3])$ and $X_{\mathrm{Exist}}([v_2, v_3])$.

Given a set of types, their attributes and an instantiation, we defined a universe of discourse, which is the set $\mathcal{X}_{\mathcal{I}}$ of random variables. An *attribute instantiation* ω (or just instantiation) is an assignment of values to all random variables in $\mathcal{X}_{\mathcal{I}}$. We use both $\omega(X_{\mathsf{A}}(e))$ and $x_{\mathsf{A}}(e)$ to refer to the assigned value to the attribute A of the entity e.

We now turn to the final component of our relational model. To define a log-linear model over the random variables $\mathcal{X}_{\mathcal{I}}$, we need to introduce *features* that capture preferences for specific configurations of values to small groups of related random variables. In our graph example, we can introduce a univariate feature for edges that describes the potential for the existence of an edge in the graph. A more complex feature can describe preferences over triplets of interactions (*e.g.*, prefer triangles over open chains).

We start by defining template level features as a recipe that will be assigned to a large number of specific sets of random variables in the instantiated model. Intuitively, a template feature defines a function that can be applied to a set of attributes of related entities. To do so, we need to provide a mechanism to capture sets of entity attributes with particular relationships. For example, to put a feature over a triangle of edges, we want a feature over

	Arguments	Formal entities	Attr.	Function
$\mathcal{F}_e$	$\langle \xi_1, \xi_2 \rangle$ $\langle T_v, T_v \rangle$	$[\xi_1, \xi_2]$ T_e	Exist	$f_\delta(z) = \mathbf{1}\{z = 1\}$
$\mathcal{F}_t$	$\langle \xi_1, \xi_2, \xi_3 \rangle$ $\langle T_v, T_v, T_v \rangle$	$[\xi_1, \xi_2]$ $[\xi_1, \xi_3]$ $[\xi_2, \xi_3]$ T_e, T_e, T_e	Exist Exist Exist	$f_3(z_1, z_2, z_3) =$ $\mathbf{1}\{(z_1 = 1) \wedge$ $(z_2 = 1) \wedge$ $(z_3 = 1) \quad \}$

Table 2.1: Example of two template-level features for a graph model. The first is a feature over single edges, and the second is one over triplets of coincident edges (triangles).

the variables $X_{\text{Exist}}([v_1, v_2])$, $X_{\text{Exist}}([v_1, v_3])$, and $X_{\text{Exist}}([v_2, v_3])$ for every choice of three vertices v_1, v_2, and v_3. The actual definition, thus involves entities that we quantify over (e.g., v_1, v_2, and v_3), the complex entities over these arguments we examine (e.g., $[v_1, v_2]$, $[v_1, v_3]$, and $[v_2, v_3]$), the attributes of these entities, and the actual feature.

Definition 4 *A* **Template Feature** $\mathcal{F}$ *is defined by four components:*

- *A tuple of* arguments $\langle \xi_1, \ldots, \xi_k \rangle$ *with a corresponding list of* type signature $\langle T_1^q, \ldots, T_k^q \rangle$*, such that* ξ_i *denotes an entity of basic type* T_i^q*.*
- *A list of formal entities* $\varepsilon_1, \ldots, \varepsilon_j$*, with corresponding types* $T_1^f, \ldots, T_j^f$ *such that each formal entity* ε *is either one of the arguments, or a complex entity constructed from the arguments. (For technical reasons, we require that formal entities refer to each argument at most once.)*
- *A list of attributes* $\mathsf{A}_1(T_1^f), \ldots, \mathsf{A}_j(T_j^f)$*.*
- *A function* $f : Val(\mathsf{A}_1(T_1^f)) \times \cdots \times Val(\mathsf{A}_j(T_j^f)) \mapsto I\!R$*.*

∎

For example, Table 2.1 shows such a formalization for a graph model with two such template level features.

We view a template-level feature as a recipe for generating multiple instance-level features by applying different *bindings* of objects to the arguments. For example, in our three vertices instantiation, we could create instances of the feature $\mathcal{F}_e$ such as $f_\delta(X_{\text{Exist}}([v_1, v_2]))$ and $f_\delta(X_{\text{Exist}}([v_1, v_3]))$. We now formally define this process.

Definition 5 *Let* $\mathcal{F}$ *be a template feature with components as in Definition 4, and let* $\mathcal{I}$ *be an entity instantiation. A* **binding** *of* $\mathcal{F}$ *is an ordered tuple of* k *entities* $\beta = \langle e_1, \ldots, e_k \rangle$ *such that* $e_i \in \mathcal{I}(T_i^q)$*. A binding is* legal *if each entity in the binding is unique. We define*

$$\text{Bindings}(\mathcal{F}) \quad = \quad \{\beta \in \mathcal{I}(T_1^q) \times \cdots \times \mathcal{I}(T_k^q) : \beta \textit{ is legal for } \mathcal{F}\}$$

Given a binding $\beta = \langle e_1, \ldots, e_k \rangle \in \mathrm{Bindings}(\mathcal{F})$, we define the entity $\varepsilon_i|_\beta$ to be the entity corresponding to ε_i when we assign e_i to the argument ξ_i. Finally, we define the ground feature *$\mathcal{F}|_\beta$ to be the function over ω:*

$$\mathcal{F}|_\beta(\omega) = f\left(\omega(X_{\mathsf{A}_1}(\varepsilon_1|_\beta)), \ldots, \omega(X_{\mathsf{A}_j}(\varepsilon_j|_\beta)\right)$$

∎

For example, consider the binding $\langle v_1, v_2, v_3 \rangle$ for $\mathcal{F}_t$ of Table 2.1. This binding is legal since all three entities are of the proper type and are different from each other. This binding defines the ground feature

$$\mathcal{F}_t|_{\langle v_1, v_2, v_3 \rangle}(\omega) = f_3(x_{\mathsf{Exist}}([v_1, v_2]), x_{\mathsf{Exist}}([v_1, v_3]), x_{\mathsf{Exist}}([v_2, v_3]))$$

That is, $\mathcal{F}_t|_{\langle v_1, v_2, v_3 \rangle}(\omega) = 1$ iff there is a triangle of edges between the vertices v_1, v_2 and v_3. Note that each binding defines a ground feature. However, depending on the choice of feature function, some of these ground features might be equivalent. In our last example, the binding $\langle v_1, v_3, v_2 \rangle$ creates the same feature. While this creates a redundancy, it does not impact the usefulness of the language. We now have all the components in place.

Definition 6 *A* **Relational MRF scheme** *$\mathcal{S}$ is defined by a set of types $\mathcal{T}$, their attributes $\mathcal{A}$ and a set of template features $\mathcal{FF} = \{\mathcal{F}_1, \ldots, \mathcal{F}_k\}$. A* model *is a scheme combined with a vector of* parameters *$\theta = \langle \theta_i, \ldots, \theta_k \rangle \in \mathbb{R}^k$. Given an entity instantiation $\mathcal{I}$ a scheme uniquely defines the universe of discourse $\mathcal{X}_\mathcal{I}$. Using a log-linear representation we can define the joint distribution of a full assignment ω as:*

$$P(\omega : \mathcal{S}, \mathcal{I}, \theta) = \frac{1}{Z(\theta, \mathcal{I})} \exp \sum_{i=1}^{k} \theta_i \mathcal{F}_i(\omega) \tag{2.1}$$

where (with slight abuse of notation)

$$\mathcal{F}_i(\omega) = \sum_{\beta \in \mathrm{Bindings}(\mathcal{F}_i)} \mathcal{F}_i|_\beta(\omega)$$

is the total weight of all groundings of the feature $\mathcal{F}_i$, and Z is the normalizing constant, also called the partition function. ∎

This definition of a joint distribution is similar to standard log-linear models, except that all groundings of a template feature share the same parameter [Della Pietra et al. 1997]. Notice that this means features are not necessarily binary which will influence the complexity of the learning task (more details will follow in Chapter 4).

Now that we have defined the class of models of interest, we are ready to address the problem of inference in such models.

Chapter 3

Compact Approximate Inference

As mentioned earlier, variational methods are a broad class of approximate inference algorithms. Here we show the application of our idea to *loopy belief propagation* [Murphy et al. 1999; Yedidia et al. 2002], which is one of the most common approaches in this field. At the end of this chapter We make a note on applying the same idea to a broader class of variational methods called *Generalized Belief Propagation* [Yedidia et al. 2002].

1. Belief Propagation

In the Belief Propagation algorithm we introduce (variational) variables which can naturally be understood as *messages* between nodes in the graph about the state they should be in [Pearl 1988]. It is sometimes convenient to view this process as operating on a data structure called *Factor Graph* [Kschischang et al. 2001] (more details will follow bellow). The *belief* of a group of nodes is obtained by the product of its local potential and all messages coming into it (Eq. (3.5)). The algorithm uses a recursive message update rule defined bellow in Eq. (3.3) and Eq. (3.4).

The belief propagation algorithm updates messages of this kind until they converge to some value. If the graph is a tree, this recursive algorithm is guaranteed to converge to the correct marginal probabilities (in a single iteration if the order is chosen right). Surprisingly, the same algorithm turns out to work well in many problems in which the graph structure contains loops [Murphy et al. 1999]. To understand this success we turn to the concept of *Energy Functions* [Yedidia et al. 2002].

1.1 Free Energies

As mentioned in Chapter 1, we are looking for a distribution Q that is both simple (so we could run inference efficiently) and close to our target distribution P. A natural measure of distance between distributions is the Kullback-Leibler divergence (KL), also known as the *relative entropy*, defined by: $D\left(Q||P\right) = \sum_x q(x) \ln \frac{q(x)}{p(x)}$. So we have an optimization problem where we are looking for $\mathrm{argmin}_Q D\left(Q||P\right)$.

If we assume that P factorizes as in Eq. (1.1) then:

$$\begin{aligned} D\left(Q||P\right) &= \sum_x q(x) \ln \frac{q(x)}{p(x)} \\ &= \sum_x q(x) \ln q(x) - \sum_x q(x) \ln \left(\frac{1}{Z} \prod_c \pi_c(x_c) \right) \\ &= \sum_x q(x) \ln q(x) - \sum_x q(x) \sum_c \ln \pi_c(x_c) + \ln Z \\ &= -\mathcal{H}\left(q(x)\right) - \mathcal{U}\left(q(x)\right)) + \ln Z \\ &= \ln Z - F[P,Q] \end{aligned} \tag{3.1}$$

Where we denote the entropy of Q by $\mathcal{H}\left(q(x)\right), \mathcal{U}\left(q(x)\right))$ is called the *average energy*, and $F[P,Q] = \mathcal{U}\left(q(x)\right)) + \mathcal{H}\left(q(x)\right)$ is the *energy functional* which is related to concepts from statistical mechanics.

This result has important ramifications. First, since $\ln Z$ does not depend on Q, minimizing $D\left(Q||P\right)$ is equivalent to maximizing $F[P,Q]$. Second, since $D\left(Q||P\right) \geq 0$ for any two distributions we have that $\ln Z \geq F[P,Q]$, which means that the energy functional gives a lower bound on the logarithm of the partition function.

By the properties of KL divergence we know that there exists a unique optimal solution to this optimization problem in which: $Q = P$, $D\left(Q||P\right) = 0$ and $F[P,Q] = \ln Z$. However, optimizing $F[P,Q]$ directly is computationally expensive, as expected. Instead, we can try to find the optimum of an approximation to $F[P,Q]$. Surprisingly, It has been shown [Yedidia et al. 2002] that the BP algorithm can be viewed as optimizing an approximation to the energy functional called the *Bethe approximation* [Bethe 1935].

This approximation is defined as:

$$F_{Bethe}[P,Q] = \sum_c \sum_{x_c} b(x_c) \ln(\pi_c(x_c)) + \sum_c \mathcal{H}_{\pi_c}(c) - \sum_i (d_i - 1)\mathcal{H}_{\pi_i}(X_i) \tag{3.2}$$

where $b(x_c)$ are our approximations to $q(x)$ marginal probabilities, and $d_i = |\{c : X_i \in scope(c)\}|$ is the number of cliques containing the variable X_i in their scope. Note that in this approximation we sum over variable and cluster potentials so Q is a rather simple distribution to handle.

We can now formulate the revised optimization problem as:

$$\begin{array}{ll} Find & Q = \{\pi_i : \mathbf{C}_i \in \kappa\} \cup \{\mu_{i,j} : \mathbf{C}_i - \mathbf{C}_j \in \kappa\} \\ That\ maximizes & F_{Bethe}[P,Q] \\ Subject\ to & \mu_{i,j}[s_{i,j}] = \sum_{\mathbf{C}_i - \mathbf{S}_{i,j}} \pi_i[c_i] \quad \forall (\mathbf{C}_i - \mathbf{C}_j) \in \kappa, \forall s_{i,j} \in Val(\mathbf{S}_{i,j}) \\ & \sum_{\mathbf{C}_i} \pi_i[c_i] = 1 \quad \forall \mathbf{C}_i \in \kappa \end{array}$$

where $\mathbf{C}$ are cliques in the graph (denoted as κ), and $\mu_{i,j}$ can be viewed as messages between cliques. The constraints are introduced to ensure that marginal probabilities over

cliques are calibrated through messages, and that the local beliefs are legal distributions (they should sum to 1).

