University of California

Los Angeles

Decomposition Techniques for Learning Graphical Models

A dissertation submitted in partial satisfaction

of the requirements for the degree

Doctor of Philosophy in Computer Science

by

Khaled Refaat

2015

Abstract of the Dissertation

Decomposition Techniques for Learning Graphical Models

by

Khaled Refaat

Doctor of Philosophy in Computer Science

University of California, Los Angeles, 2015

Professor Adnan Youssef Darwiche, Chair

Probabilistic graphical models are ubiquitous tools for reasoning under uncertainty that have been useful to many fields. Despite their importance, learning these models from incomplete data remains a challenge, due to the high non-convexity of the corresponding optimization problem. Iterative algorithms, such as Expectation Maximization (EM), are typically used for learning from incomplete data, yet these approaches tend to exhibit behaviors that are independent of the degree of incompleteness in the data. We argue in this thesis that the degree of incompleteness is a main indicator of the difficulty of a learning problem. As such, we investigate a number of learning approaches, which are driven and motivated by this degree. In particular, we show that by exploiting certain patterns in the dataset, the learning problem can be decomposed into smaller and independent learning problems, which can lead to orders-of-magnitude speed-up in learning time. Moreover, we propose a new class of algorithms for learning graphical models, whose learned parameters and running time improve as the data becomes less incomplete.

The dissertation of Khaled Refaat is approved.

University of California, Los Angeles

2015

To my mother…

Table of Contents

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

First and foremost, I would like to express my sincere gratitude to my advisor, Adnan Darwiche, who worked hard with me throughout my PhD, giving me freedom when appropriate and guidance when necessary. I much appreciated the freedom he gave me to spend time studying all the mathematical topics I am curious about, even though many of them were not directly related to our research. I learned from Adnan how to turn theory to practical algorithms, and how to present my work. Adnan set the bar very high and always helped get our research in the best possible form. I always felt that Adnan was a key member in the team before being an advisor.

I was fortunate to work with Arthur Choi throughout my PhD. I always enjoyed discussing research problems with him. I found Arthur to be a valuable resource to our research team.

I was lucky to have an outstanding PhD committee including: David Heckerman, Lieven Vandenberghe, Richard Korf, and Stott Parker. Lieven has introduced me to the field of optimization and has indirectly contributed to the work in this thesis.

I would also like to thank all the great people I worked with at Google since 2011 in Santa Monica, Mountain View, Zurich, and London; and especially my outstanding mentors. Furthermore, I am very grateful to my masters advisor, Amir Atiya, for the great things he taught me throughout the years.

Last but not least, I would like to thank my family and friends for their endless support.

VITA

2015	Northrop-Grumman Outstanding Research Award.
2010–2015	Research Assistant, Automated Reasoning Group, UCLA.
2012–2015	Teaching Assistant, Computer Science Department, UCLA.
2011-2014	Research Intern, Google Research.
2010–2012	MSc: University of California, Los Angeles, USA.
2007–2010	MSc: Cairo University, Egypt.
2007–2010	Teaching Assistant, Cairo University, Egypt.
2009	Research Intern, LAAS-CNRS, Toulouse, France.
2007–2009	Research Engineer, IBM, Egypt.
2003–2007	BSc: Cairo University, Egypt.

PUBLICATIONS

Khaled S. Refaat, Adnan Darwiche, An Upper Bound on the Global Optimum in Parameter Estimation. Conference on Uncertainty in Artificial Intelligence (UAI 2015).

Khaled S. Refaat, Adnan Darwiche, Data Compression for Learning MRF Parameters.

International Joint Conference on Artificial Intelligence (IJCAI 2015).

Khaled S. Refaat, Arthur Choi, Adnan Darwiche, Decomposing Parameter Estimation Problems. Advances in Neural Information Processing Systems (NIPS 2014).

Khaled S. Refaat, Arthur Choi, Adnan Darwiche, EDML for Learning Parameters in Directed and Undirected Graphical Models. Advances in Neural Information Processing Systems (NIPS 2013).

Khaled S. Refaat, Sugato Basu, Deirdre O'brien, Liadan O'Callaghan, Large-Scale Query Understanding. Big Learning Workshop (NIPS 2012), Lake Tahoe, Nevada, USA, 2012.

Khaled S. Refaat, Arthur Choi, Adnan Darwiche, New Advances and Theoretical Insights into EDML. Proceedings of the 28th Conference on Uncertainty in Artificial Intelligence (UAI 2012), Catalina Island, USA, 2012.

Khaled S. Refaat, Arthur Choi, Adnan Darwiche, EDML: A Method for Learning Parameters in Bayesian Networks. Proceedings of the 27th Conference on Uncertainty in Artificial Intelligence (UAI 2011), Barcelona, 2011.

Khaled S. Refaat, Pierre-Emmanuel Hladik, Efficient Stochastic Analysis of Real Time Systems via Random Sampling. Proceedings of the 22nd Euromicro Conference on Real-Time Systems (ECRTS 2010), Brussels, 2010, pp. 175-183.

CHAPTER 1

Introduction

The world is full of uncertainty. Computer systems in many applications need to take decisions and reason under uncertainty. For example, an autonomous robot exploring Mars may encounter unexpected obstacles, and may need to take critical decisions while partially observing the environment.

Probability theory has been around for centuries and can be used to effectively provide answers to probabilistic questions. However, storing probability distributions, which are required to answer these probabilistic questions, can be problematic for real-world problems. For example, in an application where we have 100 binary random variables, the computer needs to store 2^{100} numbers. This calls for an efficient way to store probability distributions compactly, and do automated reasoning directly using such compact representations.

Fortunately, in some cases, there is a way to compactly store a distribution in the form of a graphical model, which exploits conditional independence assumptions, and stores the distribution as a graph structure associated with some probabilities, called parameters. The number of parameters required to populate the graph structure can be modest compared to the size of the represented probability distribution.

Indeed, graphical models are powerful tools for reasoning under uncertainty, and have been used in many fields including computer science, cognitive science, statistics, and philosophy. In particular, graphical models have benefited applications in computer vision, bioinformatics, natural language processing, and statistical physics; for example see (Li, 2001; Yanover, Schueler-Furman, & Weiss, 2007; Lafferty, McCallum, &

Pereira, 2001; Marinari, Parisi, & Ruiz-Lorenzo, 1997).

The creation of useful graphical models for real-world problems has been a challenge. Domain experts crafting models and parameters usually fail to cope with large-scale problems. Fortunately, due to the wide availability of data, one attractive method of determining graphical model parameters is to learn them from data.

Learning graphical model parameters from data is typically reduced to finding the maximum likelihood estimates, which are the estimates that maximize the probability of observing the data. Such estimates are attractive as they are asymptotically consistent, which means that as the size of the data goes to infinity, the maximum likelihood estimates converge to the true unknown parameters (Fisher, 1922).

When the data is complete, i.e. has no missing values, learning can be viewed as solving a convex optimization problem. However, for some types of graphical models, every step of the optimization requires inference which is #P-hard (Roth, 1996). As a result, commonly used convex optimization algorithms, such as gradient descent, conjugate gradient (CG) (Hestenes & Stiefel, 1952), L-BFGS (Liu & Nocedal, 1989), and Iterative Proportional Fitting (IPF) (Jirousek & Preucil, 1995) can be very slow, due to the many function evaluations, and therefore inferences, they require.

When the data has missing values, the problem is generally non-convex. In that case, Expectation Maximization (EM) (Dempster, Laird, & Rubin, 1977; Lauritzen, 1995) has been widely used to get a locally optimal solution. While EM enjoys the monotonic enhancement property, it can be very slow for large models as it requires inference at every iteration.

Due to the complexity of learning maximum likelihood estimates, other simplified methods have been proposed in the literature such as ratio matching (Hyvarinen & Dayan, 2005), composite maximum likelihood (Varin, Reid, & Firth, 2011), and contrastive divergence (Hinton, 2000). For relational Markov networks or Markov networks that otherwise assume a feature-based representation (Domingos & Lowd, 2009), evaluating the likelihood is typically intractable, in which case one typically optimizes instead the pseudo-log-likelihood (Besag, 1975). The efficiency gains of these methods are however counterbalanced by less attractive statistical properties. For more on parameter estimation in graphical models, see (Koller & Friedman, 2009; Murphy, 2012).

We argue in this thesis that the degree of incompleteness is a main indicator of the difficulty of a learning problem. As such, we investigate a number of learning approaches, which are driven and motivated by this degree.

We have proposed the Edge Deletion Maximum Likelihood (EDML) algorithm, which is used for learning Maximum a Posteriori (MAP) parameters of a Bayesian network from incomplete data (Choi, Refaat, & Darwiche, 2011; Refaat, Choi, & Darwiche, 2012). EDML is procedurally very similar to Expectation Maximization (EM) (Dempster et al., 1977; Lauritzen, 1995), yet it has certain advantages, both theoretically and practically. Theoretically, EDML can in certain specialized cases provably converge in one iteration, whereas EM may require many iterations to solve the same learning problem. Some empirical evaluations further suggested that EDML and hybrid EDML/EM algorithms could find better parameter estimates than vanilla EM, in fewer iterations and less time. Furthermore, the learned parameters and running time of EDML improve as the data becomes less incomplete.

EDML was originally derived in terms of approximate inference on a meta-network used for Bayesian approaches to parameter estimation. This graphical representation of the estimation problem lent itself to the initial derivation of EDML, as well as to the identification of certain key theoretical properties, such as the ones we just described. The formal details, however, can be somewhat tedious, as EDML draws on a number of different concepts.

Later, we discovered a new perspective on EDML, giving new insights (Refaat, Choi, & Darwiche, 2013), by relating it to coordinate descent for continuous optimization, where sub-problems are parameterized by inference. This new perspective has a number of advantages. First, it makes immediate some results that were previously obtained for EDML, but through some effort. Second, it facilitates the design of new EDML algorithms for new classes of models, where graphical formulations of parameter estimation, such as meta-networks, are lacking. In particular, we derived a new parameter estimation algorithm for Markov networks, which is in many ways a more challenging task, compared to the case of Bayesian networks. Empirically, we find that EDML is sometimes capable of finding better parameter estimates, under complete data, in less time than more popular approaches like conjugate-gradient and L-BFGS methods, and in some cases, an order-of-magnitude faster.

In addition to this, we proposed a method to exactly decompose the problem of learning Bayesian network parameters from incomplete data into independent learning problems, which can lead to orders-of-magnitude speed-ups in learning time, without sacrificing quality (Refaat, Choi, & Darwiche, 2014). In particular, we exploit variables that are always observed in the dataset to decompose the learning problem. We show that as a result of this decomposition, the dataset can be compressed and the decomposed problems can converge independently. Moreover, we show that similar decomposition techniques can be used to compress the dataset when learning Markov random fields (MRFs) from incomplete data. We show that compressing the dataset may allow learning MRFs from large datasets hundreds of times faster.

We next give an overview of each chapter.

In Chapter 2, we give an introduction to graphical models, define our notation, and discuss the problem of learning graphical model parameters from data.

In Chapter 3, we propose EDML for learning MAP parameters in binary Bayesian networks under incomplete data. The method assumes Beta priors and can be used to learn maximum likelihood parameters when the priors are uninformative. EDML exhibits interesting behaviors, especially when compared to EM. We introduce EDML, explain its origin, and study some of its properties both analytically and empirically.

In Chapter 4, we provide a simple characterization of EDML that enables us to prove that its fixed points are exactly the stationary points of the likelihood (MAP) function. Furthermore, we modify EDML to guarantee improving the likelihood (MAP), at every step. In addition to this, we generalize EDML to support multivalued variables. We also propose a simple iterative method for solving its optimization problems. Finally, we provide experimental results suggesting that the modified EDML can be faster than EM.

In Chapter 5, we propose a greatly simplified perspective on EDML, which casts it as a general approach to continuous optimization. The new perspective simplifies some proofs and facilitates the design of EDML algorithms for new graphical models, leading to a new algorithm for learning parameters in Markov networks. We derive this algorithm in this chapter, and show empirically that it can sometimes learn estimates more efficiently from complete data, compared to commonly used optimization methods, such as conjugate gradient and L-BFGS.

In Chapter 6, we propose a technique for decomposing the parameter learning problem in Bayesian networks into independent learning problems. Our technique applies to incomplete datasets and exploits variables that are either hidden or observed in the given dataset. We show empirically that the proposed technique can lead to orders-of-magnitude savings in learning time. We explain analytically and empirically the reasons behind our reported savings, and compare the proposed technique to related ones that are sometimes used by inference algorithms.

In Chapter 7, we propose a technique for decomposing and compressing the dataset in the parameter learning problem in Markov random fields. Our technique applies to incomplete datasets and exploits variables that are always observed in the given dataset. We show that our technique allows exact computation of the gradient and the likelihood, and can lead to orders-of-magnitude savings in learning time. We summarize our contributions in Chapter 8.

CHAPTER 2

Technical Preliminaries

In this chapter, we give an introduction to Bayesian networks and Markov Random Fields as probabilistic graphical models. After that, we discuss the problem of learning graphical model parameters from data, and briefly discuss the notion of soft evidence.

2.1 Bayesian Networks

Consider the following scenario, which is due to (Pearl, 1988). Suppose that we have an alarm in our house that triggers when there is a burglary. However, the alarm could also trigger if an earthquake occurs. Our neighbor who has a hearing problem will give us a call in case she hears the alarm. One day we are at work and we receive a call from our neighbor saying that she heard the alarm. Given that we received this phone call, what is the probability that there is a burglary?

To answer this and other similar questions, we first need to specify the random variables of interest in this problem. There is Alarm (A), Burglary (B), Earthquake (E), and Call (C). Each variable can take one of two values: true or false. For example, E = true means that an earthquake occurred, whereas E = false means that it did not occur. Once the variables are known, we need to know the probability of every assignment for those variables. For example, what is the probability of E = false, B = false, A= false, C = false? If we are living in a safe place with no earthquakes, and both the alarm and our neighbor are reliable, then the probability of such an assignment will be close to 1. The probabilities of all assignments is called the joint probability distribution which can take the following form:

E	B	A	C	$Pr(.)$
false	false	false	false	0.8
false	false	false	true	0.05
false	false	true	false	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

where $Pr(.)$ denotes the probability of a value assignment to each of the variables.

Using probability theory, it is now easy to compute the probability of burglary given the call.

The joint probability distribution in this problem has 16 entries since we have 4 binary variables. In real world problems, the number of variables could be large leading to an impractical number of entries in the joint distribution which calls for a compact representation.

A Bayesian network could be used to provide a compact representation of the joint distribution. Figure 2.1 shows a Bayesian network for this problem. To build a Bayesian network, we start with the random variables in the problem definition as our nodes. After that, a directed edge is added from variable i to variable j if we perceive variable i to be a direct cause of variable j.

In this problem, both Earthquake and Burglary directly cause the Alarm and therefore there are edges from Earthquake to Alarm, and from Burglary to Alarm. In this case, Earthquake and Burglary are called the parents of Alarm, whereas Alarm is called their child. Moreover, the Alarm causes the Call and therefore we add an edge from Alarm to Call. Note that Earthquake and Burglary affects the Call variable. However, an edge was not added from Earthquake or Burglary to Call because they only affect the call indirectly (through Alarm). Formally, the graph structure is interpreted as a set of conditional independence constraints (Pearl, 1988). For example, given A, E and C are conditionally independent, and also B and C.