Using Lagrange multipliers we can characterize the fixed point of the optimum of this constrained optimization problem by a set of equations. These equations can be reformulated, in turn, to yield an iterative approach for optimizing the parameters of Q ($b(x_c)$). This iterative procedure can be viewed as message passing in the graph associated with the model, exactly as done in belief propagation.

2. Factor Graphs

To describe loopy belief propagation we consider the data structure of a Factor Graph [Kschischang et al. 2001]. A factor graph is a bi-partite graph that consists of two layers. In the first layer, we have for each random variable in the domain a *variable node* X. In the second layer we have *factor nodes* (see Figure 3.1(a)). Each factor node ψ is associated with a set $\mathbf{C}_\psi$ of random variables and a feature π_ψ. If $X \in \mathbf{C}_\psi$, then we connect the variable node X to the factor node ψ.

A factor graph is *faithful* to a log-linear model if each feature is assigned to a node whose scope contains the scope of the feature. Combining all these features multiplied by their parameters defines for each factor node ψ a potential function $\pi_\psi[\mathbf{c}_\psi]$ that assigns a real value for each value of $\mathbf{C}_\psi$. For example, if the potential has the form of a tabular CPD, then each entry in the table is a multiplication of all features that are consistent with the assignment of that entry and their parameters (the feature may not include all variable in the potential's scope). There is usually a lot of flexibility in defining the set of factor nodes. For simplicity, we focus now on factor graphs where we have a factor node for each ground feature.

For example, let us consider a model over an undirected graph where we also depict the colors of the vertices. We create for each vertex v_i a variable node $X_{\mathsf{Color}}(v_i)$ and for each pair of vertices $[v_i, v_j]$ a variable node $X_{\mathsf{Exist}}([v_i, v_j])$. We consider two template features - the triangle feature we described earlier, and a co-colorization feature that describes a preference of two vertices that are connected by an edge to have the same color. To instantiate the triangle feature, we consider all undirected tuples of three vertices $\beta = [v_i, v_j, v_k] \in \mathrm{Bindings}(\mathcal{F}_t)$ and define ψ_β with scope $\mathbf{C}_\beta = \{X_{\mathsf{Exist}}([v_i, v_j]), X_{\mathsf{Exist}}([v_i, v_k]), X_{\mathsf{Exist}}([v_j, v_k])\}$. To instantiate the co-colorization feature, we consider all tuples of two vertices $\beta = [v_i, v_j] \in \mathrm{Bindings}(\mathcal{F}_e)$ and define ψ_β with scope $\mathbf{C}_\beta = \{X_{\mathsf{Exist}}([v_i, v_j]), X_{\mathsf{Color}}(v_i), X_{\mathsf{Color}}(v_j)\}$. See Figure 3.1(a) for such a factor graph instantiated over 4 vertices. This factor graph is faithful since each ground feature is assigned to a dedicated feature node.

Loopy belief propagation over a factor graph is defined as repeatedly updating messages of the following form:

$$m_{X \to \psi}(x) \quad \leftarrow \quad \prod_{\psi' : X \in \mathbf{C}_{\psi'}, \psi' \neq \psi} m_{\psi' \to X}(x) \tag{3.3}$$

$$m_{\psi \to X}(x) \leftarrow \sum_{\mathbf{c}_\psi \langle X \rangle = x} \left(\pi_\psi[\mathbf{c}_\psi] \prod_{X \neq X' \in \mathbf{C}_\psi} m_{X' \to \psi}(x') \right) \tag{3.4}$$

where $\mathbf{c}_\psi\langle X\rangle$ is the value of X in the assignment of values $\mathbf{c}_\psi$ to $\mathbf{C}_\psi$. When these messages converge, we can define beliefs about nodes in the factor graph as:

$$b_\psi(\mathbf{c}_\psi) \propto \pi_\psi[x] \prod_{X' \in \mathbf{C}} m_{X \to \psi}(\mathbf{c}_\psi \langle X' \rangle) \tag{3.5}$$

where the beliefs over $\mathbf{C}_\psi$ are normalized to sum to 1. These beliefs are the approximation of the marginal probability over the variables in $\mathbf{C}_\psi$ [Yedidia et al. 2002].

Trying to reason about a network over 1000 vertices with features over univariate ($\mathcal{F}_e$) and triangle ($\mathcal{F}_t$) that we described earlier, will produce $\binom{1000}{2}$ variable nodes (one for each edge), and $\binom{1000}{3}$ triplet feature. Unfortunately, building the factor graph for this problem and performing loopy belief propagation with it is extremely time consuming. However, our main insight is that we can exploit some special properties of this model for much efficient representation and inference. The basic observation is that the factor graphs for the class of models we defined satisfy basic symmetry properties.

Specifically, consider the structure of the factor graph we just described. An instantiation of graph vertices defines both the list of random variables and of features that will be created. Each feature node represents a ground feature that originates from a legal binding to a template feature. Each grounding of an edge variable or an edge feature ($\mathcal{F}_e|_\beta$) spans two vertices, while the groundings of the triplet feature ($\mathcal{F}_t|_\beta$) cover three vertices. Since we are considering all legal bindings (*i.e.*, all 2-mers and 3-mers of vertices) while spanning the factor graph, each edge variable node will be included in the scope of 1 edge feature node and $(n-2)$ triplet feature nodes. More importantly, since all the edge variables have the same *local neighborhood*, they will also compute the same messages during belief propagation over and over again.

3. Compact Belief Propagation

We now formalize this idea and show we can use it to enable efficient representation and inference.

Definition 7 *We say that two nodes in the factor graph have the same* **type** *if they were instantiated from the same template attribute or template feature. We say that a factor graph has the* **local neighborhood** *property if every two nodes in the factor graph having the same type are connected to the same number of nodes of each type.* ∎

In the example above, each variable node of type edge has $(n-2)$ neighbors of type triplet and each factor node of type triplet has 3 neighbors of type edge.

Given this definition, we can present our main claim formally:

Theorem 3.1: If a factor graph has the local neighborhood property then at every stage t of *synchronous* belief propagation that is initiated with uniform messages, if v_i, v_k are a two factor graph nodes from the same type and also v_j, v_l are from the same type then $m^t_{v_i \to v_j}(x) = m^t_{v_k \to v_l}(x)$.

Proof: The proof is by induction over the stage of the belief propagation algorithm. For $t = 0$ the equality holds since all messages are uniform. Now let us assume that $m^{t-1}_{v_i \to v_j}(x) = m^{t-1}_{v_k \to v_l}(x)$. We consider two cases: either v_i, v_k are variable nodes and v_j, v_l are factor nodes, or vice versa. In the first case, we use the inductive assumption and the local neighborhood property to get:

$$\begin{aligned} m^t_{v_i \to v_j}(x) &= \prod_{\psi' : v_i \in \mathbf{C}_{\psi'}, \psi' \neq v_j} m^{t-1}_{\psi' \to v_i}(x) \\ &= \prod_{\psi' : v_k \in \mathbf{C}_{\psi'}, \psi' \neq v_l} m^{t-1}_{\psi' \to v_k}(x) \\ &= m^t_{v_k \to v_l}(x) \end{aligned}$$

And similarly for the second case:

$$\begin{aligned} m^t_{v_i \to v_j}(x) &= \sum_{\mathbf{c}_{v_i} \langle v_j \rangle = x} \left(\pi_{v_i}[\mathbf{c}_{v_i}] \prod_{v_j \neq X' \in \mathbf{C}_{v_i}} m^{t-1}_{X' \to v_i}(x') \right) \\ &= \sum_{\mathbf{c}_{v_k} \langle v_l \rangle = x} \left(\pi_{v_k}[\mathbf{c}_{v_k}] \prod_{v_l \neq X' \in \mathbf{C}_{v_k}} m^{t-1}_{X' \to v_k}(x') \right) \\ &= m^t_{v_k \to v_l}(x) \end{aligned}$$

And this concludes the inductive step. ∎

The requirement that a factor graph has the local neighborhood property might seem too restrictive. However, it turns out that many interesting problems obey this requirement. Specifically, we can show that if we build a model according to Definition 6 over all legal bindings, then the resulting factor graph has the desired property. In this work we focus on such models, but other interesting problems, such as the wrapped-around-grid, also fall into this category.

We now prove the first claim:

Lemma 8 In a model created according to Definition 6 over all legal bindings, if two nodes in the factor graph have the same type, then they have the same local neighborhood. That is, they have the same number of neighbors of each type.

Proof: If v_i and v_j are factor nodes, then since they are of the same type, they are instantiations of the same template feature. From Definition 4 and Definition 5 we can see that

this means that they are defined over variables from the same type. Since each feature is connected only to the variables in its scope, this proves our claim. However, if v_i and v_j are variable nodes, it suffices to show that they take part in the same types of features, and in the same number of features of each such type. For simplicity, we will assume that v_i is instantiated from the attribute of some basic type T (the proof in case it is a complex type is similar). We need to compute how many ground features contain v_i in their scope, and do not contain v_j. From Definition 5 we can see that all the legal bindings that include v_i and do not include v_j are legal also if we replace v_i with v_j. ∎

After showing that many calculations are done over and over again, we now show how we can use a more efficient representation to enable much faster inference.

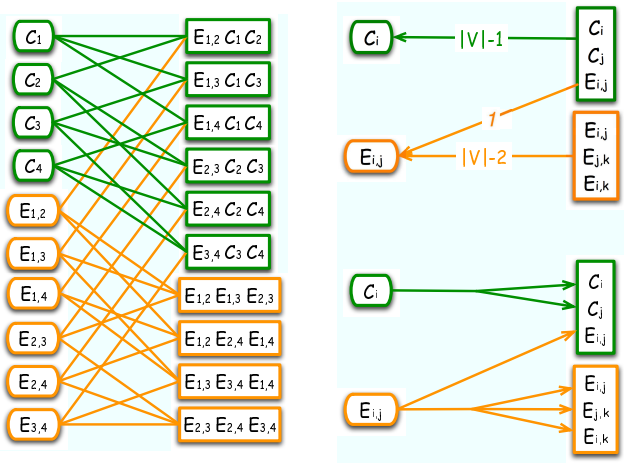
Definition 9 *A* **template factor graph** *over a template log-linear model is a bi-partite graph, with one level corresponding to attributes and the other corresponding to template features.*

- *Each template attribute* T *that corresponds to a formal entity in some template feature* $\mathcal{F}$ *is mapped to a* **template attribute node** *on one side of the graph. And each template feature is mapped to a* **template feature node** *on the other side of the graph. Each template attribute node is connected with an edge to all the template feature nodes that contain this attribute in their scope.*
- *A feature node needs to distinguish between its neighbors, since each message carries information about a different variable. Hence, in the template factor graph we term an association to a variable inside a template feature node as* **port** *. If a factor contains more than one variable of the same type, the corresponding edge splits to the corresponding ports when arriving to the factor node.*
- *In addition, each ground variable node takes part in many features that were instantiated by the same template feature with different bindings. Hence, each edge from a template feature node to a template attribute node in the template factor graph is assigned with a* **cardinality** *indicating the number of repetitions it has in the full factor graph.*

∎

Figure 3.1(b) shows such a template factor graph for the Triangle-Colorization example.

Running loopy belief propagation on this template factor graph is straightforward. The algorithm is similar to the standard belief propagation only that when an edge in the template-graph represents many edges in the instance-level factor graph, we interpret this by raising the message to the appropriate power. The number of edges in the instance-level factor graph (cardinality) is obtained by a simple combinatorical computation. Since Theorem 3.1 shows that at all stages in the standard synchronous belief propagation the messages between nodes of the same type are similar, running belief propagation on the template factor graph is equivalent to running synchronous belief propagation on the full factor graph. However, we reduced the cost of representation and inference from being



(a) Full factor graph (b) Compact factor graph

Figure 3.1: Shown are the full (a) and template (b) factor graphs modeling a colored graph. We have basic types for colors and vertices, and a complex type for edges. We consider two template features - the triangle feature and a co-colorization feature. For clarity, $X_{\mathsf{Exist}}([v_i, v_j])$ is shown as $E_{i,j}$ and $X_{\mathsf{Color}}(v_i)$ is shown as C_i. Orange edges show the edges connected to edge variables and green edges are connected to color variables. $|V|$ stands for the number of vertices in the graph.

proportional to the size of the instantiated model, to be proportional to the size of the template-level scheme. Specifically, this representation does not depend on the size of the instantiations and can deal with a huge number of variables.