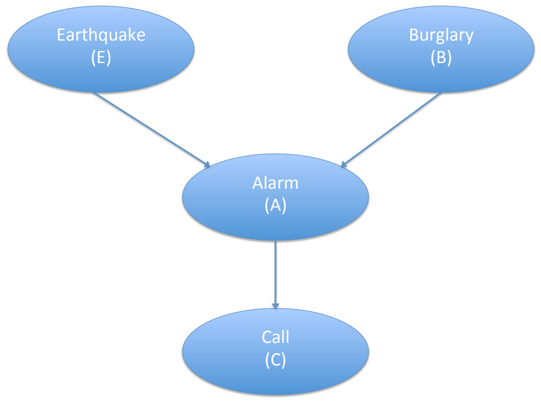


Figure 2.1: A Bayesian network for the Burglary problem

After building the Bayesian network structure, we associate with each variable a table called the conditional probability table (CPT). A CPT of a variable is a conditional probability distribution of this variable given its parents. For example, for Call, we have the following CPT:

A	C	$Pr(C\|A)$
false	false	0.7
false	true	0.3
true	false	0.2
true	true	0.8

Such CPTs could be created by a domain expert or by learning techniques, (see (Darwiche, 2009)). Once all of the CPTs are ready, there are many inference algorithms that could help us answer questions like the one raised at the beginning of this chapter. In other words, the CPTs (and conditional independencies) uniquely specify a joint probability distribution.

Note that the largest CPT will be the CPT of the variable with the largest number of direct causes (parents), assuming all variables have the same cardinality. However, the size of this CPT is typically much smaller than that of the joint probability distribution. The CPT size is exponential in the number of parents, whereas the size of the joint probability distribution is exponential in all of the variables involved in the problem definition. This is the main advantage of Bayesian networks over the explicit representation of joint probability distributions.

In our burglary example, we have manually built the network structure using the notion of cause and effect. Sometimes, deriving such relations is problematic or unknown in advance. For example, in a bioinformatics application, the role of a certain gene in causing a disease might not be known and therefore we might not be sure if we should add an edge from this gene to the disease.

One method to create a Bayesian network modeling some system or phenomenon is to learn the network from data examples. This learning problem was proved to be NP-Hard (Chickering & Heckerman, 1995).

2.2 Markov Random Fields

We have discussed Bayesian networks as a useful tool for representing a probability distribution. However, for some domains, being forced to choose a direction for edges can be rather awkward. For example, when we model an image, we may assume that the intensity values of neighboring pixels are correlated. We can then create a node for each pixel, and an edge between each two neighboring pixels. However, deciding on the direction of each edge may seem unnatural. An alternative is to use a Markov Random Field (MRF) which is a graphical model where edges do not have directions. MRFs can be more natural for problems such as image analysis (Murphy, 2012).

In brief, MRFs are probabilistic graphical models that have been useful to many fields, including computer vision, bioinformatics, natural language processing, and statistical physics. Like a Bayesian network, an MRF represents a joint probability distribution compactly using a structure populated with parameters. The structure is an undirected graph defining conditional independence relationships between variables in the graph, whereas the parameters consist of a factor for every maximal clique in the graph; see (Kindermann & Snell, 1980; Koller & Friedman, 2009; Murphy, 2012).

Figure 2.2 shows an MRF for the same Burglary problem we discussed. The graph is now undirected, and every maximal clique is associated with a factor. For example, the factor over variables A and C can take the form:

A	C	$f(A, C)$
false	false	7
false	true	3
true	false	2
true	true	8

The number associated with a particular value assignment in the factor denotes the affinity between these values: the higher the number, the more compatible these values are. Roughly speaking, $f(A, C)$ asserts that it is more likely that $A =$ true, $C =$ true than $A =$ true, $C =$ false.

Like in a Bayesian network, the parameters of the MRF defines the local interactions between directly related variables. In a Bayesian network, we combine the CPTs by multiplication. However, in an MRF, we have no guarantees that the result of this process is a normalized joint distribution. Thus, in an MRF, we combine the factors by multiplication, and then normalize the result to define a legal distribution; see (Koller & Friedman, 2009).

2.3 Learning Graphical Models

We use upper case letters (X) to denote random variables and lower case letters (x) to denote their specific values. Variable sets are denoted by bold-face upper case letters ($\mathbf{X}$) and their sets of instantiations by bold-face lower case letters ($\mathbf{x}$). Generally, we will use X to denote a variable in a Bayesian network and $\mathbf{U}$ to denote its parents. A network parameter will therefore have the general form $\theta_{x|\mathbf{u}}$, representing the probability $Pr(X = x|\mathbf{U} = \mathbf{u})$.

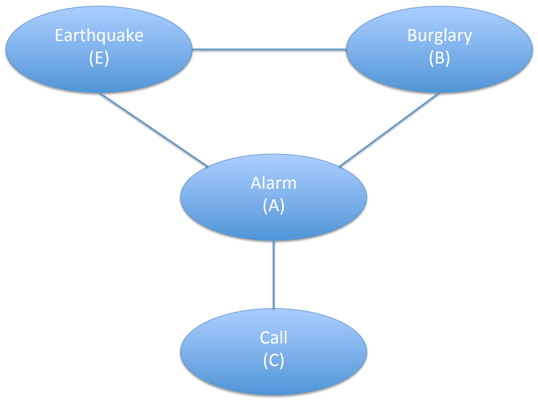


Figure 2.2: An MRF for the Burglary problem

Each variable X in a Bayesian network can be thought of as inducing a number of conditional random variables, denoted by $X|\mathbf{u}$, where the values of variable $X|\mathbf{u}$ are drawn based on the conditional distribution $Pr(X|\mathbf{u})$. Parameter estimation in Bayesian networks can be thought of as a process of estimating the distributions of these conditional random variables.

We will use θ to denote the set of all network parameters. Given a network structure G, our goal is to learn its parameters from an incomplete dataset, such as:

example	E	B	A	C
1	e_1	b_1	a_1	?
2	?	b_2	a_2	?
3	e_1	b_2	a_2	c_1

We use $\mathcal{D}$ to denote a dataset, and $\mathbf{d}_i$ to denote an example. The dataset above has three examples, with example $\mathbf{d}_2$ being $B = b_2$, $A = a_2$, and E and C being unknown.

2.3.1 Learning Parameters in Bayesian Networks

A commonly used measure for the quality of parameter estimates θ is their likelihood, defined as:

$$L(\theta|\mathcal{D}) = \prod_{i=1}^{N} Pr_\theta(\mathbf{d}_i),$$

where Pr_θ is the distribution induced by network structure G and parameters θ, and N is the number of data examples. We assume that the data examples are drawn independently from the true distribution. The likelihood is computed by computing the probability of each data example, and multiplying these probabilities to get the probability of observing the dataset. In the case of complete data (each example fixes the value of each variable), the maximum likelihood (ML) parameters are unique and easily obtainable by counting. For example, $\theta_{x|\mathbf{u}}$ is computed as $\frac{\#(x,\mathbf{u})}{\#(\mathbf{u})}$, where $\#(x, \mathbf{u})$ is the number of times that x and $\mathbf{u}$ appear in the dataset, and $\#(\mathbf{u})$ is the number of times that $\mathbf{u}$ appears in the dataset; see (Darwiche, 2009).

Learning ML parameters is harder when the data is incomplete and the EM algorithm (Dempster et al., 1977; Lauritzen, 1995) is typically employed. EM starts with some initial parameters θ^0, called a *seed,* and successively improves on them via iteration. EM uses the update equation:

$$\theta_{x|\mathbf{u}}^{k+1} = \frac{\sum_{i=1}^{N} Pr_{\theta^k}(x\mathbf{u}|\mathbf{d}_i)}{\sum_{i=1}^{N} Pr_{\theta^k}(\mathbf{u}|\mathbf{d}_i)},$$

which requires inference on a Bayesian network parameterized by θ^k, in order to compute $Pr_{\theta^k}(x\mathbf{u}|\mathbf{d}_i)$ and $Pr_{\theta^k}(\mathbf{u}|\mathbf{d}_i)$. This iterative equation uses the current learned parameters θ^k to compute the most likely new parameters θ^{k+1}. Given the new learned parameters θ^{k+1}, it computes new statistics $Pr_{\theta^{k+1}}(x\mathbf{u}|\mathbf{d}_i)$ and $Pr_{\theta^{k+1}}(\mathbf{u}|\mathbf{d}_i)$ to be used to compute new parameters θ^{k+2}, and the process repeats.

It is known that one run of the jointree algorithm on each example is sufficient to implement an iteration of EM, which is guaranteed to never decrease the likelihood of its estimates across iterations. EM can also converge to every local maximum given that it starts with an appropriate seed. It is common to run EM with multiple seeds, keeping the best local maximum it finds. See (Darwiche, 2009; Koller & Friedman, 2009) for recent treatments on parameter learning in Bayesian networks via EM and related methods.

EM can also be used to find Maximum a Posteriori (MAP) parameters given Dirichlet priors on network parameters. The Dirichlet prior for the parameters of a random variable $X|\mathbf{u}$ is specified by a set of exponents, $\psi_{x|\mathbf{u}}$, leading to a density $\propto \prod_x [\theta_{x|\mathbf{u}}]^{\psi_{x|\mathbf{u}}-1}$. It is common to assume that exponents are greater than one, which guarantees a unimodal density. For MAP parameters, EM uses the update (see, e.g., Darwiche (2009)):

$$\theta^{k+1}_{x|\mathbf{u}} = \frac{\psi_{x|\mathbf{u}} - 1 + \sum_{i=1}^{N} Pr_{\theta^k}(x\mathbf{u}|\mathbf{d}_i)}{\psi_{X|\mathbf{u}} - |X| + \sum_{i=1}^{N} Pr_{\theta^k}(\mathbf{u}|\mathbf{d}_i)}, \tag{2.1}$$

where $\psi_{X|\mathbf{u}} = \sum_x \psi_{x|\mathbf{u}}$. This iterative equation uses the current learned parameters θ^k to compute the most likely new parameters θ^{k+1}. Given the new learned parameters θ^{k+1}, it computes new statistics $Pr_{\theta^{k+1}}(x\mathbf{u}|\mathbf{d}_i)$ and $Pr_{\theta^{k+1}}(\mathbf{u}|\mathbf{d}_i)$ to be used to compute new parameters θ^{k+2}, and the process repeats. When $\psi_{x|\mathbf{u}} = 1$, the equation reduces to the one for computing ML parameters. Moreover, using $\psi_{x|\mathbf{u}} = 2$ leads to ML parameters with Laplace smoothing. This is a common technique to deal with the problem of insufficient counts (i.e., instantiations that never appear in the dataset, leading to zero probabilities and division by zero). We will use Laplace smoothing in our experiments.

2.3.2 Learning Parameters in MRFs

In Bayesian networks, computing the probability of a complete variable assignment is done by multiplying CPT values. However, in MRFs, factors do not have direct probabilistic semantics, and therefore a normalization constant, called the partition function, Z_θ, is used to give them probabilistic meaning. The log-likelihood in the case of a Markov network is:

$$\ell\ell(\theta|\mathcal{D}) = \sum_{i=1}^{N} \log \frac{Z_\theta(\mathbf{d}_i)}{Z_\theta} = -N \log Z_\theta + \sum_{i=1}^{N} \log Z_\theta(\mathbf{d}_i) \tag{2.2}$$

where Z_θ is the partition function, and where $Z_\theta(\mathbf{d}_i)$ is similar to $Pr_\theta(\mathbf{d}_i)$ except that it is not normalized. The second term $\sum_{i=1}^{N} \log Z_\theta(\mathbf{d}_i)$ takes the same form as the log-likelihood used for BNs, whereas the first term $-N \log Z_\theta$ is added for normalization, in order to ensure computing valid probability values. Namely, we have $\frac{Z_\theta(\mathbf{d}_i)}{Z_\theta} = Pr_\theta(\mathbf{d}_i)$. The goal is to maximize the likelihood as in the case of Bayesian networks.

2.3.3 Soft Evidence

EDML makes heavy use of soft evidence (i.e., evidence that changes the distribution of a variable without necessarily fixing its value). In this section, we give an introduction to the semantics of soft evidence.

We follow the treatment of (Pearl, 1988) for soft evidence, which models soft evidence as hard evidence on a virtual event η. In particular, soft evidence on some variable X with k values is quantified by a vector $\lambda_{x_1}, \ldots, \lambda_{x_k}$ with $\lambda_{x_i} \in [0, \infty)$. The semantics is that $\lambda_{x_1} : \cdots : \lambda_{x_k} \propto Pr(\eta|x_1) : \cdots : Pr(\eta|x_k)$. The soft evidence on variable X is then emulated by asserting the hard evidence η. That is, the new distribution on variable X after having asserted the soft evidence is modeled by $Pr(X|\eta)$. Note that $Pr(X|\eta)$ depends only on the ratios $\lambda_{x_1} : \cdots : \lambda_{x_k}$, not on their absolute values. Hard evidence of the form $X = x_j$ can be modeled using $\lambda_{x_i} = 0$ for all $i \neq j$, and $\lambda_{x_j} = 1$. Moreover, neutral evidence can be modeled using $\lambda_{x_i} = 1$ for all i. The reader is referred to (Pearl, 1988) for more details.

As an example of soft evidence, consider the burglary problem, and suppose that rumors were heard that an earthquake occurred. Since the rumors are not conclusive, we would like to model this evidence as soft rather than hard. One way to assert the soft evidence to model the rumors is to add an auxiliary node Rumors as shown in Figure 2.3 and fix Rumors to true. The CPT of Rumors quantifies the strength of the soft evidence.

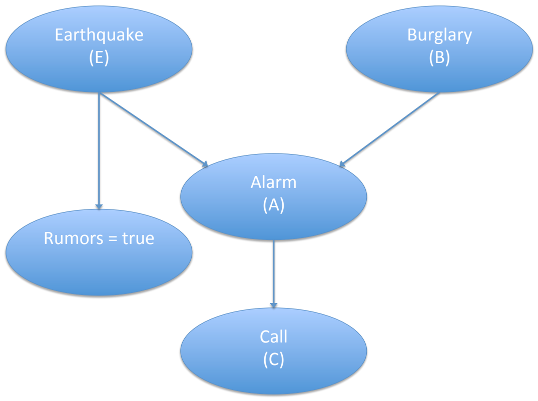


Figure 2.3: A Bayesian network for the Burglary problem with soft evidence on Earthquake

CHAPTER 3

EDML: A Method for Learning Parameters in Bayesian Networks

We propose a method called EDML for learning MAP parameters in binary Bayesian networks under incomplete data. The method assumes Beta priors and can be used to learn maximum likelihood parameters when the priors are uninformative. EDML exhibits interesting behaviors, especially when compared to EM. We introduce EDML, explain its origin, and study some of its properties both analytically and empirically. This chapter is based on (Choi et al., 2011).

3.1 Introduction

We consider in this chapter the problem of learning Bayesian network parameters given incomplete data, while assuming that all network variables are binary. We propose a specific method, EDML,[1] which has a similar structure and complexity to the EM algorithm (Dempster et al., 1977; Lauritzen, 1995). EDML assumes Beta priors on network parameters, allowing one to compute MAP parameters. When using uninformative priors, EDML reduces to computing maximum likelihood (ML) parameters.

EDML originated from applying an approximate inference algorithm (Choi & Darwiche, 2006) to a meta network in which parameters are explicated as variables, and on which data is asserted as evidence. The update equations of EDML resemble the

[1]EDML stands for **E**dge-**D**eletion **MAP**-**L**earning or **E**dge-**D**eletion **M**aximum-**L**ikelihood as it is based on an edge-deletion approximate inference algorithm that can compute MAP or maximum likelihood parameters.

ones for EM, yet EDML appears to have different convergence properties which stem from its being an inference method as opposed to a local search method. For example, we will identify a class of incomplete datasets on which EDML is guaranteed to converge immediately to an optimal solution, by simply reasoning about the behavior of its underlying inference method.

Even though EDML originates in a rather involved approximate inference scheme, its update equations can be intuitively justified independently. We therefore present EDML initially in Section 3.3 before delving into the details of how it was originally derived in Section 3.5.