4. Experimental Results

To evaluate our method in inference tasks we built a template-level model which includes univariate ($\mathcal{F}_e$) and closed-triangle ($\mathcal{F}_t$) features (as described in the previous section), and then perform inference with various combinations of parameter values. We compare results of other inference methods such as exact inference, MCMC [Geman and Geman 1984], and standard asynchronous belief propagation [Pearl 1988], to those of our compact belief propagation (CBP). First we consider small models where exact inference is feasible, and then we move to larger domains were we can only compare MCMC and CBP. We compare inference results in two different ways. In the first we compare marginal beliefs over some region, and in the second we compare estimates of the partition function.

Figure 3.2 shows a comparison of the marginal distributions over edge variables for different parameter settings and different inference methods. We observe that in small graphs the marginal beliefs are very similar for all inference methods. To quantify the similarity we calculate the relative deviation from the true marginals. We find that on average MCMC deviates by 0.0118 from the true marginal (stdev: 0.0159), while both belief propagation methods deviate on average by 0.0143 (stdev: 0.0817) and are virtually indistinguishable. However, in the graph over 7 vertices we notice that the two loopy belief

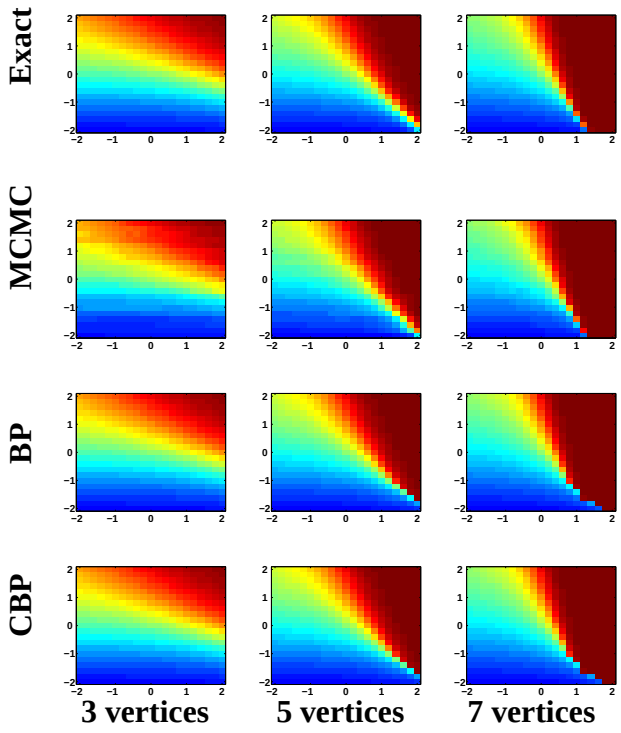


Figure 3.2: Comparison of inference methods via marginal beliefs. Each panel visualizes the the probability of an interaction when we vary two parameters: θ_e - the univariate potential for interaction (y-axis) and the potential θ_t - over closed triplet (x-axis). The color indicates probability where blue means probability closer to 0 and red means probability closer to 1. The first row of panels shows exact computation, the second MCMC, the third standard asynchronous belief propagation, and at the bottom row is our compact belief propagation.

propagation methods (BP and CBP) are slightly different from the rest in the case where the univariate parameter is small and the triplet parameter is large (lower right corner).

An alternative measurement of inference quality is the estimate of the partition function. This is especially important for learning applications, as this quantity serves to compute the likelihood function. When performing loopy belief propagation we can approximate the log-partition function using the Bethe approximation (Eq. (3.2)). As seen in Figure 3.3, the estimate of the log partition function by belief propagation closely tracks the exact solution. Moreover, as in the marginal belief test, the full and compact variants are almost indistinguishable.

It is important to note that running times are substantially different between the methods. For example, using exact inference with the 7 vertices graph (*i.e.*, one pixel in the matrices shown in Figure 3.2) takes 80 seconds on a 2.4 GHz Dual Core AMD based machine. Approximating the marginal probability using MCMC takes 0.3 seconds, standard BP takes 12 seconds, and compact BP takes 0.07 seconds.

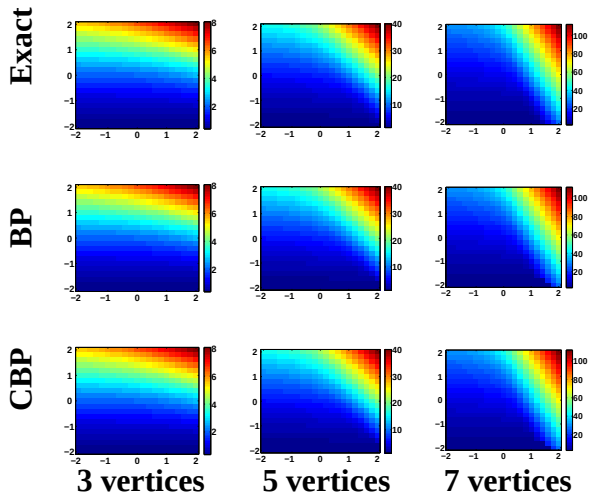


Figure 3.3: Comparison of inference methods for computing the log-partition function. Each panel visualizes the log-partition function (or its approximation) for different parameter settings (as in Figure 3.2). In the belief propagation methods, the log-partition function is approximated using the Bethe free energy approximation. On the first row is the exact computation, the second row shows standard asynchronous belief propagation, and the bottom row shows our compact belief propagation.

On larger models, where exact inference and standard belief propagation are infeasible, we compare only compact belief propagation and MCMC (see Figure 3.4). While there are some differences in marginal beliefs, we see again that in general there is good agreement between the two inference procedures. As the graph becomes larger the gain in run-time increases. Since the mixing time of MCMC should depend on the size of the model (if accuracy is to be conserved), running MCMC inference on a 100-node graph was set to 5 minutes. Note that in the region of low parameter values MCMC gives high estimates of the marginal probability. This indicates that we should have actually run the procedure for a longer time to get better marginals. As expected, compact BP still runs for only 0.07 seconds as it depends on the size of the scheme which remains the same. For protein-protein interaction networks over hundreds of vertices (see Chapter 6) all inference methods become infeasible except for compact belief propagation.

5. A Note on Generalized Belief Propagation

A broader class of variational algorithms, of which BP is a special case, is called *Generalized Belief Propagation* (GBP) [Yedidia et al. 2002]. In these methods a slightly different approximation to the energy functional is used, which is called *Kikuchi approximation* [Kikuchi 1951]. The Bethe approximation is a special case of the Kikuchi approximation. A similar derivation, which characterizes the fixed point of the approximate energy

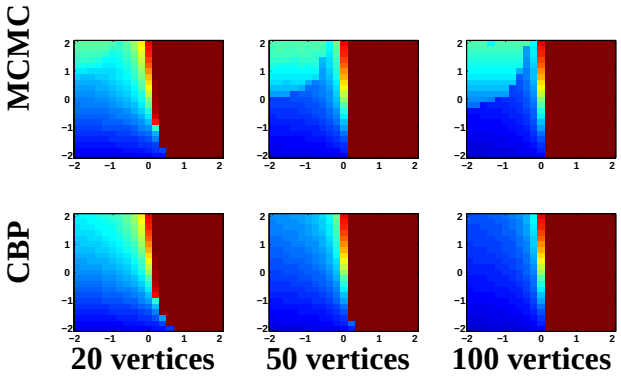


Figure 3.4: Comparison of approximate inference methods on larger graph instances. As before, we show the probability of an interaction as a function of parameter settings. On the first row is MCMC and the second row shows our compact belief propagation.

functional under the constraints, shows that this approximation can be achieved by passing messages on a graph structure. In the case of GBP such graphs are called *Region Graphs*.

In many cases GBP has considerably outperformed BP [Yedidia et al. 2002], and therefore it would be natural to try to apply our main idea to GBP as well. Recall that we view BP as an algorithm operating on a factor graph. In a similar way, GBP can be viewed as operating on a region graph. In a factor graph each factor node corresponds to a potential in the model. A region graph is more flexible, allowing to define regions over arbitrary subsets of nodes, as long as each potential is contained in the scope of at least one region. Unlike the factor graph, the region graph is not necessarily a bipartite graph so messages between regions do not have to pass through single variable nodes and can therefore be more informative about the joint distributions of their variables. The *counting number* C_R of each region R is set in a way that ensures that we count every variable and potential exactly once. See Figure 3.5(a) for an example.

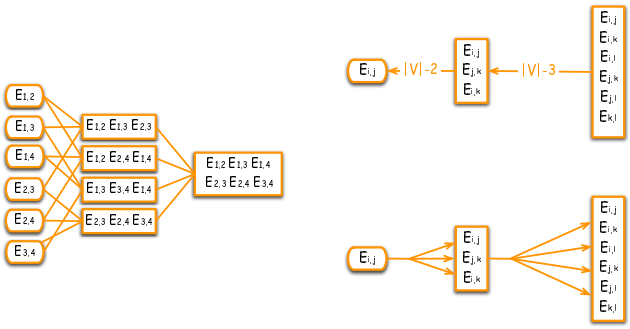
The approximate free energy in this case, termed *Kikuchi Free Energy* has the form:

$$F_{Kikuchi}[P,Q] = \sum_{R \in Top} \sum_{x_R} b(x_R) \ln(\pi^0_R(x_R)) + \sum_R C_R \mathcal{H}_{\pi_R}(X_R) \tag{3.6}$$

where Top is the set of largest regions (which are not contained in others), π^0_R is the product of all factors contained in region R, and C_R are the counting numbers. Notice that this definition differs from Eq. (3.2) in the Entropy term, which is identical when the region graph is actually a factor graph. The Entropy term will play an important role in the following analysis.

We can show that the main idea we presented above applies also for GBP. In other words, we can build a *Template Region Graph* (for an example see Figure 3.5(b)) and run compact message passing on it in a similar way we did for template factor graphs.

However, when trying to run GBP in our template-level setting and comparing approximate marginal probabilities and likelihood results to exact computation (not shown), we



(a) Full region graph (b) Template region graph

Figure 3.5: Shown are the full (a) and template (b) region graphs modeling an undirected graph. This region graph has 3 types of regions: regions over variables for undirected edges ($E_{i,j}$), regions over triplets of edges defined by triplets of vertices in the graph, and regions over 6-mers of edges defined by quadruples of graph vertices. Note that potentials might be defined for univariate edges and triplets but not for quadruples. The ports and edge cardinality are similar to those defined for a template factor graph.

get that for some parameter values the approximation is good while for other parameter combinations it is rather poor (much worse than the BP approximation). It turns out that the accuracy of GBP is highly dependent on the way the set of regions is chosen [Welling 2004]. Specifically, we noted before that the entropy term plays a central role in the approximation of the free energy functional. An approximation is called *maxent-normal* if the region-based entropy $H_R(b_R)$ achieves its maximum when all beliefs $b_R(x_R)$ are uniform. Unfortunately, as we now show, when we build a simple and intuitive template region graph for our domain, the resulting region-based approximation is not maxent-normal and we end up with a poor approximation.

To see this we follow a similar argument from Yedidia *et al.* [Yedidia et al. 2004] and use the *cluster variation method* to define a region graph. In this approach we begin with a set of large regions and repeatedly intersect regions to form layers of smaller regions until we reach single variables. If we take our previous example of an undirected graph over N vertices and define regions over 4 vertices we get regions over 6 edges (the full graph over 4 nodes), regions over 3 vertices - each comes from the intersection of two larger regions, and finally regions over pairs of vertices (edges in the undirected graph). Figure 3.5(b) depicts such region graph. The maximum entropy in this case can be calculated for the uniform distribution over $\binom{N}{2}$ binary variables (one for each edge), so we get:

$$H_{max} = \binom{N}{2} ln2$$

We now compare this to the entropy induced by the region graph we defined. There are $\binom{N}{4}$ "quadruple" regions with counting number $C_{R_4} = 1$. There are $\binom{N}{3}$ "triplet" regions each having counting number $C_{R_3} = 1 - (N - 3) = 4 - N$ (since each triplet is contained

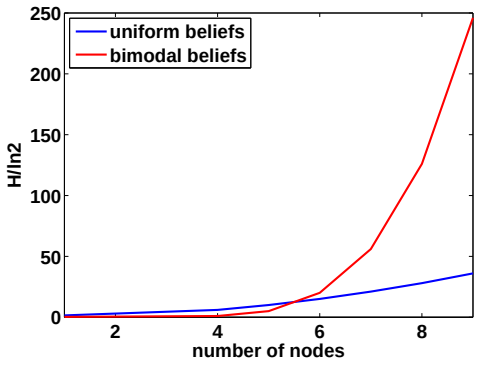


Figure 3.6: Comparison of the region-based entropy for a bimodal distribution and maximum entropy for increasing graph sizes. Regions are defined over quadruples, triplets and pairs of graph vertices using the cluster variation method. We see that when the graph contains more than 6 nodes it is no longer maxent-normal and therefore unlikely to give a good approximation.

in $N-3$ quadruples). Finally, there are $\binom{N}{2}$ edge regions with counting number of $C_{R_1} = 1-(N-2)C_{R_3} - \binom{N-2}{2}C_{R_4}$.