Intuitively, EDML can be thought of as relying on two key concepts. The first concept is that of estimating the parameters of a single random variable given *soft observations,* i.e., observations that provide soft evidence on the values of a random variable. The second key concept behind EDML is that of interpreting the examples of an incomplete data set as providing soft observations on the random variables of a Bayesian network. As to the first concept, we also show that MAP and ML parameter estimates are unique in this case, therefore, generalizing the fundamental result which says that these estimates are unique for hard observations. This result is interesting and fundamental enough that we treat it separately in Section 3.4 before we move on and discuss the origin of EDML in Section 3.5.

We discuss some theoretical properties of EDML in Section 3.6, where we identify situations in which it is guaranteed to converge immediately to optimal estimates. We present some empirical results in Section 3.7 that corroborate some of the convergence behaviors predicted. In Section 3.8, we close with some concluding remarks on related work. We note that while we focus on binary variables in this chapter, our approach generalizes to multivalued variables as well. We will comment later on this and the reason we restricted our focus here.

3.2 Technical Preliminaries

As discussed earlier, EM can be used to find MAP parameters, assuming one has some priors on network parameters. The Beta distribution is commonly used as a prior on the probability of a binary random variable. In particular, the Beta for random variable $X_{\mathbf{u}}$ is specified by two exponents, $\alpha_{X_{\mathbf{u}}}$ and $\beta_{X_{\mathbf{u}}}$, leading to a density $\propto [\theta_{x|\mathbf{u}}]^{\alpha_{X_{\mathbf{u}}}-1}[1-\theta_{x|\mathbf{u}}]^{\beta_{X_{\mathbf{u}}}-1}$. It is common to assume that exponents are > 1 (the density is then unimodal). For MAP parameters, EM uses the update equation (see, e.g., (Darwiche, 2009)):

$$\theta^{k+1}_{x|\mathbf{u}} = \frac{\alpha_{X_{\mathbf{u}}} - 1 + \sum_{i=1}^{N} Pr_{\theta^k}(x\mathbf{u}|\mathbf{d}_i)}{\alpha_{X_{\mathbf{u}}} + \beta_{X_{\mathbf{u}}} - 2 + \sum_{i=1}^{N} Pr_{\theta^k}(\mathbf{u}|\mathbf{d}_i)}.$$

When $\alpha_{X_{\mathbf{u}}} = \beta_{X_{\mathbf{u}}} = 1$ (uninformative prior), the equation reduces to the one for computing ML parameters. When computing ML parameters, using $\alpha_{X_{\mathbf{u}}} = \beta_{X_{\mathbf{u}}} = 2$ leads to what is usually known as Laplace smoothing. This is a common technique to deal with the problem of insufficient counts (i.e., instantiations that never appear in the dataset, leading to zero probabilities and division by zero). We will indeed use Laplace smoothing in our experiments.

Our method for learning MAP and ML parameters makes heavy use of two notions: (1) the *odds* of an event, which is the probability of the event over the probability of its negation, and (2) the *Bayes factor* (Good, 1950), which is the relative change in the odds of one event, say, $X = x$, due to observing some other event, say, η. In this case, we have the odds $O(x)$ and $O(x|\eta)$, where the Bayes factor is $\kappa = O(x|\eta)/O(x)$, which is viewed as quantifying the strength of *soft evidence* η on $X = x$. It is known that $\kappa = Pr(\eta|x)/Pr(\eta|\bar{x})$ and $\kappa \in [0, \infty]$. When $\kappa = 0$, the soft evidence reduces to hard evidence asserting $X = \bar{x}$. When $\kappa = \infty$, the soft evidence reduces to hard evidence asserting $X = x$. When $\kappa = 1$, the soft evidence is neutral and bears no information on $X = x$. A detailed discussion on the use of Bayes factors for soft evidence is given in (Chan & Darwiche, 2005).

Algorithm 1 EM

```
input:
  $G$:                          A Bayesian network structure
  $\mathcal{D}$:                An incomplete dataset $\mathbf{d}_1, \ldots, \mathbf{d}_N$
  $\theta$:                     An initial parameterization of $G$
  $\alpha_{X_\mathbf{u}}, \beta_{X_\mathbf{u}}$:  Beta prior for each variable $X_\mathbf{u}$
1: while not converged do
2:     $Pr \leftarrow$ distribution induced by $\theta$ and $G$
3:     Compute probabilities:
           $Pr(x\mathbf{u}|\mathbf{d}_i)$   and   $Pr(\mathbf{u}|\mathbf{d}_i)$
       for each family instantiation $x\mathbf{u}$ and example $\mathbf{d}_i$
4:     Update parameters:
           $\theta_{x|\mathbf{u}} \leftarrow \frac{\alpha_{X_\mathbf{u}} - 1 + \sum_{i=1}^{N} Pr(x\mathbf{u}|\mathbf{d}_i)}{\alpha_{X_\mathbf{u}} + \beta_{X_\mathbf{u}} - 2 + \sum_{i=1}^{N} Pr(\mathbf{u}|\mathbf{d}_i)}$
5: return parameterization $\theta$
```

Algorithm 2 EDML

```
input:
  $G$:                          A Bayesian network structure
  $\mathcal{D}$:                An incomplete dataset $\mathbf{d}_1, \ldots, \mathbf{d}_N$
  $\theta$:                     An initial parameterization of $G$
  $\alpha_{X_\mathbf{u}}, \beta_{X_\mathbf{u}}$:  Beta prior for each variable $X_\mathbf{u}$
1: while not converged do
2:     $Pr \leftarrow$ distribution induced by $\theta$ and $G$
3:     Compute Bayes factors:
           $\kappa^i_{x|\mathbf{u}} \leftarrow \frac{Pr(x\mathbf{u}|\mathbf{d}_i)/Pr(x|\mathbf{u}) - Pr(\mathbf{u}|\mathbf{d}_i) + 1}{Pr(\bar{x}\mathbf{u}|\mathbf{d}_i)/Pr(\bar{x}|\mathbf{u}) - Pr(\mathbf{u}|\mathbf{d}_i) + 1}$   (3.1)
       for each family instantiation $x\mathbf{u}$ and example $\mathbf{d}_i$
4:     Update parameters:
           $\theta_{x|\mathbf{u}} \leftarrow \underset{p}{\text{argmax}}\ [p]^{\alpha_{X_\mathbf{u}}-1}[1-p]^{\beta_{X_\mathbf{u}}-1} \times \prod_{i=1}^{N} [\kappa^i_{x|\mathbf{u}} \cdot p - p + 1]$   (3.2)
5: return parameterization $\theta$
```

3.3 An Overview of EDML

Consider Algorithm 1, which provides pseudocode for EM. EM typically starts with some initial parameter estimates, called a seed, and then iterates to monotonically improve on these estimates. Each iteration consists of two steps. The first step, Line 3, computes marginals over the families of a Bayesian network that is parameterized by the current estimates. The second step, Line 4, uses the computed probabilities to update the network parameters. The process continues until some convergence criterion is met. The main point here is that the *computation* on Line 3 can be implemented by a single run of the jointree algorithm, while the *update* on Line 4 is immediate.

Consider now Algorithm 2, which provides pseudocode for EDML, to be contrasted with the one for EM. The two algorithms clearly have the same overall structure. That is, EDML also starts with some initial parameters estimates, called a seed, and then iterates to update these estimates. Each iteration consists of two steps. The first step, Line 3, computes Bayes factors using a Bayesian network that is parameterized by the current estimates. The second step, Line 4, uses the computed Bayes factors to update network parameters. The process continues until some convergence criterion is met. Much like EM, the *computation* on Line 3 can be implemented by a single run of the jointree algorithm. Unlike EM, however, the *update* on Line 4 is not immediate as it involves solving an optimization problem, albeit a simple one. Aside from this optimization task, EM and EDML have the same computational complexity.

We next explain the two concepts underlying EDML and how they lead to the equations of Algorithm 2.

3.3.1 Estimation from Soft Observations

Consider a random variable X with values x and $\bar{x}$, and suppose that we have $N > 0$ independent observations of X, with N_x as the number of positive observations. It is well known that the ML parameter estimates for random variable X are unique in this case and characterized by $\theta_x = N_x/N$. If one further assumes a Beta prior with exponents α and β that are ≥ 1, it is also known that the MAP parameter estimates are unique and characterized by $\theta_x = \frac{N_x+\alpha-1}{N+\alpha+\beta-2}$.

Consider now a more general problem in which the observations are soft in that they only provide soft evidence on the values of random variable X. That is, each soft observation η_i is associated with a Bayes factor $\kappa_x^i = O(x|\eta_i)/O(x)$ which quantifies the evidence that η_i provides on having observed the value x of variable X. We will show later that the ML estimates remain unique in this more general case, if at least one of the soft observations is not trivial (i.e., with Bayes factor $\kappa_x^i \neq 1$). Moreover, we will show that the MAP estimates are also unique assuming a Beta prior with exponents ≥ 1. In particular, we will show that the unique MAP estimates are characterized by Equation 3.2 of Algorithm 2. Further, we will show that the unique ML estimates are characterized by the same equation while using a Beta prior with exponents $= 1$. This is the first key concept that underlies our proposed algorithm for estimating ML and MAP parameters in a binary Bayesian network.

3.3.2 Examples as Soft Observations

The second key concept underlying EDML is to interpret each example $\mathbf{d}_i$ in a dataset as providing a soft observation on each random variable $X_{\mathbf{u}}$. As mentioned earlier, soft observations are specified by Bayes factors and, hence, one needs to specify the Bayes factor $\kappa^i_{x|\mathbf{u}}$ that example $\mathbf{d}_i$ induces on random variable $X_{\mathbf{u}}$. EDML uses Equation 3.1 for this purpose, which will be derived in Section 3.5. We next consider a few special cases of this equation to highlight its behavior.

Consider first the case in which example $\mathbf{d}_i$ implies parent instantiation $\mathbf{u}$ (i.e., the parents $\mathbf{U}$ of variable X are instantiated to $\mathbf{u}$ in example $\mathbf{d}_i$). In this case, Equation 3.1 reduces to $\kappa^i_{x|\mathbf{u}} = \frac{O(x|\mathbf{u},\mathbf{d}_i)}{O(x|\mathbf{u})}$, which is the relative change in the odds of x given $\mathbf{u}$ due to conditioning on example $\mathbf{d}_i$. Note that for root variables X, which have no parents $\mathbf{U}$, Equation 3.1 further reduces to $\kappa^i_x = \frac{O(x|\mathbf{d}_i)}{O(x)}$.

The second case we consider is when example $\mathbf{d}_i$ is inconsistent with parent instantiation $\mathbf{u}$. In this case, Equation 3.1 reduces to $\kappa^i_{x|\mathbf{u}} = 1$, which amounts to neutral evidence. Hence, example $\mathbf{d}_i$ is irrelevant to estimating the distribution of variable $X_{\mathbf{u}}$ in this case, and will be ignored by EDML.

The last special case of Equation 3.1 we shall consider is when the example $\mathbf{d}_i$ is complete; that is, it fixes the value of each variable. In this case, one can verify that $\kappa^i_{x|\mathbf{u}} \in \{0, 1, \infty\}$ and, hence, the example can be viewed as providing either neutral or hard evidence on each random variable $X_{\mathbf{u}}$. Thus, an example will provide soft obser-

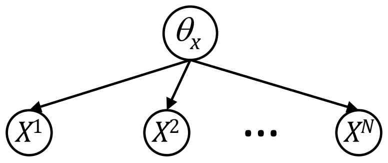


Figure 3.1: Estimation given independent observations.

vations on variables only when it is incomplete (i.e., missing some values). Otherwise, it is either irrelevant to, or provides a hard observation on each variable $X_{\mathbf{u}}$.

In the next section, we prove Equation 3.2 of Algorithm 2. In Section 3.5, we discuss the origin of EDML, where we go on and derive Equation 3.1 of Algorithm 2.

3.4 Estimation from Soft Observations

Consider a binary variable X. Figure 3.1 depicts a network where θ_x is a parameter representing $Pr(X\!=\!x)$ and $X^1, \ldots, X^N$ are independent observations of X. Suppose further that we have a Beta prior on parameter θ_x with exponents $\alpha \geq 1$ and $\beta \geq 1$. A standard estimation problem is to assume that we know the values of these observations and then estimate the parameter θ_x. We now consider a variant on this problem, in which we only have soft evidence η_i about each observation, whose strength is quantified by a Bayes factor $\kappa_x^i = O(x|\eta_i)/O(x)$. Here, κ_x^i represents the change in odds that the i-th observation is positive due to evidence η_i. We will refer to η_i as a *soft observation* on variable X, and our goal in this section is to compute (and optimize) the posterior density on parameter θ_x given these soft observations $\eta_1, \ldots, \eta_N$.

We first consider the likelihood:

$$
\begin{aligned}
&Pr(\eta_1, \ldots, \eta_N|\theta_x) = \prod_{i=1}^N Pr(\eta_i|\theta_x) \\
&= \prod_{i=1}^N [Pr(\eta_i|x, \theta_x)Pr(x|\theta_x) + Pr(\eta_i|\bar{x}, \theta_x)Pr(\bar{x}|\theta_x)] \\
&= \prod_{i=1}^N [Pr(\eta_i|x)\theta_x + Pr(\eta_i|\bar{x})(1-\theta_x)] \\
&\propto \prod_{i=1}^N [\kappa_x^i \cdot \theta_x - \theta_x + 1].
\end{aligned}
$$

The last step follows because $\kappa_x^i = O(x|\eta_i)/O(x) = Pr(\eta_i|x)/Pr(\eta_i|\bar{x})$. The posterior density is then:

$$
\begin{aligned}
&\rho(\theta_x|\eta_1, \ldots, \eta_N) \propto \rho(\theta_x)Pr(\eta_1, \ldots, \eta_N|\theta_x) \\
&\propto [\theta_x]^{\alpha-1}[1-\theta_x]^{\beta-1} \prod_{i=1}^N [\kappa_x^i \cdot \theta_x - \theta_x + 1].
\end{aligned}
$$

This is exactly Equation 3.2 of Algorithm 2 assuming we replace the random variable X with the conditional random variable $X_{\mathbf{u}}$.[2]

The second derivative of the log posterior is

$$
-\frac{\alpha - 1}{[\theta_x]^2} - \frac{\beta - 1}{[1-\theta_x]^2} - \sum_i \left[\frac{(\kappa_x^i - 1)}{(\kappa_x^i - 1)\theta_x + 1}\right]^2
$$

which is strictly negative when $\kappa_x^i \neq 1$ for at least one i. This remains true when $\alpha = \beta = 1$. Hence, both the likelihood function and the posterior density are strictly log-concave and therefore have unique modes. This means that both ML and MAP parameter estimates are unique in the case of soft, independent observations, which generalizes the uniqueness result for hard, independent observations on a variable X.

3.5 The Origin of EDML

This section reveals the technical origin of EDML, showing how Equation 3.1 of Algorithm 2 is derived, and providing the basis for the overall structure of EDML as spelled out in Algorithm 2.

[2] The case of $\kappa_x^i = \infty$ needs to be handled carefully in Equation 3.2. First note that $\kappa_x^i = \infty$ iff $Pr(\eta_i|\bar{x}) = 0$ in the derivation of this equation. In this case, the term $Pr(\eta_i|x)\theta_x + Pr(\eta_i|\bar{x})(1-\theta_x)$ equals $c \cdot \theta_x$ for some constant $c \in (0, 1]$. Since the value of Equation 3.2 does not depend on constant c, we will assume $c = 1$. Hence, when $\kappa_x^i = \infty$, the term $[\kappa_x^i \cdot \theta_x - \theta_x + 1]$ evaluates to θ_x by convention.

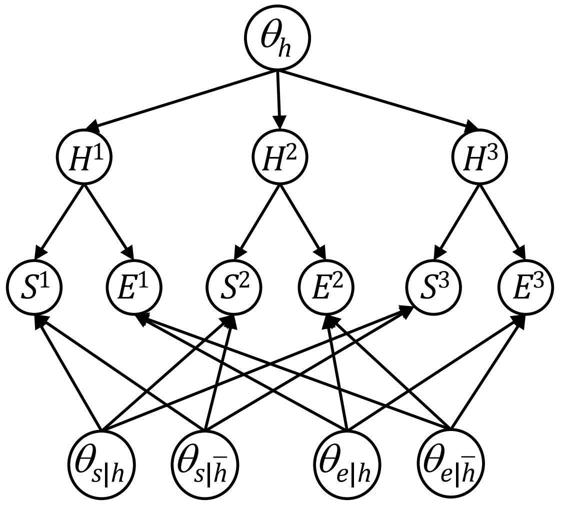


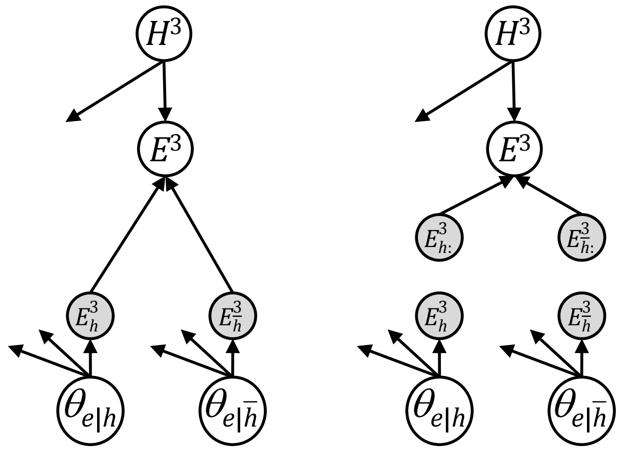
Figure 3.2: A meta network induced from a base network $S \longleftarrow H \longrightarrow E$. The CPTs here are based on standard semantics; see, e.g., (Darwiche, 2009, Ch. 18).

EDML originated from an approximation algorithm for computing MAP parameters in a *meta network.* Figure 3.2 depicts an example meta network in which parameters are represented explicitly as nodes (Darwiche, 2009). In particular, for each conditional random variable $X_{\mathbf{u}}$ in the original Bayesian network, called the *base network*, we have a node $\theta_{x|\mathbf{u}}$ in the meta network which represents a parameter that characterizes the distribution of this random variable. Moreover, the meta network includes enough instances of the base network to allow the assertion of each example $\mathbf{d}_i$ as evidence on one of these instances. Assuming that θ is an instantiation of all parameter variables, and $\mathcal{D}$ is a dataset, MAP estimates are then:

$$\theta^{\star} = \underset{\theta}{\operatorname{argmax}}\, \rho(\theta|\mathcal{D}),$$

where ρ is the density induced by the meta network.

Computing MAP estimates exactly is usually prohibitive due to the structure of the meta network. We therefore use the technique of edge deletion (Choi & Darwiche, 2006), which formulates approximate inference as exact inference on a simplified network that is obtained by deleting edges from the original network. The technique com-



(a) Adding generators (b) Deleting copy edges

Figure 3.3: Introducing generators into a meta network and then deleting copy edges from the resulting meta network, which leads to introducing clones.

pensates for these deletions by introducing auxiliary parameters whose values must be chosen carefully (and usually iteratively) in order to improve the quality of approximations obtained from the simplified network. EDML is the result of making a few specific choices for deleting edges and for choosing values for the auxiliary parameters introduced, which we explain next.

3.5.1 Introducing Generators

Let X^i denote the instance of variable X in the base network corresponding to example $\mathbf{d}_i$. The first choice of EDML is that for each edge $\theta_{x|\mathbf{u}} \longrightarrow X^i$ in the meta network, we introduce a *generator variable* $X^i_{\mathbf{u}}$, leading to the pair of edges $\theta_{x|\mathbf{u}} \longrightarrow X^i_{\mathbf{u}} \longrightarrow X^i$. Figure 3.3(a) depicts a fragment of the meta network in Figure 3.2, in which we introduced two generator variables for edges $\theta_{e|h} \longrightarrow E^3$ and $\theta_{e|\bar{h}} \longrightarrow E^3$, leading to $\theta_{e|h} \longrightarrow E^3_h \longrightarrow E^3$ and $\theta_{e|\bar{h}} \longrightarrow E^3_{\bar{h}} \longrightarrow E^3$.

Variable $X^i_{\mathbf{u}}$ is meant to generate values of variable X^i according to the distribution specified by parameter $\theta_{x|\mathbf{u}}$. Hence, the conditional distribution of a generator $X^i_{\mathbf{u}}$ is such that $Pr(x^i_{\mathbf{u}}|\theta_{x|\mathbf{u}}) = \theta_{x|\mathbf{u}}$. Moreover, the CPT of variable X^i is set to ensure that variable X^i copies the value of generator $X^i_{\mathbf{u}}$ if and only if the parents of X^i take on the value $\mathbf{u}$. That is, the CPT of variable X^i acts as a selector that chooses a particular generator $X^i_{\mathbf{u}}$ to copy from, depending on the values of its parents $\mathbf{U}$. For example, in Figure 3.3(a), when parent H^3 takes on its positive value h, variable E^3 copies the value of generator E^3_h. When parent H^3 takes on its negative value $\bar{h}$, variable E^3 copies the value of generator $E^3_{\bar{h}}$.

Adding generator variables does not change the meta network as it continues to have the same density over the original variables. Yet, generators are essential to the derivation of EDML as they will be used for interpreting data examples as soft observations.

3.5.2 Deleting Copy Edges

The second choice made by EDML is that we only delete edges of the form $X^i_{\mathbf{u}} \longrightarrow X^i$ from the augmented meta network, which we shall call *copy edges.* Figure 3.3(b) depicts an example in which we have deleted the two copy edges from Figure 3.3(a).

Note here the addition of another auxiliary variable $X^i_{\mathbf{u}:}$, called a *clone,* for each generator $X^i_{\mathbf{u}}$. The addition of clones is mandated by the edge deletion framework. Moreover, if the CPT of clone $X^i_{\mathbf{u}:}$ is chosen carefully, it can compensate for the parent-to-child information lost when deleting edge $X^i_{\mathbf{u}} \longrightarrow X^i$. We will later see how EDML sets these CPTs. The other aspect of compensating for a deleted edge is to specify soft evidence on each generator $X^i_{\mathbf{u}}$. This is also mandated by the edge deletion framework, and is meant to compensate for the child-to-parent information lost when deleting edge $X^i_{\mathbf{u}} \longrightarrow X^i$. We will later see how EDML sets this soft evidence as well, which effectively completes the specification of the algorithm. We prelude this specification, however, by making some further observations about the structure of the meta network after edge deletion.

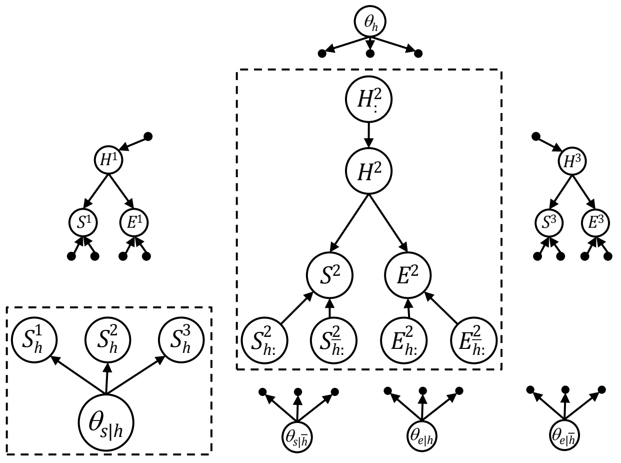


Figure 3.4: An edge-deleted network obtained from the meta network in Figure 3.2 found by: (1) adding generator variables, (2) deleting copy edges, and (3) adding cloned generators. The figure highlights the island for example $\mathbf{d}_2$, and the island for parameter $\theta_{s|h}$.

3.5.3 Parameter & Example Islands

Consider the network in Figure 3.4, which is obtained from the meta network in Figure 3.2 according to the edge-deletion process indicated earlier.

The edge-deleted network contains a set of disconnected structures, called *islands.* Each island belongs to one of two classes: a *parameter island* for each network parameter $\theta_{x|\mathbf{u}}$ and an *example island* for each example $\mathbf{d}_i$ in the dataset. Figure 3.4 provides the full details for one example island and one parameter island. Note that each parameter island corresponds to a Naive Bayes structure, with parameter $\theta_{x|\mathbf{u}}$ as the root and generators $X^i_{\mathbf{u}}$ as children. When soft evidence is asserted on these generators, we get the estimation problem we treated in Section 3.4.

EDML can now be fully described by specifying (1) the soft evidence on each generator $X^i_{\mathbf{u}}$ in a parameter island, and (2) the CPT of each clone $X^i_{\mathbf{u}:}$ in an example island. These specifications are given next.

3.5.4 Child-To-Parent Compensation

The edge deletion approach suggests the following soft evidence on generators $X^i_{\mathbf{u}}$, specified as Bayes factors:

$$\kappa^i_{x|\mathbf{u}} = \frac{O(x^i_{\mathbf{u}:}|\mathbf{d}_i)}{O(x^i_{\mathbf{u}:})} = \frac{Pr^i(\mathbf{d}_i|x^i_{\mathbf{u}:})}{Pr^i(\mathbf{d}_i|\bar{x}^i_{\mathbf{u}:})}, \tag{3.3}$$

where Pr^i is the distribution induced by the island of example $\mathbf{d}_i$. We will now show that this equation simplifies to Equation 3.1 of Algorithm 2.

Suppose that we marginalize all clones $X^i_{\mathbf{u}:}$ from the island of example $\mathbf{d}_i$, leading to a network that induces a distribution Pr. The new network has the following properties. First, it has the same structure as the base network. Second, $Pr(x|\mathbf{u}) = Pr^i(x^i_{\mathbf{u}:})$, which means that the CPTs of clones in example islands correspond to parameters in the base network. Finally, if we use $\overline{\mathbf{u}}$ to denote the disjunction of all parent instantiations excluding $\mathbf{u}$, we get:

$$\begin{aligned}
\kappa^i_{x|\mathbf{u}} &= \frac{Pr^i(\mathbf{d}_i|x^i_{\mathbf{u}:})}{Pr^i(\mathbf{d}_i|\bar{x}^i_{\mathbf{u}:})} \\
&= \frac{Pr(\mathbf{d}_i|x\mathbf{u})Pr(\mathbf{u}) + Pr(\mathbf{d}_i|\overline{\mathbf{u}})Pr(\overline{\mathbf{u}})}{Pr(\mathbf{d}_i|\bar{x}\mathbf{u})Pr(\mathbf{u}) + Pr(\mathbf{d}_i|\overline{\mathbf{u}})Pr(\overline{\mathbf{u}})} \\
&= \frac{Pr(x\mathbf{u}|\mathbf{d}_i)/Pr(x|\mathbf{u}) - Pr(\mathbf{u}|\mathbf{d}_i) + 1}{Pr(\bar{x}\mathbf{u}|\mathbf{d}_i)/Pr(\bar{x}|\mathbf{u}) - Pr(\mathbf{u}|\mathbf{d}_i) + 1}.
\end{aligned}$$

This is exactly Equation 3.1 of Algorithm 2. Hence, we can evaluate Equation 3.3 by evaluating Equation 3.1 on the base network, as long as we seed the base network with parameters that correspond to the CPTs of clones in an example island.

3.5.5 Parent-To-Child Compensation

We now complete the derivation of EDML by showing how it specifies the CPTs of clones in example islands, which are needed for computing soft evidence as in the previous section.

In a nutshell, EDML assumes an initial value of these CPTs, typically chosen randomly. Given these CPTs, example islands will be fully specified and EDML will

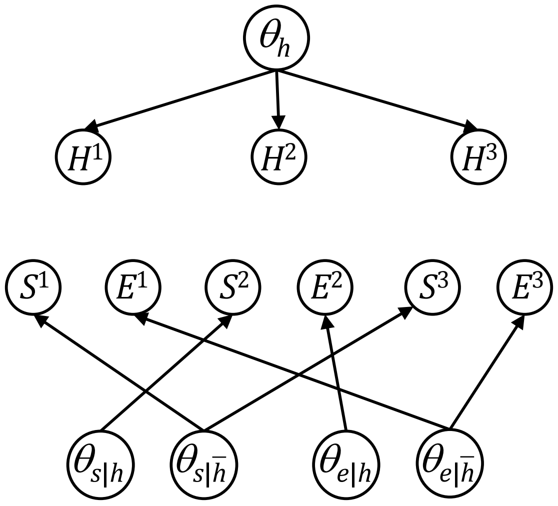


Figure 3.5: A pruning of the meta network in Figure 3.2 given $H^1 = \bar{h}$, $H^2 = h$ and $H^3 = \bar{h}$.

compute soft evidence as given by Equation 3.3. The computed soft evidence is then injected on the generators of parameter islands, leading to a full specification of these islands. EDML will then estimate parameters by solving an exact optimization problem on each parameter island as shown in Section 3.4. The estimated parameters are then used as the new values of CPTs for clones in example islands. This process repeats until convergence.

We have shown in the previous section that the CPTs of clones are in one-to-one correspondence with the parameters of the base network. We have also shown that soft evidence, as given by Equation 3.3, can be computed by evaluating Equation 3.1 of Algorithm 2 (with parameters θ corresponding to the CPTs of clones in an example island). EDML takes advantage of this correspondence, leading to the simplified statement spelled out in Algorithm 2.

3.6 Some Properties of EDML

Being an approximate inference method, one can sometimes identify good behaviors of EDML by identifying situations under which the underlying inference algorithm will produce high quality approximations. We provide a result in this section that illustrates this point in the extreme, where EDML is guaranteed to return optimal estimates and in only one iteration. Our result relies on the following observation about parameter estimation via inference on a meta network.

When the parents $\mathbf{U}$ of a variable X are observed to $\mathbf{u}$ in an example $\mathbf{d}_i$, all edges $\theta_{x|\mathbf{u}\star} \longrightarrow X^i$ in the meta network become superfluous and can be pruned, except for the one edge that satisfies $\mathbf{u}\star = \mathbf{u}$. Moreover, edges outgoing from observed nodes can also be pruned from a meta network. Suppose now that the parents of each variable are observed in a dataset. After pruning edges as indicated earlier, each parameter variable $\theta_{x|\mathbf{u}}$ will end up being the root of an isolated naive Bayes structure that has some variables X^i as its children (those whose parents are instantiated to $\mathbf{u}$ in example $\mathbf{d}_i$). Figure 3.5 depicts the result of such pruning in the meta network of Figure 3.2, given a dataset with $H^1 = \bar{h}$, $H^2 = h$ and $H^3 = \bar{h}$.

The above observation implies that when the parents of each variable are observed in a dataset, parameters can be estimated independently. This leads to the following well known result.

Proposition 1 *When the dataset is complete, the ML estimate for parameter $\theta_{x|\mathbf{u}}$ is unique and given by $\mathcal{D}\#(x\mathbf{u})/\mathcal{D}\#(\mathbf{u})$, where $\mathcal{D}\#(x\mathbf{u})$ is the number of examples containing $x\mathbf{u}$ and $\mathcal{D}\#(\mathbf{u})$ is the number of examples containing $\mathbf{u}$.*

It is well known that EM returns such estimates and in only one iteration (i.e., independently of its seed). The following more general result is also implied by our earlier observation.

Proposition 2 *When only leaf variables have missing values in a dataset, the ML estimate for each parameter $\theta_{x|\mathbf{u}}$ is unique and given by $\mathcal{D}\#(x\mathbf{u})/\mathcal{D}^{+}\#(\mathbf{u})$. Here, $\mathcal{D}^{+}\#(\mathbf{u})$ is the number of examples containing $\mathbf{u}$ and in which X is observed.*

We can now prove the following property of EDML, which is not satisfied by EM, as we show next.