Next we examine the bimodal beliefs which allow either the full graph or the empty one with equal probability (=$\frac{1}{2}$). For these marginal beliefs the entropy of each region is exactly $ln2$ so the overall entropy is the sum of counting numbers:

$$H_{region} = ln2(\binom{N}{4}C_{R_4} + \binom{N}{3}C_{R_3} + \binom{N}{2}C_{R_1})$$

Finally, we get that $H_{region} > H_{max}$ for every $N \geq 6$ and our approximation is not maxent-normal (see Figure 3.6). Therefore, we conclude that constructing intuitive template region graphs for symmetric domains in an automated manner could not be expected to work well in general.

Chapter 4

Parameter Estimation

We now address the task of learning the parameters $\theta = \langle \theta_1 \dots \theta_k \rangle$ assuming that the set of template features $\mathscr{F} = \{\mathcal{F}_1, \dots, \mathcal{F}_k\}$ is known.

1. Maximum Likelihood Estimator

To learn such parameters from evidence we can use the *Maximum Likelihood Estimator* (MLE) [Della Pietra et al. 1997]. In this method we look for the parameters that best explain the data in the sense that they find:

$$\theta^{MLE} = \text{argmax}_{\theta \in \Theta} P(\boldsymbol{D}|\theta)$$

Since there is no closed form for finding the MLE parameters of a log-linear model, various optimization techniques can be employed to find an approximate solution. Before we delve into this optimization problem we stop to make a remark about its relation to another prominent concept, that of *Maximum Entropy*.

In many works the problem of model selection by empirical evidence is viewed from another intuitive direction [Della Pietra et al. 1997; Dudik et al. 2007]. Instead of looking for θ^{MLE} one might want to find a distribution that satisfies the constraints imposed by the training data but has no additional information. Since entropy can be viewed as the inverse of information we should search for the distribution with highest entropy. This, in turn, is equivalent to finding a distribution that minimizes the Kullback-Leibler (KL) divergence with respect to the empirical distribution of the training data. Surprisingly, it turns out that the Gibbs distribution defined by a log-linear model with parameters θ^{MLE} is exactly the distribution of maximum entropy (or minimum KL). In fact, the two problems are convex duals of each other.

We now return to the optimization problem involved in finding θ^{MLE}. Instead of working with the likelihood function, it is more convenient to work with the log-likelihood:

$$\ell(\boldsymbol{D}) = \ln P(\boldsymbol{D}|\theta) = \sum_m \left(\sum_i (\mathcal{F}_i(x[m])\theta_i) - \ln Z \right) \tag{4.1}$$

where $\boldsymbol{D} = x[1], ..., x[M]$ is the set of training samples and Z is the partition function. To calculate log-likelihood, $\sum_i (\mathcal{F}_i(x[m])\theta_i)$ is easily obtained when learning from fully observed evidence, and the partition function Z can be approximated efficiently using our inference algorithm by the Bethe approximation. To see this recall from Eq. (3.1) that: $\ln Z = F[P, Q] + D(Q||P)$, so if we assume the approximation is good, then we can ignore $D(Q||P)$ and approximate the log-partition function by $\ln Z \approx F_{Bethe}[P, Q]$ (using Eq. (3.2)).

The log-likelihood is a concave function of the parameters, and since there is no closed form for θ^{MLE} we resort to a greedy search. Unfortunately, since we only have an approximation to the log-likelihood, we cannot assume concavity, and our greedy search is not guaranteed to converge to the global optimum. Instead, it finds a local maximum of the log-likelihood function. In such greedy approach an efficient calculation of the gradient is often needed.

The partial derivative of the log-likelihood $\ell(\boldsymbol{D})$ with respect to a parameter θ_j that corresponds to a template feature $\mathcal{F}_j$ can be described as:

$$\frac{\partial \ell(\boldsymbol{D})}{\partial \theta_j} = \hat{\boldsymbol{E}}_{\mathcal{D}}[\mathcal{F}_j] - M\boldsymbol{E}_{\theta}[\mathcal{F}_j] \tag{4.2}$$

Where $\hat{\boldsymbol{E}}_{\mathcal{D}}[\mathcal{F}_j]$ is the number of instances of the template feature $\mathcal{F}_j$ in $\boldsymbol{D}$, and $\boldsymbol{E}_{\theta}[\mathcal{F}_j]$ is the number of times we expect to see groundings of the template feature $\mathcal{F}_j$ according to θ [Della Pietra et al. 1997]. This expression has an intuitive interpretation: the gradient attempts to make the expected counts of a feature relative to the model equal to the counts of that feature in the empirical data. Again, the first term is relatively easy to compute in case we learn from fully observed instances, since it is simply the count of each feature in $\boldsymbol{D}$, and the second term can be approximated efficiently by our inference algorithm.

We tried several optimization techniques to find parameters that achieve high likelihood values. Some of them use only log-likelihood estimates, some use only gradient estimates, and some use both function and derivative information for parameter search (more details bellow).

2. Regularization

Unfortunately, maximum likelihood estimation is prone to overfitting to the training data. One way to overcome this is by introducing a prior distribution over the model parameters [Williams 1995; Chen and Rosenfeld 2000; Lee et al. 2007]. Two commonly used priors are the Gaussian prior and the Laplacian prior.
The Gaussian prior takes the form:

$$P_{Gaussian}(\theta|\sigma) = \prod_i \frac{1}{\sqrt{2\pi}} \exp\left\{-\frac{\theta_i^2}{2\sigma^2}\right\}$$

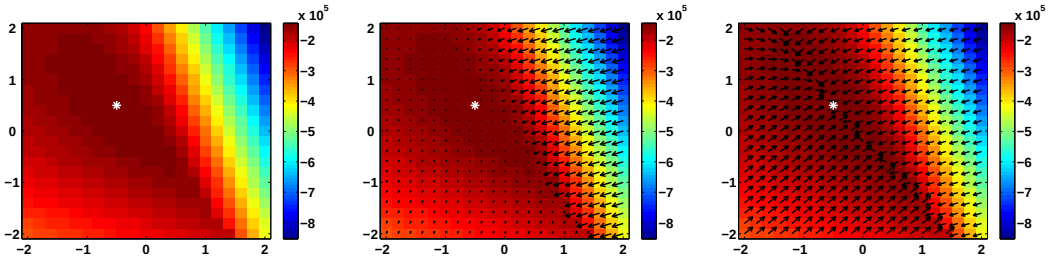


Figure 4.1: Approximate log-likelihood (a), gradient (b) and normalized gradient (c) landscape for a model of 7-node graph with features over univariate ($\mathcal{F}_e$) and closed-triangles ($\mathcal{F}_t$). In all panels values of θ_t and θ_e are shown on the x and y axes respectively. The bright asterisk shows the original parameter values that were used to generate the evidence. The middle panel shows the direction of the derivative as well as its size while the right panel shows only the derivative direction as it is normalized.

and the Laplacian prior has the form:

$$P_{Laplacian}\left(\theta|\beta\right) = \frac{1}{2\beta}\exp\left\{-\frac{|\theta|}{\beta}\right\}$$

Combining the prior with the log-likelihood function gives rise to a penalty term. In the Gaussian case this term has the form: $-\frac{1}{2\sigma^2}\sum_i \theta_i^2$, whereas in the Laplacian case we get: $-\frac{1}{2\beta}\sum_i |\theta_i|$. The first is called L_2-regularization term and the second is called L_1-regularization term. Applying the regularization terms to log-likelihood derivative, we get $-\frac{\theta_j}{\sigma^2}$ in the Gaussian case and $-\frac{1}{2\beta}sign(\theta_j)$ in the Laplacian case. In both L_1 and L_2 we penalize the magnitude of the parameters. This penalty provides a continued incentive for parameters to shrink and therefore the learned models tend to be sparser, especially with L_1 (since the L_2 penalty diminishes as the parameters get close to 0) [Tibshirani 1996]. We will use this consequence for feature selection in Chapter 5. Importantly, both L_1 and L_2 regularization terms are concave so the penalized log-likelihood is also concave and we can therefore use the same optimization techniques as in the unpenalized case.

3. Experimental Results

Using our efficient inference approximation we can reevaluate the log-likelihood and its derivative for many parameter values and thereby gain an unprecedented view of the likelihood landscape of the model. We continue with our toy model with features over univariate ($\mathcal{F}_e$) and closed-triangles ($\mathcal{F}_t$) and show in Figure 4.1 the log-likelihood and gradients calculated for a grid of parameter values. For this we start the model with some parameter values ($\theta_e = 0.5$, $\theta_t = -0.5$) and use a Gibbs sampler to produce evidence (10K samples). We then run CBP which uses the Bethe approximation of the partition function to calculate log-likelihood for multiple combinations of parameter values.

4. Optimization Problem

As mentioned, since there is no closed form solution for finding θ^{MLE}, we use greedy search methods. In order to find the MLE parameters we now study several optimization techniques. These techniques rely on likelihood function estimations, gradient estimation, or combine information of both in order to find regions in parameter space with high likelihood values.

Figure 4.2 shows learning traces of the various optimization techniques that we survey bellow.

Conjugate Gradient One widely used optimization technique is *Conjugate Gradient* (CG). In this method the function is evaluated along the direction of the gradient and the point of maximal value is chosen as the starting point for the next iteration. The next line search is performed in the direction of the conjugate direction to that of the previous step (CG methods differ in the way they define the conjugate direction). The step size is increased or decreased according to whether the last step was successful in improving function value. This strategy has been shown to converge faster than a simple steepest ascent algorithm [Fletcher 1987]. We tested two CG variants including: Fletcher-Reeves algorithm (CG-FR) and Polak-Ribiere algorithm (CG-PR) (for more details see [Fletcher 1987]).

Quasi-Newton We also tried a Quasi-Newton algorithm of Broyden-Fletcher-Goldfarb-Shanno (BFGS) which attempts to estimate the second derivative using first derivative estimates in multiple locations [Fletcher 1987].

Both CG and BFGS prove effective in finding the MLE, however the problem of using them is their assumption that the function is concave along the search line. In some scenarios sensitivity to small fluctuations in function estimates causes the search to terminate prematurely. We tried to alleviate this problem by replacing the function and gradient evaluations in the current point of the search with an average of these quantities for several points in the close vicinity of the current point. This action has the effect of smoothing the likelihood landscape and thus we hoped to overcome the sensitivity to small fluctuations. Although this solution did help us get better results from CG, it did not solve the problem entirely. Moreover, it is not scalable since the number of neighboring points should grow exponentially in the number of parameters if we want to maintain the quality of the smoothing.

Clustering Another optimization technique we explored uses only log-likelihood estimations for multiple points in parameter space. This technique starts from covering a large region and gradually narrows the search to regions of high likelihood (for more background see [Torn and Zilinskas 1989]). In our case we define a starting point and an initial region size and approximate the log-likelihood for a grid of parameter values around the starting point. We then filter the points leaving only a fraction (*e.g.*, 0.1) of the samples that have the highest function values. In the next iteration we look at the *cluster* of selected samples, set the new center to the center of mass of this cluster, and the region of interest is narrowed

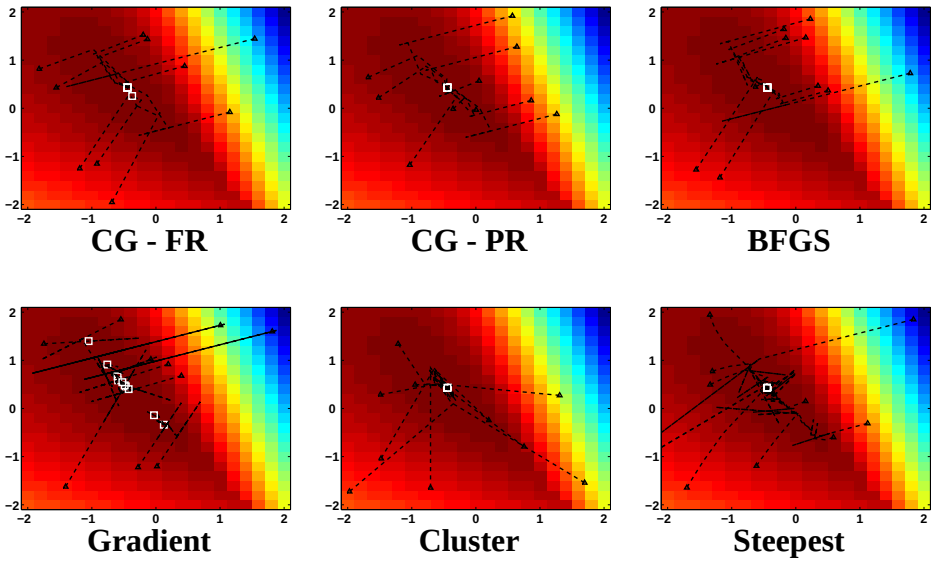


Figure 4.2: Learning trace for various optimization techniques. In each panel we show 10 paths of the steps taken by the parameter estimation algorithm corresponding to 10 random starting points. The bright squares point to the final parameters returned by each iteration of the procedure.