Theorem 1 *When only leaf variables have missing values in a dataset, EDML returns the unique ML estimates given by Proposition 2 and in only one iteration.*

Proof Consider an example $\mathbf{d}_i$ that fixes the values of parents $\mathbf{U}$ for variable X and consider Equation 3.1. First, $\kappa^i_{x|\mathbf{u}} = 1$ iff example $\mathbf{d}_i$ is inconsistent with $\mathbf{u}$ or does not set the value of X. Next, $\kappa^i_{x|\mathbf{u}} = 0$ iff example $\mathbf{d}_i$ contains $\bar{x}\mathbf{u}$. Finally, $\kappa^i_{x|\mathbf{u}} = \infty$ iff example $\mathbf{d}_i$ contains $x\mathbf{u}$. Moreover, these values are independent of the EDML seed so the algorithm converges in one iteration. Given these values of the Bayes factors, Equation 3.2 leads to the estimate of Proposition 2. $\Box$

We have a number of observations about this result. First, since Proposition 1 is implied by Proposition 2, EDML returns the unique ML estimates in only one iteration when the dataset is complete (just like EM). Next, when only the values of leaf variables are missing in a dataset, Proposition 2 says that there is a unique ML estimate for each network parameter. Moreover, Theorem 1 says that EDML returns these unique estimates and in only one iteration. Finally, Theorem 1 does not hold for EM. In particular, one can show that under the conditions of this theorem, an EM iteration will update its current parameter estimates θ and return the following estimates for $\theta_{x|\mathbf{u}}$:

$$\frac{\mathcal{D}\#(x\mathbf{u}) + \mathcal{D}^{-}\#(\mathbf{u})Pr(x|\mathbf{u})}{\mathcal{D}\#(\mathbf{u})}.$$

Here, $\mathcal{D}^{-}\#(\mathbf{u})$ is the number of examples that contain $\mathbf{u}$ and in which the value of X is missing. This next estimate clearly depends on the current parameter estimates. As a result, the behavior of EM will depend on its initial seed, unlike EDML.

When only the values of leaf variables are missing, there is a unique optimal solution as shown by Proposition 2. Since EM is known to converge to a local optimum, it will eventually return the optimal estimates as well, but possibly after some number of iterations. In this case, the difference between EM and EDML is simply in the speed of convergence.

Theorem 1 clearly suggests better convergence behavior of EDML over EM in some situations. We next present initial experiments supporting this suggestion.

3.7 More on Convergence

We highlight now a few empirical properties of EDML. In particular, we show how EDML can sometimes find higher quality estimates than EM, in fewer iterations and also in less time.

We highlight different types of relative convergence behavior in Figure 3.6, which depicts example runs on a selection of networks: `spect`, `win95pts`, `emdec6g`, and `tcc4e`. Network `spect` is a naive Bayes network induced from a dataset in the UCI ML repository, with 1 class variable and 22 attributes. Network `win95pts` (76 variables) is an expert system for printer troubleshooting in Windows 95. Networks `emdec6g` (168 variables) and `tcc4e` (98 variables) are noisy-or networks for diagnosis (courtesy of HRL Laboratories).

We simulated datasets of size 2^k, using the original CPT parameters of the respective networks, and then used EDML and EM to learn new parameters for a network with the same structure. We assumed that certain variables were hidden (latent); in Figure 3.6, we randomly chose $\frac{1}{4}$ of the variables to be hidden. Hidden nodes are of particular interest to EM, because it has been observed that local extrema and convergence rates can be problematic for EM here; see, for example (Elidan & Friedman, 2005; Salakhutdinov, Roweis, & Ghahramani, 2003).

In Figure 3.6, each plot represents a simulated data set of size 2^{10}, where EDML and

EM have been initialized with the same random parameter seeds. Both algorithms were run for a fixed number of iterations, 1024 in this case, and we observed the quality of the parameter estimates found, with respect to the log posterior probability (which has been normalized so that the maximum log probability observed is 0.0). We assumed a Beta prior with exponents 2. EDML damped its parameter updates by a factor of $\frac{1}{2}$, which is typical for (loopy) belief propagation algorithms.[3]

In the left column of Figure 3.6, we evaluated the quality of estimates over iterations of EDML and EM. In these examples, EDML (represented by a solid red line) tended to have better quality estimates from iteration to iteration (curves that are higher are better), and further managed to find them in fewer iterations (curves to the left are faster).[4] This is most dramatic in network `spect`, where EDML appears to have converged almost immediately, whereas EM spent a significant number of iterations to reach estimates of comparable quality. As most nodes hidden in network `spect` were leaf nodes, this may be expected due to the considerations from the previous section.

In the right column of Figure 3.6, we evaluated the quality of estimates, now in terms of *time*. We remark again that procedurally, EDML and EM are very similar, and each algorithm needs only one evaluation of the jointree algorithm per distinct example in the data set (per iteration). EDML solves an optimization problem per distinct example, whereas EM has a closed-form update equation in the corresponding step (Line 4 in Algorithms 1 and 2). Although this optimization problem is a simple one, EDML does require more time per iteration than EM. The right column of Figure 3.6 suggests that EDML can still find better estimates faster, especially in the cases where EDML has converged in significantly fewer iterations. In network `emdec6g`, we find that although EDML appeared to converge in fewer iterations, EM was able to find better estimates in less time. We anticipate in larger networks with higher treewidth, the time

[3]The simple bisection method suffices for the optimization sub-problem in EDML for binary Bayesian networks. In our current implementation, we used the conjugate gradient method, with a convergence threshold of 10^{-8}.

[4]We omit the results of the first 10 iterations as initial parameter estimates are relatively poor, which make the plots difficult to read.

spent in the simple optimization sub-problem will be dominated by the time to perform jointree propagation.

We also performed experiments on networks learned from binary haplotype data (Elidan & Gould, 2008), which are networks with bounded treewidth. Here, we simulated data sets of size 2^{10}, where we again randomly selected $\frac{1}{4}$ of the variables to be hidden. We further ran EDML and EM for a fixed number of iterations (512, here). For each of the 74 networks available, we ran EDML and EM with 3 random seeds, for a total of 222 cases. In Figure 3.7, we highlight a selection of the runs we performed, to illustrate examples of relative convergence behaviors. Again, in the first row, we see a case where EDML identifies better estimates in fewer iterations and less time. In the next two rows, we highlight two cases where EDML appears to converge to a superior fixed point than the one that EM appears to converge to. In the last row, we highlight an instance where EM instead converges to a superior estimate. In Figure 3.8, we compare the estimates of EDML and EM at each iteration, computing the percentage of the $74 \times 3 = 222$ cases considered, where EDML had estimates no worse than those found by EM. In this set of experiments, the estimates identified by EDML are clearly superior (or at least, no worse in most cases), when compared to EM.

We remark however, that when both algorithms are given enough iterations to converge, we have observed that the quality of the estimates found by both algorithms are often comparable. This is evident in Figure 3.6, for example. The analysis from the previous section indicates however that there are (very specialized) situations where EDML would be clearly preferred over EM. One subject of future study is the identification of situations and applications where EDML would be preferred in practice as well.

3.8 Related Work

EM has played a critical role in learning probabilistic graphical models and Bayesian networks (Dempster et al., 1977; Lauritzen, 1995; Heckerman, 1998). However learning (and Bayesian learning in particular) remains challenging in a variety of situations, particularly when there are hidden (latent) variables; see, e.g., (Elidan, Ninio, Friedman, & Shuurmans, 2002; Elidan & Friedman, 2005). Slow convergence of EM has also been recognized, particularly in the presence of hidden variables. A variety of techniques, some incorporating more traditional approaches to optimization, have been proposed in the literature; see, e.g., (Thiesson, Meek, & Heckerman, 2001).

Variational approaches are an increasingly popular formalism for learning tasks as well, and for topic models in particular, where variational alternatives to EM are used to maximize a lower bound on the log likelihood (Blei, Ng, & Jordan, 2003). Expectation Propagation also provides variations of EM (Minka & Lafferty, 2002) and is closely related to (loopy) belief propagation (Minka, 2001).

Our empirical results have been restricted to a preliminary investigation of the convergence of EDML, in contrast to EM. A more comprehensive evaluation is called for in relation to both EM and other approaches based on Bayesian inference. We have also focused this chapter on binary variables. EDML, however, generalizes to multivalued variables since edge deletion does not require a restriction to binary variables and the key result of Section 3.4 also generalizes to multivalued variables. The resulting formulation is less transparent though when compared to the binary case since Bayes factors no longer apply directly and one must appeal to a more complex method for quantifying soft evidence; see (Chan & Darwiche, 2005).

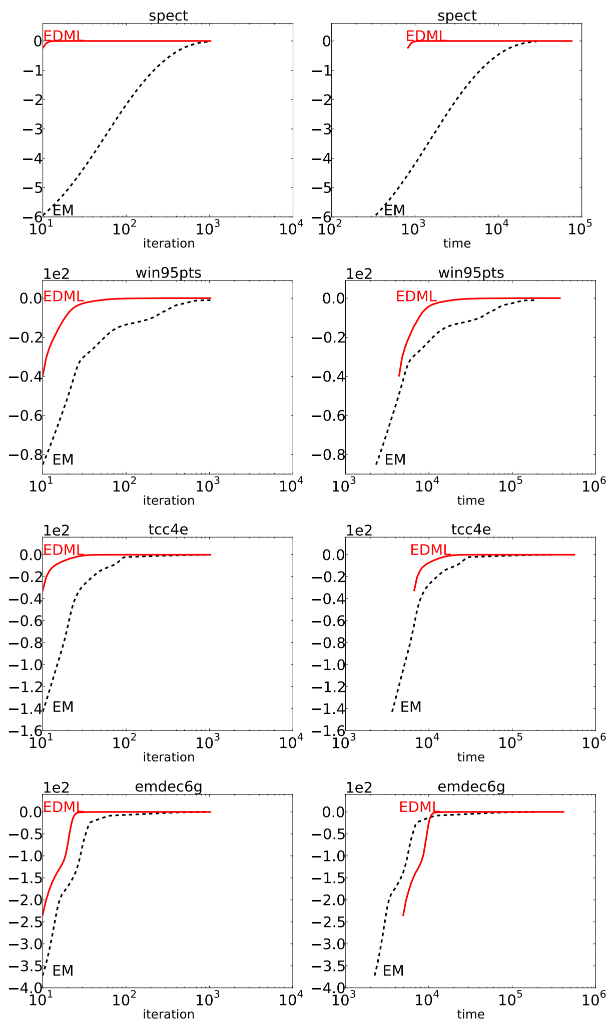


Figure 3.6: Quality of parameter estimates over iterations (left column) and time (right column). Going right on the x-axis, we have increasing iterations and time. Going up on the y-axis, we have increasing quality of parameter estimates. EDML is depicted with a solid red line, and EM with a dashed black line.

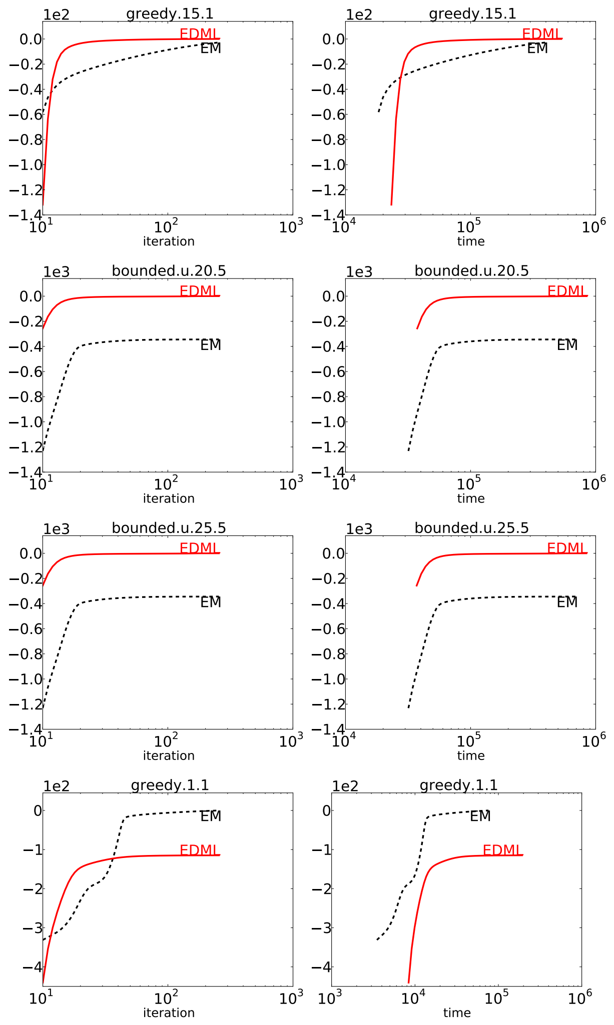


Figure 3.7: Quality of parameter estimates over iterations (left column) and time (right column). Going right on the x-axis, we have increasing iterations and time. Going up the y-axis, we have increasing quality of parameter estimates. EDML is depicted with a solid red line, and EM with a dashed black line.

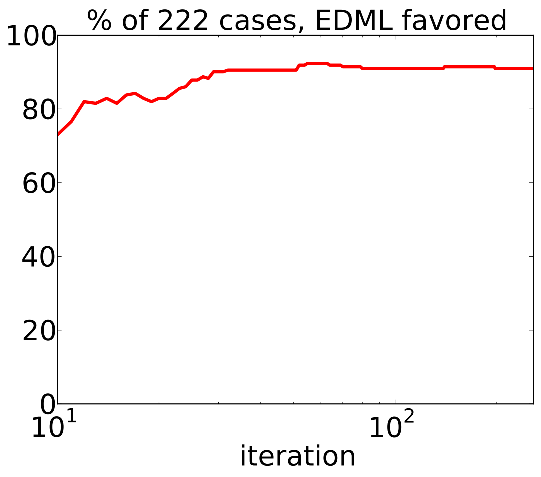


Figure 3.8: Quality of EDML estimates over 74 networks (3 cases each) induced from binary haplotype data. Going right on the x-axis, we have increasing iterations. Going up the y-axis, we have an increasing percentage of instances where EDML's estimates were no worse than those given by EM.

CHAPTER 4

Advances and Theoretical Insight into EDML

We provide a simple characterization of EDML that enables us to prove that its fixed points are exactly the stationary points of the likelihood (MAP) function. Furthermore, we modify EDML to guarantee improving the likelihood (MAP), at every step. In addition to this, we generalize EDML to support multivalued variables. We also propose a simple iterative method for solving its optimization problems. Finally, we provide experimental results suggesting that the modified EDML can sometimes be faster, in running time, than EM. This chapter is based on (Refaat et al., 2012).

4.1 Introduction

In this chapter, we present a number of new results and insights on the EDML algorithm. First, we provide a simple extension of EDML to Bayesian networks over multivalued variables. We also show that the convex optimization problem that underlies the binary version of EDML, remains convex for the multivalued case.

Next, we identify a new and simplified characterization of EDML, which facilitates a number of theoretical observations about EDML. For example, this new characterization implies a simple, fixed-point iterative algorithm for solving the convex optimization problems underlying EDML. Moreover, we show that this fixed-point algorithm monotonically improves the solutions of these convex optimization problems, which correspond to an approximate factorization of the posterior over network parameters.