(*e.g.*, 0.85 of its previous size). When the region becomes small enough we terminate the search and return the last center of mass. Of course, the main limitation of this technique is its lack of scalability since the number of sampled points around the center grows exponentially in the number of parameters involved in the search. To alleviate this we can decide to sample a fixed number of points around the center but then the quality of our coverage would deteriorate as the number of parameters grows.

Gradient Size Another approach for this optimization problem has been taken by Sharon and Segal [Sharon and Segal 2007]. They use solely gradient estimations to find optimal parameters. The idea is to proceed in the direction of the gradient and find parameter values for which the norm of the gradient is minimal. We find that this approach suffers from problems of premature stopping of the search resulting in sub-optimal parameters.

Steepest Ascent Finally, we use a simple steepest-ascent algorithm that evaluates the gradient in each point and takes a step in that direction. Steps that result in better function estimates cause the step size to grow, while bad steps reset the step size to some small initial quantity. Function values are recorded along the path and the best value seen is returned at the end. This procedure applies a TABU-like strategy and terminates when the best function value could not be improved for a predefined number of steps. This simple approach overcomes problems of previous methods as it is both scalable and less sensitive to deviations in function and gradient estimates.

To choose the best optimization technique for our problem we used the same toy model as before and conducted a series of experiments in which we start each technique from

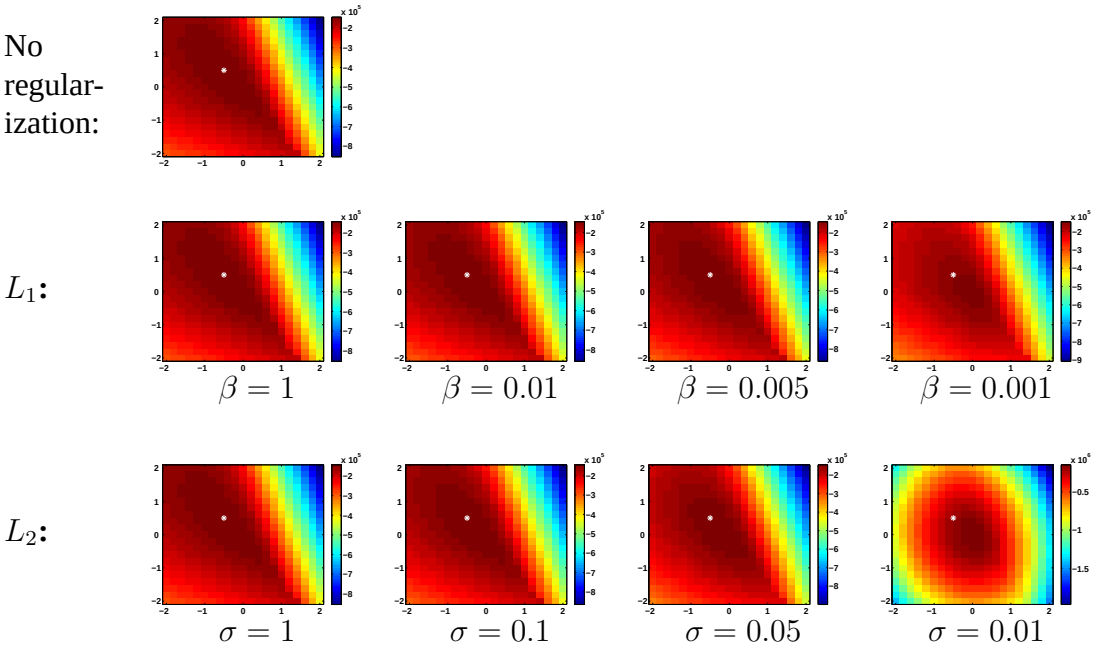


Figure 4.3: Log-likelihood landscape with different regularization terms. Panels visualize the log-likelihood minus regularization term when we vary two parameters: θ_e (y-axis) and θ_t (x-axis).

many (500) random points in parameter space and let it converge. We compare the number of times the global optimum was found, the variance in final parameter values, and running times of the various algorithms (results not shown). These experiments show that the clustering technique (based solely on likelihood evaluations) and the simple steepest ascent algorithm return the optimal parameters most often and have smaller variance than the other methods. The Conjugate Gradient and Quasi-Newton methods run much faster than the other methods, but as mentioned they suffer from premature termination. To conclude, since the clustering approach is not scalable we decided to use the steepest ascent method for the model selection experiments presented bellow. Although this method is less efficient than the CG methods, we chose it because it achieves much better results and is scalable.

As mentioned earlier, MLE can lead to overfitting and regularization is one attempt to alleviate this problem. We now explore the effects of L_1- and L_2-regularization on the likelihood function. We use the same model as before and calculate log-likelihood with CBP using the Bethe approximation. Figure 4.3 shows log-likelihood landscapes for different regularization constants. We can see that as the regularization constant becomes smaller, the penalty term becomes dominant in the regularized log-likelihood and its peak moves closer to $\vec{0}$.

To evaluate the performance of our parameter estimation procedure we need a way to compare the final parameters returned by the learning algorithm to the original parameters used to generate the evidence. We can of course simply compare the parameter values

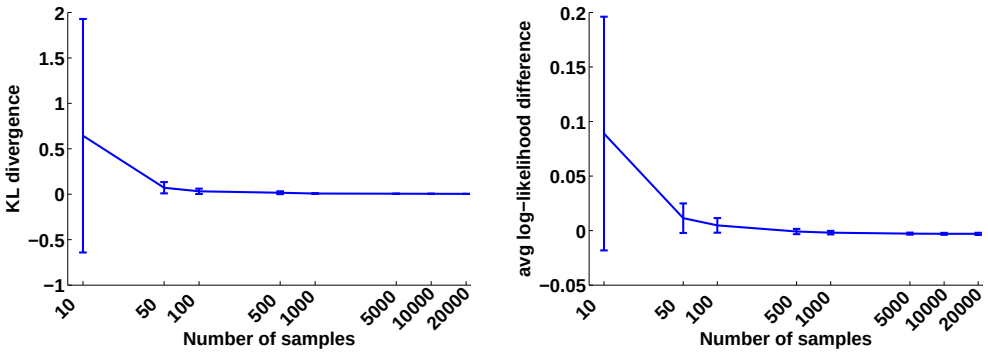


Figure 4.4: Learning curves for parameter estimation. The left plot shows KL-divergence of marginal probabilities as a function of the training sample size, and the right plot shows the same for difference in log-likelihood of a test set averaged over the number of test samples. The mean and standard deviation (shown in error bars) are obtained over 20 parameter estimation trials. KL was measured between estimates of marginal probabilities of the full factors over univariate, triplets and quadruplets of variables. We see nicely how as the number of samples grows we learn a model that is closer to the original model.

to each other, however often different parameters induce similar probability distributions. Therefore, what we are really interested in is comparing the distributions that the parameters induce. Two ways of doing so are comparing marginal probabilities and comparing likelihood estimates for a test set. To compare marginal probabilities we have to measure the distance between two estimates of the joint probability of some subsets of variables. This is naturally done using the Kullback-Leibler divergence (KL), where good parameters should return small KL distance. Here we look at the marginal probabilities defined for: (1) all assignments for 6 variables over 4 graph nodes; (2) all assignments for 3 variables over 3 graph nodes; and (3) the belief over univariate edge. To compare log-likelihood estimates we use a Gibbs sampler to generate evidence for a test set in addition to the training set. We then calculate approximate log-likelihood for the test evidence using both the original parameters and the learned ones, and examine the difference. Here we expect good parameters to have likelihood almost as high as that calculated for the original parameters. Figure 4.4 shows the learning curves for both measurements.

Chapter 5

Feature Selection

In the previous section we assumed that the set of template features is given and focused on parameter estimation. We now drop this assumption and turn to the problem of finding a set of features for a template model given evidence.

1. The Optimization Problem

We view the task of feature selection as an optimization problem. This means that we take a score-based approach in which we define an objective function for different models and then search for a high-scoring model. This approach has been used extensively before [Della Pietra et al. 1997], and here we make the necessary adjustments to make it suitable for symmetric relational MRFs. To formalize this we define $\mathbf{S}$, the universe of all possible *relational MRF schemes* $\mathcal{S}_i$. Given a set of types $\mathcal{T}$ and their attributes $\mathcal{A}$, $\mathcal{S}_i$ is defined by a set of template features $\mathscr{F} = \{\mathcal{F}_1, \ldots, \mathcal{F}_k\}$ over these types and attributes. Our goal, given an objective function $\mathbf{U}$ is to find:

$$\mathcal{S}^* = \max_{\mathcal{S}_i \in \mathbf{S}} \mathbf{U}(\mathcal{S}_i)$$

The straightforward objective function is the likelihood of the training data. Unfortunately, a pure likelihood score is not appropriate here since more complex models will always have higher likelihood. In particular, if $\mathscr{F}_{\mathcal{S}_i} \subseteq \mathscr{F}_{\mathcal{S}_j}$ then $\mathbf{U}_{like}(\mathcal{S}_i) \leq \mathbf{U}_{like}(\mathcal{S}_j)$ [Della Pietra et al. 1997]. Therefore, if we want to use the likelihood function we would have to add further restrictions. There are several ways to choose an appropriate objective function $\mathbf{U}$ and here we focus on three options:

- $\mathbf{U}_{like}(\mathcal{S}_i) = \max_\theta \left(\ell(\boldsymbol{D} : \mathcal{S}_i, \theta)\right)$
- $\mathbf{U}_{BIC}(\mathcal{S}_i) = \max_\theta \left(\ell(\boldsymbol{D} : \mathcal{S}_i, \theta) - \frac{\log(M)}{2}\mathrm{Dim}[\mathcal{S}_i]\right)$
- $\mathbf{U}_{L_1}(\mathcal{S}_i) = \max_\theta \left(\ell(\boldsymbol{D} : \mathcal{S}_i, \theta) - \frac{1}{2\beta}\sum_i |\theta_i|\right)$

where $\mathrm{Dim}[\mathcal{S}_i]$ is the degree of freedom defined by the number of features. We account for redundancy as sometimes adding a feature does not change the degree of freedom. For

example, in case we have features for all assignments to a group of variables we know the same distribution can be described by excluding any one of the features. In such case we reduce $\mathrm{Dim}[\mathcal{S}_i]$ by 1. Of course, there might be more complex dependencies between features, but we do not handle such cases here.

All scores rely on the log-likelihood function possibly adding a penalty term. The BIC score ($\mathbf{U}_{BIC}(\mathcal{S}_i)$) penalizes each degree of freedom by a fixed amount thereby drives the search towards schemes having fewer features [Schwarz 1978]. This score is an approximation to the Bayesian score for model schemes defined as:

$$\int P\left(\boldsymbol{D}|\mathcal{M},\theta\right) P\left(\theta|\mathcal{M}\right) d\theta \tag{5.1}$$

in which we account for our uncertainty about parameters by using a Bayesian prior. Notice that the BIC penalty grows only logarithmically in the number of samples whereas the log-likelihood term grows linearly in that number. This means it has the desired property of inflicting a relatively small penalty when we learn from many samples, reflecting the fact that we trust the value of the likelihood term in that case.

As discussed in Chapter 4, the L_1 objective function ($\mathbf{U}_{L_1}(\mathcal{S}_i)$) has the effect of nullifying parameter values which in turn drives the search towards sparser models. In addition, as we mentioned in Chapter 4, this objective function has a unique global optimum as it adds a linear term to the concave log-likelihood function. Therefore, we can, in theory, entirely avoid the combinatorial problem of feature selection by simply introducing all possible features into the scheme and optimize the parameters relative to the L_1 objective function. The sparsifying effect of L_1 will drive parameters of "weaker" features to 0, practically excluding them from the scheme. Unfortunately this is generally not a good idea for two reasons. First, it might not be feasible in practice since inference on the model constructed over all features might be intractable, and second, even if the set of all features is not too large (in template models this is more likely to happen), it is known that the quality of approximation drops as the number of features increases [Lee et al. 2007]. However, the L_1 objective function has several benefits including: the deletion of previously added features, reduced sensitivity to the order of introduction of features, and a natural stopping criterion (see bellow).