Armed with this new characterization of EDML, we go on to identify a surprising

Algorithm 3 EM

```
input:
  $G$: A Bayesian network structure
  $\mathcal{D}$: An incomplete dataset $\mathbf{d}_1, \ldots, \mathbf{d}_N$
  $\theta$: An initial parameterization of structure $G$
  $\psi$: A Dirichlet prior for each parameter set $\theta_{X|\mathbf{u}}$

1: while not converged do
2:     $Pr \leftarrow$ distribution induced by $\theta$ and $G$
3:     Compute probabilities:
           $Pr(x\mathbf{u}|\mathbf{d}_i)$ and $Pr(\mathbf{u}|\mathbf{d}_i)$
       for each family instantiation $x\mathbf{u}$ and example $\mathbf{d}_i$
4:     Update parameters:
           $\theta_{x|\mathbf{u}} \leftarrow \frac{\psi_{x|\mathbf{u}} - 1 + \sum_{i=1}^{N} Pr(x\mathbf{u}|\mathbf{d}_i)}{\psi_{X|\mathbf{u}} - |X| + \sum_{i=1}^{N} Pr(\mathbf{u}|\mathbf{d}_i)}$
5: return parameterization $\theta$
```

Algorithm 4 Multivalued EDML

```
input:
  $G$: A Bayesian network structure
  $\mathcal{D}$: An incomplete dataset $\mathbf{d}_1, \ldots, \mathbf{d}_N$
  $\theta$: An initial parameterization of structure $G$
  $\psi$: A Dirichlet prior for each parameter set $\theta_{X|\mathbf{u}}$

1: while not converged do
2:     $Pr \leftarrow$ distribution induced by $\theta$ and $G$
3:     Compute soft evidence parameters:
           $\lambda^i_{x|\mathbf{u}} \leftarrow Pr(x\mathbf{u}|\mathbf{d}_i)/Pr(x|\mathbf{u}) - Pr(\mathbf{u}|\mathbf{d}_i) + 1$    (4.1)
       for each family instantiation $x\mathbf{u}$ and example $\mathbf{d}_i$
4:     Update parameters:
           $\theta_{X|\mathbf{u}} \leftarrow \underset{\hat{\theta}_{X|\mathbf{u}}}{\operatorname{argmax}} \prod_{x} [\hat{\theta}_{x|\mathbf{u}}]^{\psi_{x|\mathbf{u}}-1} \prod_{i=1}^{N} \sum_{x} \lambda^i_{x|\mathbf{u}} \hat{\theta}_{x|\mathbf{u}}$    (4.2)
5: return parameterization $\theta$
```

connection between EDML and EM (considering their theoretical and practical differences). In particular, we show that a fixed point of EDML is a fixed point of EM, and vice versa. This observation has a number of implications. First, it provides a new perspective on EM fixed points, based on the semantics of EDML, which was originally inspired by an approximate inference algorithm for Bayesian networks that subsumed the influential loopy belief propagation algorithm as a degenerate case (Pearl, 1988; Choi & Darwiche, 2006). Second, it suggests a hybrid EDML/EM algorithm that seeks to take advantage of the desirable properties from each: the improved convergence behavior of EDML, and the monotonic improvement property of EM.

4.2 Multivalued EDML

One contribution of this chapter is the extension of binary EDML so that it handles multivalued variables as well. In principle, the extension turns out to be straightforward and is depicted in Algorithm 4. However, two issues require further discussion. The first concerns the specification of soft evidence for multivalued variables. The second is confirming that the optimization problem corresponding to a parameter island (on Line 4 of Algorithm 4) remains strictly concave, therefore, admitting unique solutions. We will consider both issues next.

4.2.1 Examples as Soft Evidence

The first key concept of EDML is to interpret a data example $\mathbf{d}_i$ in the dataset as soft evidence on a conditional random variable $X|\mathbf{u}$. As mentioned earlier, soft evidence on a variable is modeled using a vector of parameters, one for each value of the variable. We will therefore use $\lambda_{x|\mathbf{u}}$ to denote the parameter pertaining to value x of variable $X|\mathbf{u}$.

EDML uses Equation 4.1 in Algorithm 4 to compute soft evidence. In particular, example $\mathbf{d}_i$ is viewed as soft evidence on conditional random variable $X|\mathbf{u}$ that is quantified as follows:

$$\lambda^i_{x|\mathbf{u}} \leftarrow Pr(x\mathbf{u}|\mathbf{d}_i)/Pr(x|\mathbf{u}) - Pr(\mathbf{u}|\mathbf{d}_i) + 1$$

We will not derive this equation here as it resembles the one for binary EDML. We will, however, discuss some of its key properties in this and further sections.

Consider the case when the example $\mathbf{d}_i$ is inconsistent with the parent instantiation $\mathbf{u}$. In this case, the example should be irrelevant to variable $X|\mathbf{u}$. Equation 4.1 does the right thing here as it reduces to 1 for all values of x, which amounts to neutral evidence.

Another special case is when example $\mathbf{d}_i$ is complete; that is, it has no missing values. In this case, one can verify that if $\mathbf{d}_i$ is consistent with $\mathbf{u}$ (relevant), then $\lambda^i_{x|\mathbf{u}} = 0$ for all values x except the value $x^\star$ consistent with $\mathbf{d}_i$. This is equivalent to hard evidence in favor of $x^\star$. On the other hand, if $\mathbf{d}_i$ is inconsistent with $\mathbf{u}$ (irrelevant), then

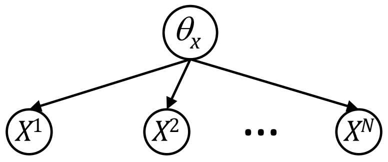


Figure 4.1: Learning from independent, hard observations $X^1, \ldots, X^N$. The distribution of variable X is specified by parameter set θ_X.

$\lambda^i_{x|\mathbf{u}} = 1$ for all values x, providing neutral evidence. In a nutshell, a complete example provides either hard evidence, if it is relevant; or neutral evidence, if it is irrelevant.

4.2.2 Learning from Soft Evidence

Consider the standard learning problem depicted in Figure 4.1. Here, we have a variable X that takes k values $x_1, \ldots, x_k$ and has a distribution specified by a parameter set $\theta_X : \theta_{x_1}, \ldots, \theta_{x_k}$. That is, each parameter θ_{x_i} represents the corresponding probability $Pr(X = x_i)$. Suppose further that we have a Dirichlet prior $\rho(\theta_X)$ on the parameter set with exponents ψ_{x_i} greater than one. A standard learning problem here is to compute the MAP estimates of parameter set θ_X given N independent observations on the variable X. MAP estimates are known to be unique in this case and have a corresponding closed form. In particular, it is known that the posterior $\rho(\theta_X|X^1, \ldots, X^N)$ is a unimodal Dirichlet and, hence, has a unique maximum; see for example (Darwiche, 2009).

EDML is based on a variant of this learning problem in which we are given N *soft observations* on variable X instead of hard observations. This variant is shown in Figure 4.2, where each observation X^i has a child η^i that is used to emulate soft evidence on X^i. That is, to represent soft evidence $\lambda^i_{x_1}, \ldots, \lambda^i_{x_k}$, we simply choose the CPT for η^i so that $Pr(\eta^i|x_1) : \cdots : Pr(\eta^i|x_k) = \lambda^i_{x_1} : \cdots : \lambda^i_{x_k}$.

Note here that the posterior density $\rho(\theta_X|\eta^1 \ldots \eta^N)$ is no longer Dirichlet, but takes the more complex form given by Equation 4.2 of Algorithm 4. Yet, we have the fol-

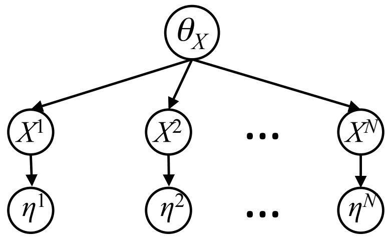


Figure 4.2: Learning from independent, soft observations $\eta_1, \ldots, \eta_N$. The distribution of variable X is specified by parameter set θ_X.

lowing result.

Theorem 2 *Given N soft observations η^i on a variable X, and a Dirichlet prior on its parameters θ_X, with Dirichlet exponents $\psi_x > 1$, the posterior density $\rho(\theta_X|\eta^1 \ldots \eta^N)$ is strictly log concave.*

Therefore, the posterior density, $\rho(\theta_X|\eta^1 \ldots \eta^N)$, has a unique maximum. We next provide a simple iterative method for obtaining the maximum in this case.

4.3 Simple EDML

Multivalued EDML, as given in Algorithm 4, works as follows. We start with some initial parameter estimates, just like EM. We then iterate, while performing two steps in each iteration. In the first step, each example $\mathbf{d}_i$ in the dataset is used to compute soft evidence on each variable $X|\mathbf{u}$, as in Equation 4.1 of Algorithm 4. Using this soft evidence, a learning problem is set up for the parameter set $\theta_{X|\mathbf{u}}$ as given in Figure 4.2, which is a learning *sub*-problem in the context of Equation 4.2 of Algorithm 4. The solution to this learning sub-problem provides the next estimate for parameter set $\theta_{X|\mathbf{u}}$. The process repeats. In principle, any appropriate optimization algorithm could be used to solve each of these learning sub-problems.

We will next derive a simpler description of EDML that has two key components.

First, we will provide a convergent update equation that iteratively solves the optimization problem in Equation 4.2 of Algorithm 4. Second, we will use this update equation to provide a characterization of EDML's fixed points.

First, consider the likelihood:

$$\begin{aligned} Pr(\eta^1, \ldots, \eta^N \mid \theta_X) &= \prod_{i=1}^{N} Pr(\eta^i \mid \theta_X) \\ &= \prod_{i=1}^{N} \sum_x Pr(\eta^i \mid x) Pr(x \mid \theta_X) = \prod_{i=1}^{N} \sum_x \lambda_x^i \theta_x \end{aligned}$$

Next, we have the posterior:

$$\begin{aligned} \rho(\theta_X \mid \eta^1, \ldots, \eta^N) &\propto \rho(\theta_X) Pr(\eta^1, \ldots, \eta^N \mid \theta_X) \\ &= \rho(\theta_X) \prod_{i=1}^{N} \sum_x \lambda_x^i \theta_x \end{aligned}$$

Assuming a Dirichlet prior over parameter set θ_X, with Dirichlet exponents ψ_x, the log of the posterior is:

$$\begin{aligned} &\log \rho(\theta_X \mid \eta^1 \ldots \eta^N) \\ &= \sum_x (\psi_x - 1) \log \theta_x + \sum_{i=1}^{N} \log \sum_x \lambda_x^i \theta_x + \gamma \end{aligned}$$

where γ is a constant that is independent of θ_X, which we can ignore. By Theorem 2 we know that the log of the posterior is strictly concave when $\psi_x > 1$.

To get the unique maximum of the log posterior, we solve the optimization problem:

$$\begin{aligned} \text{minimize} \quad & -\log \rho(\theta_X \mid \eta^1, \ldots, \eta^N) \\ \text{subject to} \quad & \sum_x \theta_x = 1 \end{aligned}$$

In order to characterize the unique maximum of the log posterior, we start by taking the Lagrangian:

$$L(\theta_X, \overline{\mathbf{u}}) = -\log \rho(\theta_X \mid \eta^1, \ldots, \eta^N) + \overline{\mathbf{u}} \cdot (\sum_x \theta_x - 1)$$

where $\overline{\mathbf{u}}$ is a Lagrange multiplier. We get the following condition for the optimal parameter estimates, by setting the gradient of $L(\theta_X, \overline{\mathbf{u}})$ with respect to θ_X to zero:

$$\theta_x = \frac{\psi_x - 1 + \sum_{i=1}^{N} \frac{\lambda_x^i \theta_x}{\sum_{x^*} \lambda_{x^*}^i \theta_{x^*}}}{\psi_X - |X| + N}$$

where $\psi_X = \sum_x \psi_x$, and where we used the constraint $\sum_x \theta_x = 1$ to identify that $\overline{\mathbf{u}} = \psi_X - |X| + N$.

The above equation leads to a much stronger result.

Theorem 3 *The following update equation monotonically increases the posterior* $\rho(\theta_{X|\mathbf{u}} \mid \eta^1, \ldots, \eta^N)$*:*

$$\theta_{x|\mathbf{u}}^t = \frac{\psi_{x|\mathbf{u}} - 1 + \sum_{i=1}^{N} \frac{\lambda_{x|\mathbf{u}}^i \theta_{x|\mathbf{u}}^{t-1}}{\sum_{x^\star} \lambda_{x^\star|\mathbf{u}}^i \theta_{x^\star|\mathbf{u}}^{t-1}}}{\psi_{X|\mathbf{u}} - |X| + N} \tag{4.3}$$

This theorem suggests a convergent iterative algorithm for solving the convex optimization problem of Equation 4.2 in Algorithm 4. First, we start with some initial parameter estimates $\theta_{x|\mathbf{u}}^0$ at iteration $t = 0$. For iteration $t > 0$, we use the above update to compute parameters $\theta_{x|\mathbf{u}}^t$ given the parameters $\theta_{x|\mathbf{u}}^{t-1}$ from the previous iteration. If at some point, the parameters of one iteration do not change in the next (in practice, up to some limit), we say that the iterations have converged to a fixed point. The above theorem, together with Theorem 2, shows that these updates are convergent to the unique maximum of the posterior.

Given Equation 4.3, one can think of two types of *iterations* in EDML: local and global. A *global* iteration corresponds to executing Lines 2–4 of Algorithm 4 and is similar to an EM iteration. Within each global iteration, we have *local* iterations which correspond to the evaluations of Equation 4.3. Note that each parameter set $\theta_{X|\mathbf{u}}$ has its own local iterations, which are meant to find the optimal values of this parameter set. Moreover, the number of local iterations for each parameter set $\theta_{X|\mathbf{u}}$ may be different, depending on the soft evidence pertaining to that set (i.e., $\lambda_{x|\mathbf{u}}^i$) and depending on how Equation 4.3 is seeded for that particular parameter set (i.e., $\theta_{x|\mathbf{u}}^0$).

This leads to a number of observations on the difference between EM updates (Equation 2.1) and EDML updates (Equation 4.3). From a time complexity viewpoint, an EM update implies exactly one local iteration for each parameter set since Equation 2.1 needs to be evaluated only once for each parameter set. As mentioned earlier, however, an EDML update requires a varying number of local iterations. We have indeed observed that some parameter sets may require several hundred local iterations, depending on the seed of Equation 4.3 and the convergence criteria used. Another important observation is that EDML has a secondary set of seeds, as compared to EM, which are needed to start off Equation 4.3 at the beginning of each global iteration of EDML. In our experiments, we seed Equation 4.3 using the parameter estimates obtained from the previous global iteration of EDML. We note, however, that the choice of these secondary seeds is a subject that can significantly benefit from further research.

4.4 EDML Fixed Points

One of the more well known facts about EM is that its fixed points are precisely the stationary points of the log-likelihood function (or more generally, the posterior density when Dirichlet priors are used). This property has a number of implications, one of which is that EM is capable of converging to every local maxima of the log-likelihood, assuming that the algorithm is seeded appropriately.

In this section, we show that the fixed points of EDML are precisely the fixed points of EM. We start by formally defining what a fixed point is.

Both EM and EDML can be viewed as functions $f(\theta)$ that take a network parameterization θ and returns another network parameterization $f(\theta)$. Each algorithm is seeded with initial parameters θ^0. After the first iteration, each algorithm produces the next parameters $\theta^1 = f(\theta^0)$. More generally, at iteration i, each algorithm produces the parameters $\theta^{i+1} = f(\theta^i)$. When $\theta^{i+1} = \theta^i$, we say that parameters θ^i are a fixed point for an algorithm. We also say that the algorithm has converged to θ^i. We now have the following results.

Theorem 4 *A parameterization θ is a fixed point for EDML if and only if it is a fixed point for EM.*

The proof of this theorem rests on two observations. First, using the update equation of EM given on Line 4 of Algorithm 3, one immediately gets that the EM fixed points are characterized by the following equation

$$Pr(x|\mathbf{u}) = \frac{\psi_{x|\mathbf{u}} - 1 + \sum_{i=1}^{N} Pr(x\mathbf{u}|\mathbf{d}_i)}{\psi_{X|\mathbf{u}} - |X| + \sum_{i=1}^{N} Pr(\mathbf{u}|\mathbf{d}_i)} \tag{4.4}$$

That is, we can test whether a network parameterization θ is a fixed point for EM by simply checking the probability distribution Pr it induces to see if satisfies the above equation (this is actually a set of equations, one for each family instantiation $x\mathbf{u}$ in the network).