Algorithm 1: findLocalMaximum($\mathcal{S}_0, \boldsymbol{D}, \mathrm{U}$)

```
Data: Initial scheme (S_0), dataset D, score function U
Result: S_i = local − maximum(U(S_i|D))
S_i = S_0 ;
improved = true ;
while improved do
    improved=false ;
    S = getNeighbors(S_i) ;
    forall S_j ∈ S do
        if compareScores(S_i, S_j) then
            S_i = S_j ;
            improved = true ;
            break;
        end
    end
end
return S_i ;
```

One issue to consider when using $\mathbf{U}_{L_1}(\mathcal{S}_i)$ is how to set the meta parameter β. This meta parameter should reflect our preference for sparse models over dense ones. It is obvious that if we take β to bee too small then the penalty term becomes dominant and we end up learning the empty scheme. On the other hand, if we take β to bee too large then the penalty term becomes negligible so we actually calculate $\mathbf{U}_{like}(\mathcal{S}_i)$, which leads to the inclusion of all features in the scheme. Values of β in between these two extremes are interesting. We follow the approach of Lee *et al.* [Lee et al. 2007] and utilize an annealing schedule for β. This means that we start β from a very small value, leading to sparse schemes, and gradually increase it to allow "weaker" features into the scheme. We use cross-validation in order to determine when to stop cooling β - we stop when the test likelihood ceases to improve. Of course, this approach is inappropriate when we want to learn from just a few samples. Unfortunately, this is the case for many interesting real-life problems (see Chapter 6), so in such cases we would have to use either $\mathbf{U}_{like}(\mathcal{S}_i)$ or $\mathbf{U}_{BIC}(\mathcal{S}_i)$, or find another way to set a value for β.

Having selected an objective function for our model scheme it remains to address the optimization problem. Since the universe of all possible $\mathcal{S}_i$'s is exponentially large in the number of features we are willing to consider, we need to devise some efficient way to explore it. A common solution is to use some greedy hill-climbing search in the universe of schemes. We can describe a search in the space of all possible schemes by starting from an initial state $\mathcal{S}_0$ defined by an initial set of template features $\mathscr{F}_0$. Now we consider transitions to other schemes by various changes to the set of features, and this procedure is repeated until some termination condition is met (see Algorithm 1).

No. Nodes	No. Variables	Features
1	0	$\emptyset$
2	1	
3	2	
	3	

Figure 5.1: The set of features $\mathcal{F}$ used in feature selection experiments. The rows contain features with increasing levels of complexity. A broken edge has assignment 0 (not exists) while a full edge as assignment 1 (exists).

2. Incremental Feature Introduction

As mentioned, the size of the search space we are considering is normally prohibitively large so we must apply some heuristic in order to explore it. The simplest approach is to include a feature introduction component which gradually introduces new features to the model. In this approach we maintain two groups of features - an active set and an idle set. At every iteration all features from the idle set are considered for introduction and we move the one which yields the largest gain in score to the active set. Once we decide to add a feature it will never be excluded from the active set. This approach has been used successfully before [Della Pietra et al. 1997].

An alternative approach follows from the fact that each template feature is defined over a set of entities and a list of attributes associated with them (see Definition 4). Thus, we can allow transition between schemes by moving from schemes over small features to schemes over more complex features. This can be done by adding one step of complexity to the template feature, either by enlarging the list of attributes over the same set of entities, or by increasing the number of entities. Practically, in this approach we start from an empty scheme and try to add candidate template features with increasing complexity (in terms of their entities and attributes). Every row in Figure 5.1 contains features from another level of complexity.

3. Stopping Conditions

Both $\mathbf{U}_{BIC}(\mathcal{S}_i)$ and $\mathbf{U}_{L_1}(\mathcal{S}_i)$ give us a convenient stopping criterion for our search algorithm - we can simply stop the search when no feature introduction move is beneficial. In the case of $\mathbf{U}_{L_1}(\mathcal{S}_i)$ taking this approach guarantees convergence to the global optimum as this is a convex optimization problem. As we mentioned before, $\mathbf{U}_{like}(\mathcal{S}_i)$ can only increase when we introduce new features, so we cannot apply such criterion. A common approach in this case is simply to halt the search when the improvement in score does not exceed a certain threshold. Here we take a slightly different approach to this problem and use a statistical test to decide when to terminate the search. To be more specific, we utilize a statistical test called *Likelihood-Ratio Test* to determine if the improvement in likelihood is statistically significant. To understand this we notice that at the end of every feature introduction step we need to compare the likelihoods of the previous and current schemes. If the two schemes are defined by $\mathcal{F}_{\mathcal{S}_i}$ and $\mathcal{F}_{\mathcal{S}_j}$ such that $\mathcal{F}_{\mathcal{S}_i} \subset \mathcal{F}_{\mathcal{S}_j}$ then the likelihood-ratio test is based on the difference $\Lambda = \mathbf{U}_{like}(\mathcal{S}_i) - \mathbf{U}_{like}(\mathcal{S}_j)$ (notice that for exact calculation $\Lambda \leq 0$). As the number of samples approaches ∞ the test statistic -2Λ will be asymptotically χ^2 distributed with degree of freedom equal to the difference in dimension between $\mathcal{S}_i$ and $\mathcal{S}_j$ ($|\mathcal{F}_{\mathcal{S}_j}| - |\mathcal{F}_{\mathcal{S}_i}|$). For finite sample size the distribution is only approximately χ^2, but we can still use this approximation to set a stopping criterion based on the P-value of the likelihood-ratio statistic. The null hypothesis in this test is that $\mathcal{F}_{\mathcal{S}_i}$ is a better model for the given evidence than $\mathcal{F}_{\mathcal{S}_j}$, meaning that it would be a mistake to move from $\mathcal{S}_i$ to $\mathcal{S}_j$. In other words, the Likelihood-Ratio statistic indicates whether the improvement in likelihood is caused by noise in the data or really significant.

4. Parameter Estimation for Feature Selection

Recall that each feature selection step involves parameter estimation. Several strategies can be employed to handle this and we now consider some of them.

- **AtOnceZero**: A simple parameter estimation scheme is to find θ^{MLE} starting from $\theta = \vec{0}$ using one of the optimization algorithms described in Chapter 4. In this case all parameters are allowed to move freely until convergence. In other words, for each candidate feature we start a new search in parameter space that does not use information from previous iterations.
- **AddPrev**: An alternative approach is to start parameters from their previously learned values and start only the parameter associated with the newly introduced candidate feature from 0. This approach assumes that parameters learned for simpler models can serve as a good starting point when learning parameters for more complex models.
- **AddFix/Free**: In a variant of this approach we fix previously learned parameters to their learned values and allow only the new $\theta_{\mathcal{F}}$ to converge freely. We can either terminate here (**AddFix**) or use the recent parameter values as a starting point for a

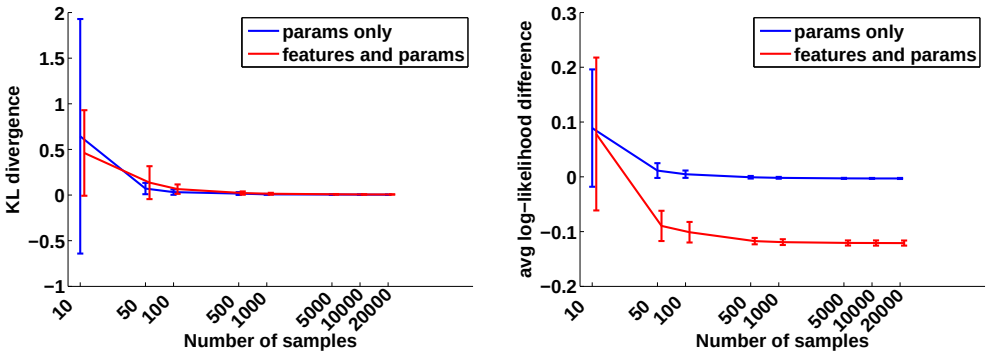


Figure 5.2: Learning curves for parameter estimation when the correct set of features is known vs. complete model learning which includes both feature selection and parameter estimation. As before, The left plot shows KL-divergence of marginal probabilities as a function of the training sample size, and the right plot shows the same for difference in log-likelihood of a test set averaged over the number of test samples.

new search where all parameters are free to converge (**AddFree**) [Della Pietra et al. 1997].

5. Experimental Results

To evaluate our feature selection approach we use a synthetic setup similar to the one we used for parameter estimation in Chapter 4. We take our simple toy model with features over univariate ($\mathcal{F}_e$) and closed-triangles ($\mathcal{F}_t$), and set its parameters to some arbitrary values ($\theta_e = 0.5$, $\theta_t = -0.5$). We use a Gibbs sampler to produce train and test evidence and use the train evidence to learn a template model (features and parameters). Here we define $\mathscr{F}$ to consist of all template features over up to 3 graph vertices (Figure 5.1). Finally, we compare the learned model to the original one using KL of some marginal probabilities and test-set likelihood as we discussed above. We stress again that such large-scale experiments require hundreds of thousands of inference steps and are therefore only possible when the inference calculation is extremely efficient, as in our algorithm.

We begin the evaluation by comparing the learning curve of parameter estimation alone (assuming we have the correct set of features) and complete model learning. Figure 5.2 shows the two learning curves for KL divergence over marginal probabilities and difference in log-likelihood on a test set. Encouragingly, the result shows that we can learn the feature set from evidence in a satisfying way as the learning curve of feature selection is not worse than that of parameter estimation alone.

Recall that we use two measures to evaluate the learned model: KL divergence of some marginals and difference in log-likelihood of test evidence. Unfortunately, the second measure we described, namely the difference in log-likelihood, is solely dependent on the approximation of the partition function (in case we learn from fully observed data). It is common knowledge in the field that the approximation of the partition function is less

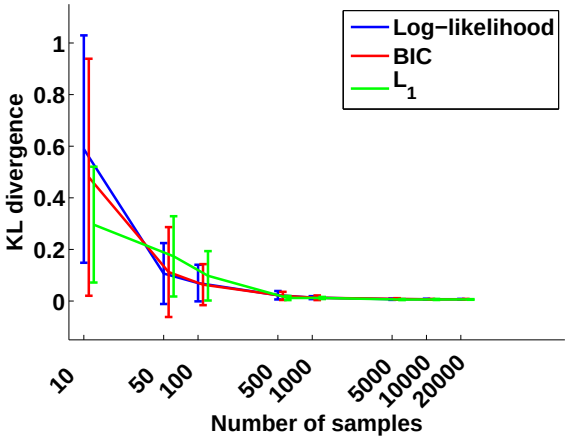


Figure 5.3: Comparison of the three objective functions used for feature selection: $\mathbf{U}_{like}(\mathcal{S}_i)$, $\mathbf{U}_{BIC}(\mathcal{S}_i)$ and $\mathbf{U}_{L_1}(\mathcal{S}_i)$.

reliable than the approximation of the marginals. Specifically, in Figure 5.2 it can be seen that the approximate log-likelihood calculated for the original model (which was used to generate the train and test data) is lower than the approximate likelihood of the model that we learned. Calculating the exact log-likelihood for this case verified that the log-likelihood approximation for the learned model was in fact inaccurate. This inaccuracy was also present for standard asynchronous BP so it was not an artifact of our algorithm but rather a shortcoming of the Bethe approximation. Therefore from here on we show only learning curves based on KL divergence between marginals and make a note to address this issue in future work (see Chapter 7).

In Figure 5.3 we show a comparison of the three objective functions. One can see that for a small sample size $\mathbf{U}_{L_1}(\mathcal{S}_i)$ using an annealing schedule for the meta parameter β gives the best results. The other objective functions, $\mathbf{U}_{like}(\mathcal{S}_i)$ and $\mathbf{U}_{BIC}(\mathcal{S}_i)$, have very close learning curves. Since the likelihood term in all scores grows linearly in the number of samples, for large samples this term becomes dominant and we effectively compute $\mathbf{U}_{like}(\mathcal{S}_i)$ for all scores. This is evident in the plot as we get similar score for all 3 methods.

Figure 5.4 shows a comparison between the two approaches we discussed for incremental feature introduction. As expected, the flat search, where we consider all idle features at every step, performs better. This is not surprising since it contains the search by levels as a special case. Of course, the running time of the flat search is much longer than the levels variant, so there is a trade-off here between computation time and quality of solution.

Figure 5.5 shows the performance of the various parameter estimation schemes. We see that for large sample size all methods perform equally well, while for small sample size it seems better to start parameter estimation from $\vec{0}$ without using information from previous steps. Fixing values of learned features without setting them free later seems as the worse choice.

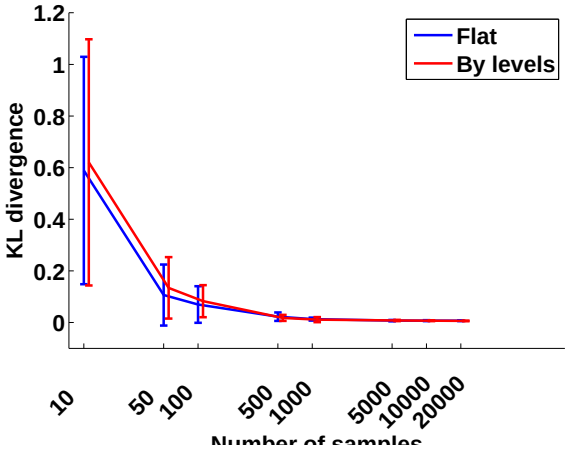


Figure 5.4: Comparison of learning curves for the two variants of the feature introduction component. The "Flat" curve shows the result when we consider all idle features for addition at every step, while "By levels" shows results when we only consider introduction of features of a certain level of complexity.

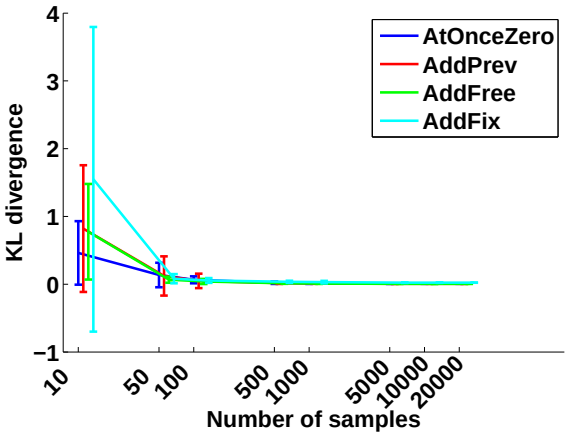


Figure 5.5: Comparison of the various ways to initialize parameter values at the beginning of a feature selection step. "AtOnceZero" means we initialize all parameters from 0, in "AddPrev" we use the recent learned values as starting point for the new search, "AAddFree" we initially freeze values of learned parameters letting only the new parameter converge and the free all to converge from that point. Lastly, in "AddFix" we do not allow the second step of "AddFree".

From these results we conclude that the best performance for the synthetic example we have chosen is achieved with the likelihood-gain method ($\mathbf{U}_{like}$) searching in the flat set of idle features and starting the parameters from $\vec{0}$ at every iteration. We now conduct further experiments to better understand the learning process.

It is interesting to see in which order features are added to the scheme and how close the final model is to the original one. We use the same $\mathscr{F}$ as before and then follow the

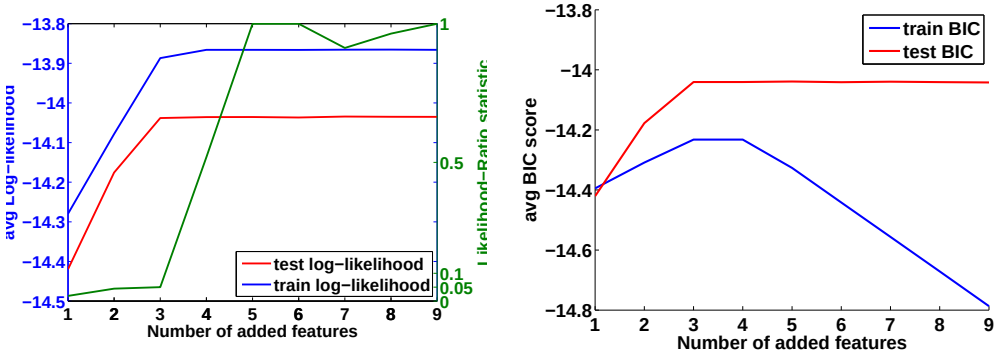


Figure 5.6: Figure (a) shows the change in log-likelihood (averaged over the number of samples) when gradually introducing more features into the scheme. Figure (b) shows the change in BIC score as a function of the number of added features.

Original Model	**Learned Model**
$\mathcal{F}_e$ with $\theta_e = 0.3$ $\mathcal{F}_t$ with $\theta_t = -0.6$	$\mathcal{F}_e$ with $\theta_e = 0.18$ $\mathcal{F}_t$ with $\theta_t = -0.195$ $\mathcal{F}_{star2}$ with $\theta_{star2} = -0.116$

Table 5.1: Original vs. learned model in synthetic experiment.

feature selection procedure to see which feature is added at every step and how it effects the objective function and the likelihood of the test evidence. We use a training set consisting of 10 samples and a test set consisting of $5K$ samples. Figure 5.6 (a) shows that as the gain in likelihood reduces, the Likelihood-Ratio statistic is no longer significant (> 0.05). We can also see in Figure 5.6 (b) how the BIC score reduces when we include more features in the scheme. This does not happen in the test score since the test set contains $5K$ samples so the penalty term inflicted by BIC becomes small relative to the log-likelihood term. We note that the feature introduction scheme we use, in which all features are eventually included in the learned model, is guaranteed to lead to over-parametrization, which means that some features can be excluded as they can be described by a combination of other features. Since this is done for a didactic purpose we ignore this issue here.

We note that we expected the average log-likelihood of the test set to drop when too many features are added to the model, reflecting overfitting to the training data. Surprisingly we see that the likelihood of the test set remains at the same level even though the model becomes more and more complex. One possible explanation for this might be that it is another reflection of the problem in the approximation of the likelihood.

From this experiment we see that although we used only 2 features in the original model ($\mathcal{F}_e$ and $\mathcal{F}_t$ with $\theta_e = 0.3$ and $\theta_t = -0.6$), we actually end up learning a model consisting of 3 features. If we look at the order of feature addition we find that $\mathcal{F}_e$ is the first feature to be admitted to the scheme since its improvement over the empty model is the largest. Second, we add $\mathcal{F}_t$, and finally we add the "star2" feature which is composed of two edges having a mutual vertex in the undirected graph (rightmost feature in second row of Figure 5.1). So we have that the first two features we include in the model are the ones we used in the original model. Table 5.1 compares the original and learned models.

Chapter 6

Learning with Real-life Evidence

To demonstrate the power of our method We now proceed to learning a model over a real-life domain of interactions between proteins (PPI). We build on a simplified version of the model described in Jaimovich *et al.* [Jaimovich et al. 2006] for protein-protein interactions. This model is analogous to our running example, where the vertices of the graph are proteins and the edges are interactions. We define the basic type T_p for proteins and the complex type $T_i = [T_p, T_p]$ for interactions between proteins. As with edges, we consider the template attribute $\mathsf{X}_e(T_i)$ that equals one if the two proteins interact and zero otherwise. We reason about an instantiation for a set of 813 proteins related to DNA transcription and repair [Collins et al. 2007b]. We collected statistics over interactions between these proteins from various experiments [Mewes et al. 1998; Gavin et al. 2006; Krogan et al. 2006; Collins et al. 2007a].

Using the methods described above we learn a generative rMRF for this PPI network. The set of features we consider here consists of all features defined over upto 4 proteins which are connected and have "all-1" assignment. Figure 6.1 shows all 9 features. Our objective function for feature selection is $\mathbf{U}_{like}$, and we use the Likelihood-Ratio test as a stopping criterion. We use the "Flat" variant of feature introduction, meaning that at every step we consider all idle feature for introduction and choose the one with highest likelihood gain. Finally, we start every parameter estimation run from $\theta = \vec{0}$ and let all parameter move freely until convergence.

Table 6.1 shows the learned model at the end of every feature introduction step. We see that the first feature to be admitted to the model is the single-interaction feature ("Pair"), which is added with a large negative parameter since the network of interactions is rather sparse (1672 interactions). The second feature to join the model is the ring of size 4 ("Ring4"), which is added with a small negative parameter, while θ_{Pair} becomes positive. According to the Likelihood-Ratio test we should have stopped the search then. Instead, we let it run some more and check what are the next models produced by the search. We see that the next feature to be included is "Triangle", but as we said the improvement in likelihood is no longer significant (P-value > 0.05). It seems as the parameter θ_{Ring4} remains unchanged while θ_{Pair} is split in two and shared with the newly added feature. At

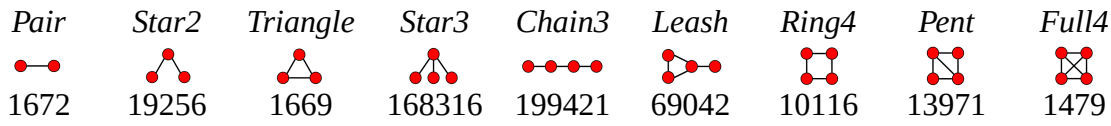


Figure 6.1: The set of features $\mathcal{F}$ used in feature selection for the PPI network. We show the name we assign to the feature, its graphical representation, and the number of occurrences it has in the PPI evidence.

Feature selection step	**Added feature**	**Parameter values**	**Likelihood-Ratio statistic**	**Log-Likelihood**
1		$\theta_{Pair} = -5.28$	0	-10504.8
2		$\theta_{Pair} = 0.414$ $\theta_{Ring4} = -0.009$	0	-1.04421
3		$\theta_{Pair} = 0.209$ $\theta_{Ring4} = -0.009$ $\theta_{Triangle} = 0.207$	0.16	-0.0606451
4		$\theta_{Pair} = 0.306$ $\theta_{Ring4} = -0.00137$ $\theta_{Triangle} = 0.269$ $\theta_{Full4} = -0.029$	1	-0.592723
⋮				

Table 6.1: First 4 steps of model selection for the PPI problem.

the next step the "Full4" feature is added, but here the approximate likelihood no longer improves (due to the approximation it is actually lower), so the remaining of the search is less interesting to follow.

We defer the interpretation of these results for future work and proceed to discuss several points that arise from this work.

Chapter 7

Discussion

1. Contribution

We have presented a powerful method for learning probabilistic models for structured relational domains. This method relies on a lifted inference algorithm that operates in the template-level of the relational model. Specifically, we have shown how we exploit symmetry in relational MRFs to perform lifted approximate inference at the template-level model. This results in an extremely efficient approximate inference procedure. We have shown that this procedure is equivalent to synchronous belief propagation in the ground model. We have also empirically shown that on small graphs our inference algorithm approximates the true marginal probability very well. Furthermore, other approximation methods, such as MCMC yield inference results that are similar to ours on larger graphs. Note that other works show that synchronous and asynchronous belief propagation are not always equivalent [Elidan et al. 06]. The key limitation of our procedure is that it relies on the lack of evidence. Once we introduce evidence the symmetry is disrupted and our method does not apply. While this seems to be a serious limitation, we notice that inference without evidence is the main computational step in learning such models from fully observed data. We showed how this procedure enables us to deal with learning problems in large relational models that were otherwise infeasible.

We mentioned that previous works on lifted inference focused on exact inference via variable elimination or caching intermediate calculations for ground entities to be used by other entities originating from the same template-level entity [Poole 2003; de Salvo Braz et al. 2005; Pfeffer et al. 1999]. In many practical cases, when the tree-width is large, exact inference is infeasible even in the template level. Our method is the first to provide template-level approximate inference with run-time that is independent on the size of the instantiated model.

Using the efficiency of our method we are able to repeat the learning procedure many times. We use this advantage to conduct a survey of different techniques for the various stages involved in model selection. Specifically, we compared several optimization algorithms for parameter estimation, we compared two strategies for feature introduction, we compared different ways to initialize parameters in feature selection, and we compared sev-

eral model scoring functions to be optimized in the search. The main insight we gain from this survey is that commonly used optimization techniques that play an important role in the search for θ^{MLE}, such as Conjugate Gradient and BFGS, encounter difficulties when tackling the log-likelihood landscape. In particular, since this landscape tends to have long and narrow ridges many of the common techniques halt the search prematurely with sub-optimal parameters. We find that utilizing a TABU-like steepest ascent algorithm achieves much better results as it is able to cross such ridges in many cases.

For the first time in this context, we employ a statistical test to be used as a stopping condition for the likelihood based score for model selection. Specifically, we show in synthetic experiments that using the Likelihood-Ratio test is useful for stopping the search after the important features have been included in the model and before the model overfits the training data.

2. Limitations

Some of our empirical experiments indicate that our approximation of the log-likelihood function might be inaccurate. Specifically, we get that the test-set likelihood using the learned model is higher than that of the original model that was used to produce the data. Comparing to exact likelihood calculation on a small graph we verify that, indeed, the log-likelihood calculated for the learned model is very different from the exact log-likelihood of this model. Moreover, we made sure that this inaccuracy was not introduced by our algorithm, but rather was a limitation of the standard BP approximation. We note that our method for model selection relies heavily on the likelihood function and that we should address this issue in future work. An alternative objective function that might be suitable in this case is pseudo-likelihood [Besag 1975].

Trying to apply our compact approximate inference to Generalized Belief Propagation yields poor results. A short investigation revealed that the template region graph we have built was not maxent-normal, meaning that it assigned higher entropy to non-uniform assignments than to the uniform one. Such graphs have been shown previously to give poor approximations [Yedidia et al. 2004]. Therefore, we decided to focus on standard Belief Propagation in this work. Alternatively, we can try to think of a way to build template region graphs that are maxent-normal and therefore more likely to perform well.

As mentioned earlier, our method is not applicable when evidence is provided. Reasoning with partial evidence is an important inference task and it would be very useful to handle it in a lifted inference framework. Unfortunately, we have yet to advance in this direction.

3. Other Issues

The Hessian matrix is the matrix of second derivatives. In the context of MRFs this matrix plays a role in several aspects of the learning problem. First, it can be used for parameter estimation with Newton's method instead of the first derivative. Second, it is used in the

penalty term of the Laplace score for model selection. The Laplace score is another way to approximate the Bayesian score (Eq. (5.1)).