Consider now EDML updates as given by Equation 4.3 and suppose that we have reached a fixed point, where $\theta^{t}_{x|\mathbf{u}} = \theta^{t-1}_{x|\mathbf{u}} = Pr(x|\mathbf{u})$ for all $x\mathbf{u}$. If we now replace each $\lambda^{i}_{x|\mathbf{u}}$ by its corresponding value in Equation 4.1 of Algorithm 4, we obtain, after some simplification:

$$\begin{aligned} Pr(x|\mathbf{u}) &= \frac{\psi_{x|\mathbf{u}} - 1 + \sum_{i=1}^{N} \frac{\lambda^{i}_{x|\mathbf{u}} Pr(x|\mathbf{u})}{\sum_{x^\star} \lambda^{i}_{x^\star|\mathbf{u}} Pr(x^\star|\mathbf{u})}}{\psi_{X|\mathbf{u}} - |X| + N} \\ &= \frac{\psi_{x|\mathbf{u}} - 1 + \sum_{i=1}^{N} Pr(x\mathbf{u}|\mathbf{d}_i) + Pr(\neg\mathbf{u}|\mathbf{d}_i) Pr(x|\mathbf{u})}{\psi_{X|\mathbf{u}} - |X| + N} \end{aligned} \tag{4.5}$$

Thus, a parameterization θ is a fixed point for EDML iff it satisfies Equation 4.5. After noting that $N = \sum_{i=1}^{N} Pr(\mathbf{u}|\mathbf{d}_i) + Pr(\neg\mathbf{u}|\mathbf{d}_i)$, we rearrange Equation 4.5 to obtain Equation 4.4, which is the characteristic equation for EM fixed points. Thus, a parameterization θ satisfies Equation 4.5 iff it is a fixed point for EM.

4.5 Hybrid EDML/EM

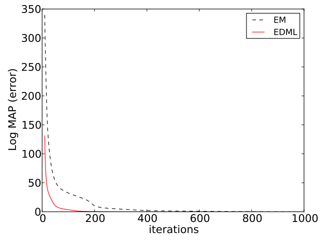
By Theorem 4, EDML and EM share the same fixed points. This result is fairly surprising, considering the theoretical and practical differences between the two algorithms. For example, certain specialized situations were identified where EDML converges to optimal parameter estimates in a single global iteration (regardless of how it is seeded), whereas EM may require many iterations to converge to the same estimates (Choi et al., 2011). On the other hand, EDML is not guaranteed to monotonically improve its estimates after each global iteration as EM does (improvement here is in terms of increasing the likelihood of estimates, or the MAP when Dirichlet priors are used).

By carefully examining EDML updates (Equation 4.3), one can see that the quantities it needs to perform a single local iteration are the same quantities needed to perform an EM update (Equation 2.1). Hence, without any additional computational effort, one can obtain EM updates as a side effect of computing EDML updates (the converse is not true since Equation 4.3 may require many local iterations). As both algorithms share the same fixed points, it thus makes sense to consider a hybrid algorithm that takes advantage of the improved theoretical and practical benefits of EDML (with respect to faster convergence) and the monotonic improvement property of EM.

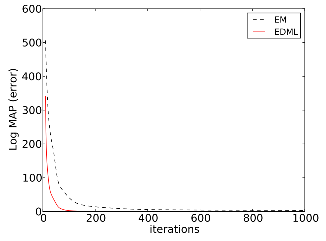
We thus propose a very simple hybrid algorithm. At each global iteration of EDML, we also compute EM updates simultaneously. We then evaluate each update and choose the one that increases the posterior the most. As EM is guaranteed to improve the posterior, this hybrid algorithm is also trivially guaranteed to monotonically improve the posterior, therefore, inheriting the most celebrated feature of EM.

While additionally computing an EM update is not much overhead if one is computing an EDML update, evaluating both updates with respect to the posterior incurs a non-trivial cost. As we shall see in the following section, however, the improved convergence behavior of EDML enables this hybrid algorithm to realize improvements over EM, both in terms of faster convergence (i.e., number of global iterations) and in terms of time (i.e., when taking both global and local iterations under consideration).

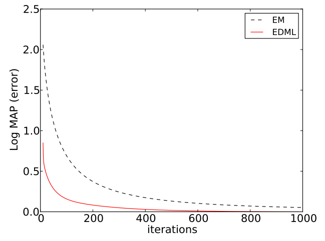
4.6 Experimental Results



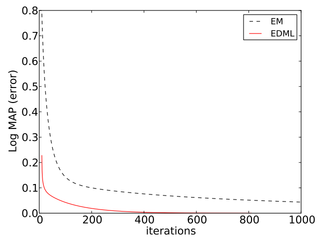
(a) andes



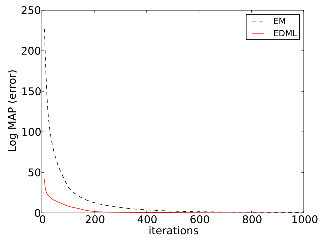
(b) andes



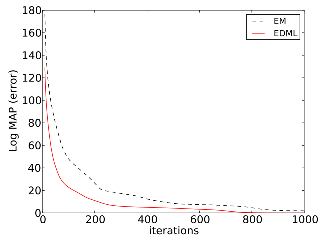
(c) asia



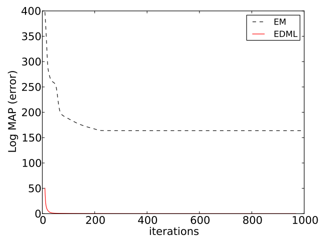
(d) asia



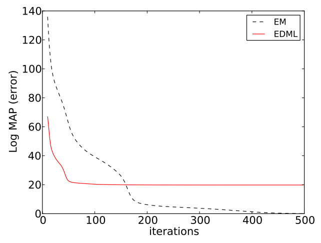
(e) diagnose



(f) diagnose



(g) alarm



(h) alarm

Figure 4.3: MAP Error of parameter estimates over iterations. Going right on the x-axis, we have increasing iterations. Going up on the y-axis, we have increasing error. EDML is depicted with a solid red line, and EM with a dashed black line. The curves are, in pairs, for the networks: andes, asia, diagnose, and alarm. Each pair of curves represents a selection of two different datasets of size 2^{10}.

In our first set of experiments, we show that simple EDML, as compared to EM, can often find better estimates in fewer global iterations. In our second set of experiments, we show that a hybrid EDML/EM algorithm can find better estimates in less time, as well as fewer global iterations. We use the following networks: alarm, andes, asia, diagnose, pigs, spect, water, and win95pts. Network spect is a naive Bayes network induced from a dataset in the UCI ML repository, with 1 class variable and 22 attributes. Network diagnose is from the UAI 2008 evaluation. The other networks are commonly used benchmarks.[1]

Using these networks, we simulated data sets of a certain size (2^{10}), then made the data incomplete by randomly selecting a certain percentage of the nodes to be hidden (10%, 25%, 35%, 50%, and 70%). For each of these cases, and for each network, we further generated 3 data sets at random. A combination of a network, a percentage of hidden nodes, and a generated data set constitutes a learning problem. Both EM and EDML are seeded with the same set of randomly generated parameters. Local EDML iterations are seeded with the estimates of the previous global iteration.

4.6.1 Experiments I

First, we study the behavior of EDML compared to EM, with respect to global iterations (more on the computation time, in the next section). For every learning problem, we run both EM and EDML[2] for 1000 global iterations and identify the best MAP estimates achieved by either of them, for the purpose of evaluation. In each global iteration, EM and EDML each try to improve their current estimates by improving the posterior (again, EM is provably guaranteed to increase the posterior, while EDML is not). The difference between the (log) posterior of the current estimates, and the best (log) posterior found, by either algorithm, is considered to be the error. The error is measured at every global iteration of EM and EDML until it decreases below 10^{-4}. Table 4.1 summarizes the results by showing the percentage of global iterations in which each algorithm had less error than the other. In the global iterations where an algorithm had

[1]Available at `http://www.cs.huji.ac.il/site/labs/compbio/Repository/` and `http://genie.sis.pitt.edu/networks.html`

[2]Soft evidence and parameter updates are damped in EDML, which is typical for algorithms like loopy belief propagation, which EDML is, in part, inspired by (Choi & Darwiche, 2006).

less error, the factor by which it decreases the error of the other algorithm, on average, is computed, and is considered the relative improvement: r and r', for EM and EDML, respectively. Table 4.1 shows the results for three different breakdowns: (1) by different networks, (2) by different hiding percentages, and (3) on average.

We see here that EDML can obtain better estimates than EM in much fewer global iterations. Interestingly, this is the case even though EDML is not guaranteed to improve estimates after each global iteration, as EM does. Another interesting observation is that decreasing the percentage of hidden nodes widens the gap between EDML and EM, in favor of EDML. This is not surprising though since the approximate inference scheme on which EDML is based becomes more accurate with more observations. In particular, the local optimization problems that EDML solves exactly and independently, become more independent with more observations (i.e., as the dataset becomes more complete). Figure 4.3 highlights a selection of error curves given by EM and EDML for different learning problems. One can see that in most cases shown, the EDML error goes to zero much faster than EM.

4.6.2 Experiments II

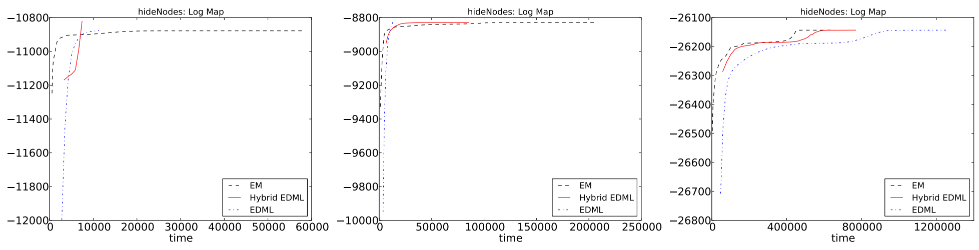


Figure 4.4: MAP of parameter estimates over time. Going right on the x-axis, we have increasing time (ms). Going up on the y-axis, we have increasing MAP. Hybrid EDML is depicted with a solid red line, EM with a dashed black line, and EDML with a blue dotted dashed line. The curves are from left to right for the following problems: alarm with hiding 25%, win95pts with hiding 35%, and water with hiding 50%.

Table 4.1: Speedup results (iterations)

category	% EDML	% EM	r	r'
alarm	89.25%	10.75%	76.21%	76.44%
andes	75.89%	24.11%	88.95%	79.29%
asia	99.01%	0.99%	92.05%	76.91%
diagnose	78.99%	21.01%	77.99%	80.18%
pigs	83.34%	16.66%	83.51%	60.57%
spect	86.65%	13.35%	82.70%	79.96%
water	82.77%	17.23%	91.55%	83.78%
win95pts	78.73%	21.27%	91.75%	79.89%
hiding 10%	93.82%	6.18%	84.59%	87.13%
hiding 25%	90.95%	9.05%	83.83%	75.70%
hiding 35%	82.24%	17.76%	86.26%	75.09%
hiding 50%	77.61%	22.39%	87.8%	80.21%
hiding 70%	75.65%	24.35%	84.48%	74.21%
average	83.05%	16.95%	85.41%	76.96%

Our first set of experiments showed that EDML can obtain better estimates in significantly fewer global iterations than EM. A global EDML iteration, however, is more costly than a global EM iteration as EDML performs local iterations which are needed to solve the convex optimization problem associated with each parameter set. Thus, EDML can potentially take more time to converge than EM in some cases, therefore reducing the overall benefit over EM, in terms of time (more on this later).[3] EDML can still perform favorably time-wise compared to EM, but we show here that a hybrid EDML/EM can go even further.

We first run EM until convergence (or until it exceeds 1000 iterations). Second, we

[3]This could be alleviated, for example, by better seeding of the local EDML iterations (to speed up convergence of the local iterations), or by performing the local EDML iterations in parallel, across parameter sets.

Table 4.2: Speedup results (time)

network	% Hybrid	% EM	s	s'
alarm	46.67%	53.33%	75.80%	56.22%
andes	53.33%	46.67%	42.27%	47.08%
asia	66.67%	33.33%	70.37%	41.00%
diagnose	26.67%	73.33%	52.71%	43.14%
pigs	73.33%	26.67%	52.35%	31.32%
spect	100%	0%	98.03%	—
water	35.71%	64.29%	46.15%	43.55%
win95pts	80.00%	20.00%	66.37%	54.95%
average	60.50%	39.50%	67.45%	45.55%

run our hybrid EDML/EM until it achieves the same quality of parameter estimates as EM (or until it exceeds 1000 iterations). To summarize the results, Table 4.2 shows the percentage of learning problems in which each algorithm was faster than the other. In the cases where the hybrid EDML/EM algorithm was faster, the average percentage by which it decreases the execution time of EM is reported as the speedup (s). The speedup for the cases in which EM was faster is given by s'.

The results suggest that hybrid EDML/EM can be used to get better estimates in less time. Specifically, on average, the hybrid method decreases the execution time by a factor of 3.07 in about 60.50% of the cases, and, in the rest of the cases, EM was faster by a factor of roughly 1.84. This is particularly interesting in light of the non-trivial overhead associated with hybrid EDML/EM, as it has to evaluate both EDML and EM estimates in order to select the better estimate. In comparison, EDML alone was on average faster than EM by a factor of 2.59 times in about 54.17% of the cases, whereas EM was faster than EDML alone by a factor of 2.38 in the remaining 45.83% of the cases.

Figure 4.4 shows a sample of the time curves showing two cases where EDML/EM reached better MAP estimates, faster than EM, and one case where EDML/EM finished slightly after EM. EDML alone is also plotted in Figure 4.4 where it is usually slower than EDML/EM except in some cases; see the center plot in Figure 4.4.

CHAPTER 5

Generalized EDML for Learning Parameters in Directed and Undirected Graphical Models

In this chapter, we propose a greatly simplified perspective on EDML, which casts it as a general approach to continuous optimization. The new perspective has several advantages. First, it makes immediate some results that were non-trivial to prove initially. Second, it facilitates the design of EDML algorithms for new graphical models, leading to a new algorithm for learning parameters in Markov networks. We derive this algorithm in this chapter, and show, empirically, that it can sometimes learn estimates more efficiently from complete data, compared to commonly used optimization methods, such as conjugate gradient and L-BFGS. This chapter is based on (Refaat et al., 2013).

5.1 Introduction

In this chapter, we propose a new perspective on EDML, which views it more abstractly in terms of a simple method for continuous optimization. This new perspective has a number of advantages, including the design of new EDML algorithms for new classes of models, where graphical formulations of parameter estimation, such as meta-networks, are lacking.

This chapter is structured as follows. In Section 5.2, we highlight a simple iterative method for, approximately, solving continuous optimization problems. In Section 5.3, we formulate the EDML algorithm for parameter estimation in Bayesian networks, as an instance of this optimization method. In Section 5.4, we derive a new EDML algorithm for Markov networks, based on the same perspective. In Section 5.5, we contrast the two EDML algorithms for directed and undirected graphical models, in the complete data case. We empirically evaluate our new algorithm for parameter estimation under complete data in Markov networks, in Section 5.6; review related work in Section 5.7; and conclude in Section 5.8. Proofs of theorems appear in the supplementary appendix.