It can be easily shown that the second derivative of the log-likelihood function is given by the Covariance matrix:

$$\begin{aligned}\frac{\partial \ell(\boldsymbol{D})}{\partial \theta_i \partial \theta_j} &= -M\mathbf{Cov}_\theta[\mathcal{F}_i;\mathcal{F}_j] \\ &= -M\left(\boldsymbol{E}_\theta\left[\mathcal{F}_i\mathcal{F}_j\right] - \boldsymbol{E}_\theta\left[\mathcal{F}_i\right]\boldsymbol{E}_\theta\left[\mathcal{F}_j\right]\right)\end{aligned}$$

By the way, this proves the concavity of the likelihood function since the Covariance matrix is positive semi definite.

To compute the Hessian we must calculate the joint expectation of all pairs of features. In general this is a very expensive task and often intractable, however, in symmetric rMRFs such as the ones we study it can be much cheaper as we can do it in the template level - considering pairs of template features (and their number is often not too large). We have yet to address this issue, but it seems an interesting direction since we might find a better parameter estimation method or a better objective function for model selection.

In this work we applied our approach to BP and GBP. It might be possible to apply the same idea to other approximate inference methods. We have not thought about this thoroughly yet, but the variational methods of *Mean Field* [Jordan et al. 1998] and *Expectation Propagation* [Minka 2001] seem like good candidates for starting this expansion.

4. Applications

To conclude this section we now discuss possible applications of our new method. The immediate application we intend to try is learning generative models for a variety of networks from different domains. In this work we have shown its use for a protein interaction network, and the same methodology can be applied to other undirected networks. In addition, we plan to handle in a similar manner several directed networks. The models we learn can shed new light on the characteristics of these networks, revealing local rules that govern their global structure.

As mentioned in Chapter 1, one of the prominent works in this field is Network Motifs that looks for overly abundant subgraphs [Milo et al. 2002]. Recall that such subgraphs are found to be over-represented with respect to a random ensemble of networks that preserves some of the properties of the original network. However, this approach has been criticized since over-representation turns out to be highly dependent on the qualities that are chosen to be preserved in the random ensemble [Artzy-Randrup et al. 2004]. We believe that our approach is more elegant as we assume less about the structure of the underlying network. Furthermore, since rMRFs are very expressive we can learn richer models. Such models can incorporate additional information about nodes and edges, and we can even use Chain Networks models that combine undirected and directed potentials instead of MRFs, as suggested in Jaimovich *et al.* [Jaimovich et al. 2006]. This way we could go beyond bare networks and find more complex rules that apply in large domains.

Finally, since our approach is applicable whenever the factor graph has the local neighborhood property we can use it in other domains that obey this constraint. Specifically, we already mentioned that the square-wrapped-around-lattice has this property, and the same is true for infinite MRFs that are defined by repeated local features [Singla and Domingos 07]. In such infinite models our method is a natural choice as it is independent on the size of the ground model, but only on the template-level scheme.

All of these applications could bring us one step closer towards successful modeling of complex networks using relational probabilistic models.

Acknowledgements

I am grateful to Nir Friedman and Ariel Jaimovich for coming up with the idea behind this work, and for guiding me paitently through the long way it took to make it work in practice. I am also grateful to the members of Nir's group that have devoted time in several occasions to hear and comment about my research (although all they really care about is biology :). Specifically, I want to thank Tommy Kaplan, Naomi Habib, Moran Yassour, Tal El-Hai and Matan Ninio. I also acknowledge Chen Yanover, Tal El-Hay and Gal Elidan for their useful comments on previous versions of this manuscript. In addition, I want to thank Nir for his generous funding and also the Rudin Foundation and the Liss Fellowship for their vital financial support. And most of all I would like to thank Avital for helping me get through all this and for not dumping me despite my long working hours.

Bibliography

Y. Artzy-Randrup, S. J. Fleishman, N. Ben-Tal, and L.Stone. Comment on “network motifs: Simple building blocks of complex networks” and “superfamilies of evolved and designed networks”. *Science*, 305:1107, 2004.

A. L. Barabasi and R. Albert. Emergence of scaling in random networks. *Science*, 286: 509–512, 1999.

J. E. Besag. Statistical analysis of non-latice data. *The Statistician*, 24:179–195, 1975.

H. A. Bethe. Statistical theory of superlattices. *Proc. Roy. Soc. London*, 150:552, 1935.

S. F. Chen and R. Rosenfeld. A survey of smoothing techniques for me models. *IEEE Trans. on Speech and Audio Processing*, 8:37–50, 2000.

S. R. Collins, P. Kemmeren, X. C. Zhao, J. F. Greenblatt, F. Spencer, F. C. Holstege, J. S. Weissman, and N. J. Krogan. Towards a comprehensive atlas of the physical interactome of saccharomyces cerevisiae. *Mol Cell Proteomics*, 2007a.

S. R. Collins, K. M. Miller, N. L. Maas, A. Roguev, J. Fillingham, C. S. Chu, M. Schuldiner, M. Gebbia, J. Recht, M. Shales, H. Ding, H. Xu, J. Han, K. Ingvarsdottir, B. Cheng, B. Andrews, C. Boone, S. L. Berger, P. Hieter, Z. Zhang, G. W. Brown, C. J. Ingles, A. Emili, C. D. Allis, D. P. Toczyski, J. S. Weissman, J. F. Greenblatt, and N. J. Krogan. Functional dissection of protein complexes involved in yeast chromosome biology using a genetic interaction map. *Nature*, 2007b.

R. de Salvo Braz, E. Amir, and D. Roth. Lifted first-order probabilistic inference. In *IJCAI ’05*, pages 1319–1325, 2005.

S. Della Pietra, V. Della Pietra, and J. Lafferty. Inducing features of random fields. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 19(4):380–393, 1997.

M. Dudik, S. J. Phillips, and R. E. Schapire. Maximum entropy density estimation with generalized regularization and an application to species distribution modeling. *Journal of Machine Learning Research*, 8:1217–1260, 2007.

G. Elidan, I. McGraw, and D. Koller. Residual belief propagation: Informed scheduling for asynchronous message passing. In *Proc. Twenty Second Conference on Uncertainty in Artificial Intelligence (UAI '06)*, 06.

R. Fletcher. *Practical Methods of Optimization (Second Edition)*. Wiley, 1987.

N. Friedman, L. Getoor, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In *IJCAI '99*, pages 1300–1309. 1999.

A. C. Gavin, P. Aloy, P. Grandi, R. Krause, M. Boesche, M. Marzioch, C. Rau, L. J. Jensen, S. Bastuck, B. Dumpelfeld, A. Edelmann, M. A. Heurtier, V. Hoffman, C. Hoefert, K. Klein, M. Hudak, A. M. Michon, M. Schelder, M. Schirle, M. Remor, T. Rudi, S. Hooper, A. Bauer, T. Bouwmeester, G. Casari, G. Drewes, G. Neubauer, J. M. Rick, B. Kuster, P. Bork, R. B. Russell, and G. Superti-Furga. Proteome survey reveals modularity of the yeast cell machinery. *Nature*, 440(7084):631–636, Mar 2006.

S. Geman and D. Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, pages 721–741, 1984.

L. Getoor, N. Friedman, D. Koller, and B. Taskar. Learning probabilistic models of relational structure. In *Eighteenth International Conference on Machine Learning (ICML)*. 2001.

J. Hammersley and P. Clifford. Markov fields on finite graphs and lattices. Unpublished manuscript, 1971.

A. Jaimovich, G. Elidan, H. Margalit, and N. Friedman. Towards an integrated protein-protein interaction network: a relational Markov network approach. *J. Comut. Biol.*, 13: 145–164, 2006.

M. I. Jordan, Z. Ghahramani, T. Jaakkola, and L. K. Saul. An introduction to variational approximations methods for graphical models. In M. I. Jordan, editor, *Learning in Graphical Models*. Kluwer, Dordrecht, Netherlands, 1998.

R. Kikuchi. A theory of cooperative phenomena. *Phys. Rev.*, 81:988–1003, 1951.

N. J. Krogan, G. Cagney, H. Yu, G. Zhong, X. Guo, A. Ignatchenko, J. Li, S. Pu, N. Datta, A. P. Tikuisis, T. Punna, J. M. Peregrin-Alvarez, M. Shales, X. Zhang, M. Davey, M. D. Robinson, A. Paccanaro, J. E. Bray, A. Sheung, B. Beattie, D. P. Richards, V. Canadien, A. Lalev, F. Mena, P. Wong, A. Starostine, M. M. Canete, J. Vlasblom, S. Wu, C. Orsi, S. R. Collins, S. Chandran, R. Haw, J. J. Rilstone, K. Gandi, N. J. Thompson, G. Musso, P. St Onge, S. Ghanny, M. H. Lam, G. Butland, A. M. Altaf-Ul, S. Kanaya, A. Shilatifard, E. O'Shea, J. S. Weissman, C. J. Ingles, T. R. Hughes, J. Parkinson, M. Gerstein, S. J. Wodak, A. Emili, and J. F. Greenblatt. Global landscape of protein complexes in the yeast Saccharomyces cerevisiae. *Nature*, 440(7084):637–643, Mar 2006.

F. R. Kschischang, B. J. Frey, and H. A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47, 2001.

S. Lee, V. Ganapathi, and D. Koller. Efficient structure learning of markov networks using l_1-regularization. In *Advances in Neural Information Processing Systems 16*, Cambridge, Mass., 2007. MIT Press.

HW Mewes, J Hani, F Pfeiffer, and D Frishman. MIPS: a database for genomes and protein sequences. *Nucleic Acids Research*, 26:33–37, 1998.

R Milo, S Shen-Orr, S Itzkovitz, N Kashtan, D Chklovskii, and Alon U. Network motifs: simple building blocks of complex networks. *Science*, 298:824–7, 2002.

T. P. Minka. Expectation propagation for approximate Bayesian inference. In *Proc. Seventeenth Conference on Uncertainty in Artificial Intelligence (UAI '01)*, pages 362–369, 2001.

K. P. Murphy, Y. Weiss, and M. I. Jordan. Loopy belief propagation for approximate inference: an empirical study. In *Proc. Fifthteenth Conference on Uncertainty in Artificial Intelligence (UAI '99)* **?**.

S. Parise and M. Welling. Learning in markov random fields: An empirical study. In *Joint Statistical Meeting*, 2005.

J. Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, 1988.

A. Pfeffer, D. Koller, B. Milch, and K. Takusagawa. SPOOK: A system for probabilistic object-oriented knowledge representation. In *Proc. Fifthteenth Conference on Uncertainty in Artificial Intelligence (UAI '99)* **?**, pages 541–550.

D. Poole. First-order probabilistic inference. In *IJCAI '03*, pages 985–991, 2003.

M. Richardson and P. Domingos. Markov logic networks. *ML*, 62:107–136, 2006.

G. Schwarz. Estimating the dimension of a model. *The Annals of Statistics*, 6:461–464, 1978.

E. Segal, M. Shapira, A. Regev, D. Pe'er, D. Botstein, D. Koller, and N. Friedman. Module networks: identifying regulatory modules and their condition-specific regulators from gene expression data. *Nat Genet*, 34(2):166–176, Jun 2003.

E. Sharon and E. Segal. A feature-based approach to modeling protein-dna interactions. In *Eleventh Inter. Conf. on Research in Computational Molecular Biology (RECOMB)*, 2007.

P. Singla and P. Domingos. Markov logic in infinite domains. In *Proc. Twenty Third Conference on Uncertainty in Artificial Intelligence (UAI '07)*, 07.

B. Taskar, A. Pieter Abbeel, and D. Koller. Discriminative probabilistic models for relational data. In *Proc. Eighteenth Conference on Uncertainty in Artificial Intelligence (UAI '02)*, pages 485–492, 2002.

B. Taskar, M. F. Wong, P. Abbeel, and D. Koller. Link prediction in relational data. In *Advances in Neural Information Processing Systems 16*, Cambridge, Mass., 2004. MIT Press.

R. Tibshirani. Regression shrinkage and selection via the lasso. *J. Royal. Statist. Soc B*, 1996.

A. A. Torn and A. Zilinskas. *Global optimization, Lecture Notes in Computer Science 350*. Springer-Verlag, 1989.

M. Welling. On the choice of regions for generalized belief propagation. In *Proc. Twentieth Conference on Uncertainty in Artificial Intelligence (UAI '04)*, 2004.

P. M. Williams. Bayesian regularization and pruning using a laplace prior. *Neural Computation*, 7:117–143, 1995.

J. Yedidia, W. Freeman, and Y. Weiss. Constructing free energy approximations and generalized belief propagation algorithms. Technical Report TR-2002-35, Mitsubishi Electric Research Labaratories, 2002.

J. Yedidia, W. Freeman, and Y. Weiss. Constructing free energy approximations and generalized belief propagation algorithms. Technical Report TR-2004-040, Mitsubishi Electric Research Labaratories, 2004.