5.2 An Approximate Optimization of Real-Valued Functions

Consider a real-valued objective function $f(x)$ whose input x is a vector of components:

$$x = (x_1, \ldots, x_i, \ldots, x_n),$$

where each component x_i is a vector in $\mathbb{R}^{k_i}$ for some k_i. Suppose further that we have a constraint on the domain of function $f(x)$ with a corresponding function g that maps an arbitrary point x to a point $g(x)$ satisfying the given constraint. We say in this case that $g(x)$ is a *feasibility function* and refer to the points in its range as *feasible points.*

Our goal here is to find a feasible input vector $x = (x_1, \ldots, x_i, \ldots, x_n)$ that optimizes the function $f(x)$. Given the difficulty of this optimization problem in general, we will settle for finding stationary points x in the constrained domain of function $f(x)$. One approach for finding such stationary points is as follows.

Let $x^\star = (x_1^\star, \ldots, x_i^\star, \ldots, x_n^\star)$ be a feasible point in the domain of function $f(x)$. For each component x_i, we define a sub-function

$$f_{x^\star}(x_i) = f(x_1^\star, \ldots, x_{i-1}^\star, x_i, x_{i+1}^\star, \ldots, x_n^\star).$$

That is, we use the n-ary function $f(x)$ to generate n sub-functions $f_{x^\star}(x_i)$. Each of these sub-functions is obtained by fixing all inputs x_j of $f(x)$, for $j \neq i$, to their values in $x^\star$, while keeping the input x_i free. We further assume that these sub-functions are subject to the same constraints that the function $f(x)$ is subject to.

We can now characterize all feasible points $x^\star$ that are stationary with respect to the function $f(x)$, in terms of local conditions on sub-functions $f_{x^\star}(x_i)$.

Claim 1 *A feasible point $x^\star = (x_1^\star, \ldots, x_i^\star, \ldots, x_n^\star)$ is stationary for function $f(x)$ iff for all i, component $x_i^\star$ is stationary for sub-function $f_{x^\star}(x_i)$.*

This is immediate from the definition of a stationary point. Assuming no constraints, at a stationary point $x^\star$, the gradient $\nabla f(x^\star) = 0$, i.e., $\nabla_{x_i} f(x^\star) = \nabla f_{x^\star}(x_i^\star) = 0$ for all x_i, where $\nabla_{x_i} f(x^\star)$ denotes the sub-vector of gradient $\nabla f(x^\star)$ with respect to component x_i.[1]

With these observations, we can now search for feasible stationary points $x^\star$ of the constrained function $f(x)$ using an iterative method that searches instead for stationary points of the constrained sub-functions $f_{x^\star}(x_i)$. The method works as follows:

1. Start with some feasible point x^t of function $f(x)$ for $t = 0$
2. While some x_i^t is not a stationary point for constrained sub-function $f_{x^t}(x_i)$
   (a) Find a stationary point y_i^{t+1} for each constrained sub-function $f_{x^t}(x_i)$
   (b) $x^{t+1} = g(y^{t+1})$
   (c) Increment t

The real computational work of this iterative procedure is in Steps 2(a) and 2(b), although we shall see later that such steps can, in some cases, be performed efficiently. With an appropriate feasibility function $g(y)$, one can guarantee that a fixed-point of this procedure yields a stationary point of the constrained function $f(x)$, by Claim 1 [2]. Further, any stationary point is trivially a fixed-point of this procedure (one can seed this procedure with such a point).

As we shall show in the next section, the EDML algorithm—which has been proposed for parameter estimation in Bayesian networks—is an instance of the above

[1]Under constraints, we consider points that are stationary with respect to the corresponding Lagrangian.

[2]We discuss this point further in the supplementary appendix.

procedure with some notable observations: (1) the sub-functions $f_{x^t}(x_i)$ are convex and have unique optima; (2) these sub-functions have an interesting semantics, as they correspond to posterior distributions that are induced by Naive Bayes networks with soft evidence asserted on them; (3) defining these sub-functions requires inference in a Bayesian network parameterized by the current feasible point x^t; (4) there are already several convergent, fixed-point iterative methods for finding the unique optimum of these sub-functions; and (5) these convergent methods produce solutions that are always feasible and, hence, the feasibility function $g(y)$ corresponds to the identity function $g(y) = y$ in this case.

We next show this connection to EDML as proposed for parameter estimation in Bayesian networks. We follow by deriving an EDML algorithm (another instance of the above procedure), but for parameter estimation in undirected graphical models. We will also study the impact of having complete data on both versions of the EDML algorithm, and finally evaluate the new instance of EDML by comparing it to conjugate gradient and L-BFGS when applied to complete datasets.

5.3 EDML for Bayesian Networks

Consider a (possibly incomplete) dataset $\mathcal{D}$ with examples $\mathbf{d}_1, \ldots, \mathbf{d}_N$, and a Bayesian network with parameters θ. Our goal is to find parameter estimates θ that minimize the negative log-likelihood:

$$f(\theta) = -\ell\ell(\theta|\mathcal{D}) = -\sum_{i=1}^{N} \log Pr_\theta(\mathbf{d}_i). \tag{5.1}$$

Here, $\theta = (\ldots, \theta_{X|\mathbf{u}}, \ldots)$ is a vector over the network parameters. Moreover, Pr_θ is the distribution induced by the Bayesian network structure under parameters θ. As such, $Pr_\theta(\mathbf{d}_i)$ is the probability of observing example $\mathbf{d}_i$ in dataset $\mathcal{D}$ under parameters θ.

Each component of θ is a parameter set $\theta_{X|\mathbf{u}}$, which defines a parameter $\theta_{x|\mathbf{u}}$ for each value x of variable X and instantiation $\mathbf{u}$ of its parents $\mathbf{U}$. The feasibility constraint here is that each component $\theta_{X|\mathbf{u}}$ satisfies the convex sum-to-one constraint: $\sum_x \theta_{x|\mathbf{u}} = 1$.

The above parameter estimation problem is clearly in the form of the constrained optimization problem that we phrased in the previous section and, hence, admits the same iterative procedure proposed in that section for finding stationary points. The relevant questions now are: What form do the sub-functions $f_{\theta^\star}(\theta_{X|\mathbf{u}})$ take in this context? What are their semantics? What properties do they have? How do we find their stationary points? What is the feasibility function $g(y)$ in this case? Finally, what is the connection to previous work on EDML? We address these questions next.

5.3.1 Form

We start by characterizing the sub-functions of the negative log-likelihood given in Equation 5.1.

Theorem 5 *For each parameter set $\theta_{X|\mathbf{u}}$, the negative log-likelihood of Equation 5.1 has the sub-function:*

$$f_{\theta^\star}(\theta_{X|\mathbf{u}}) = -\sum_{i=1}^{N} \log \left(C^i_{\mathbf{u}} + \sum_x C^i_{x|\mathbf{u}} \cdot \theta_{x|\mathbf{u}} \right) \tag{5.2}$$

where $C^i_{\mathbf{u}}$ and $C^i_{x|\mathbf{u}}$ are constants that are independent of parameter set $\theta_{X|\mathbf{u}}$, given by

$$C^i_{\mathbf{u}} = Pr_{\theta^\star}(\mathbf{d}_i) - Pr_{\theta^\star}(\mathbf{u}, \mathbf{d}_i) \qquad \text{and} \qquad C^i_{x|\mathbf{u}} = Pr_{\theta^\star}(x, \mathbf{u}, \mathbf{d}_i)/\theta^\star_{x|\mathbf{u}}$$

To compute the constants C^i, we *require inference* on a Bayesian network with parameters $\theta^\star$.[3]

5.3.2 Semantics

Equation 5.2 has an interesting semantics, as it corresponds to the negative log-likelihood of a root variable in a naive Bayes structure, on which soft, not necessarily hard, evi-

[3]Theorem 5 assumes tacitly that $\theta^\star_{x|\mathbf{u}} \neq 0$. More generally, however, $C^i_{x|\mathbf{u}} = \partial Pr_{\theta^\star}(\mathbf{d}_i)/\partial \theta_{x|\mathbf{u}}$, which can also be computed using some standard inference algorithms (Darwiche, 2003; Park & Darwiche, 2004).

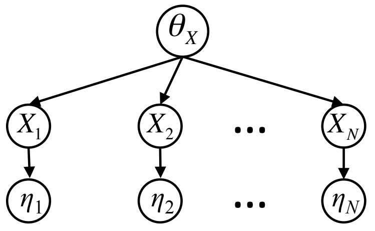


Figure 5.1: Estimation given independent soft observations.

dence is asserted (Choi et al., 2011).[4]

This model is illustrated in Figure 5.1, where our goal is to estimate a parameter set θ_X, given soft observations $\eta = (\eta_1, \ldots, \eta_N)$ on variables $X_1, \ldots, X_N$, where each η_i has a strength specified by a weight on each value x_i of X_i. If we denote the distribution of this model by $\mathbb{P}$, then (1) $\mathbb{P}(\theta)$ denotes a prior over parameters sets,[5] (2) $\mathbb{P}(x_i|\theta_X = (\ldots, \theta_x, \ldots)) = \theta_x$, and (3) weights $\mathbb{P}(\eta_i|x_i)$ denote the strengths of soft evidence η_i on value x_i. The log likelihood of our soft observations η is:

$$\log \mathbb{P}(\eta|\theta_X) = \sum_{i=1}^{N} \log \sum_{x_i} \mathbb{P}(\eta_i|x_i)\mathbb{P}(x_i|\theta_X) = \sum_{i=1}^{N} \log \sum_{x_i} \mathbb{P}(\eta_i|x_i) \cdot \theta_x \qquad (5.3)$$

The following result connects Equation 5.2 to the above likelihood of a soft dataset, when we now want to estimate the parameter set $\theta_{X|\mathbf{u}}$, for a particular variable X and parent instantiation $\mathbf{u}$.

Theorem 6 *Consider Equations 5.2 and 5.3, and assume that each soft evidence η_i has the strength $\mathbb{P}(\eta_i|x_i) = C^i_{\mathbf{u}} + C^i_{x|\mathbf{u}}$. It then follows that*

$$f_{\theta^\star}(\theta_{X|\mathbf{u}}) = -\log \mathbb{P}(\eta|\theta_{X|\mathbf{u}}) \qquad (5.4)$$

This theorem yields the following interesting semantics for EDML sub-functions. Consider a parameter set $\theta_{X|\mathbf{u}}$ and example $\mathbf{d}_i$ in our dataset. The example can then be viewed as providing "votes" on what this parameter set should be. In particular, the

[4] Soft evidence is an observation that increases or decreases ones belief in an event, but not necessarily to the point of certainty. For more on soft evidence, see (Chan & Darwiche, 2005).

[5] Typically, we assume Dirichlet priors for MAP estimation. However, we focus on ML estimation here.

vote of example $\mathbf{d}_i$ for value x takes the form of a soft evidence η_i whose strength is given by

$$\mathbb{P}(\eta_i|x_i) = Pr_{\theta^\star}(\mathbf{d}_i) - Pr_{\theta^\star}(\mathbf{u}, \mathbf{d}_i) + Pr_{\theta^\star}(x, \mathbf{u}, \mathbf{d}_i)/\theta^\star_{x|\mathbf{u}}$$

The sub-function is then aggregating these votes from different examples and producing a corresponding objective function on parameter set $\theta_{X|\mathbf{u}}$. EDML optimizes this objective function to produce the next estimate for each parameter set $\theta_{X|\mathbf{u}}$.

5.3.3 Properties

Equation 5.2 is a convex function, and thus has a unique optimum.[6] In particular, we have logs of a linear function, which are each concave. The sum of two concave functions is also concave, thus our sub-function $f_{\theta^\star}(\theta_{X|\mathbf{u}})$ is convex, and is subject to a convex sum-to-one constraint (Refaat et al., 2012). Convex functions are relatively well understood, and there are a variety of methods and systems that can be used to optimize Equation 5.2; see, e.g., (Boyd & Vandenberghe, 2004). We describe one such approach, next.

5.3.4 Finding the Unique Optima

In every EDML iteration, and for each parameter set $\theta_{X|\mathbf{u}}$, we seek the unique optimum for each sub-function $f_{\theta^\star}(\theta_{X|\mathbf{u}})$, given by Equation 5.2. Refaat, et al., has previously proposed a fixed-point algorithm that monotonically improves the objective, and is guaranteed to converge (Refaat et al., 2012). Moreover, the solutions it produces already satisfy the convex sum-to-one constraint and, hence, the feasibility function g ends up being the identity function $g(\theta) = \theta$.

In particular, we start with some initial feasible estimates $\theta^t_{X|\mathbf{u}}$ at iteration $t = 0$,

[6]More specifically, strict convexity implies a unique optimum, although under certain assumptions, we can guarantee that Equation 5.2 is indeed strictly convex.

and then apply the following update equation until convergence:

$$\theta^{t+1}_{x|\mathbf{u}} = \frac{1}{N}\sum_{i=1}^{N}\frac{(C^i_{\mathbf{u}} + C^i_{x|\mathbf{u}})\cdot\theta^t_{x|\mathbf{u}}}{C^i_{\mathbf{u}} + \sum_{x'} C^i_{x'|\mathbf{u}}\cdot\theta^t_{x'|\mathbf{u}}} \tag{5.5}$$

Note here that constants C^i are *computed by inference* on a Bayesian network structure under parameters θ^t (see Theorem 5 for the definitions of these constants). Moreover, while the above procedure is convergent when optimizing sub-functions $f_{\theta^\star}(\theta_{X|\mathbf{u}})$, the global EDML algorithm that is optimizing function $f(\theta)$ may not be convergent in general.

5.3.5 Connection to Previous Work

EDML was originally derived by applying an approximate inference algorithm to a meta-network, which is typically used in Bayesian approaches to parameter estimation (Choi et al., 2011; Refaat et al., 2012). This previous formulation of EDML, which is specific to Bayesian networks, now falls as a special instance of the one given in Section 5.2. In particular, the "sub-problems" defined by the original EDML (Choi et al., 2011; Refaat et al., 2012) correspond precisely to the sub-functions $f_{\theta^\star}(\theta_{X|\mathbf{u}})$ described here. Further, both versions of EDML are procedurally identical when they both use the same method for optimizing these sub-functions.

The new formulation of EDML is more transparent, however, at least in revealing certain properties of the algorithm. For example, it now follows immediately (from Section 5.2) that the fixed points of EDML are stationary points of the log-likelihood—a fact that was not proven until (Refaat et al., 2012), using a technique that appealed to the relationship between EDML and EM. Moreover, the proof that EDML under complete data will converge immediately to the optimal estimates is also now immediate (see Section 5.5). More importantly though, this new formulation provides a systematic procedure for deriving new instances of EDML for additional models, beyond Bayesian networks. Indeed, in the next section, we use this procedure to derive an EDML instance for Markov networks, which is followed by an empirical evaluation of the new algorithm under complete data.

5.4 EDML for Undirected Models

In this section, we show how parameter estimation for undirected graphical models, such as Markov networks, can also be posed as an optimization problem, as described in Section 5.2.

For Markov networks, $\theta = (\ldots, \theta_{\mathbf{X}_a}, \ldots)$ is a vector over the network parameters. Component $\theta_{\mathbf{X}_a}$ is a parameter set for a (tabular) factor a, assigning a number $\theta_{\mathbf{x}_a} \geq 0$ for each instantiation $\mathbf{x}_a$ of variables $\mathbf{X}_a$. The negative log-likelihood $-\ell\ell(\theta|\mathcal{D})$ for a Markov network is:

$$-\ell\ell(\theta|\mathcal{D}) = N \log Z_\theta - \sum_{i=1}^{N} \log Z_\theta(\mathbf{d}_i) \tag{5.6}$$

where Z_θ is the partition function, and where $Z_\theta(\mathbf{d}_i)$ is the partition function after conditioning on example $\mathbf{d}_i$, under parameterization θ. Sub-functions with respect to Equation 5.6 may not be convex, as was the case in Bayesian networks. Consider instead the following objective function, which we shall subsequently relate to the negative log-likelihood:

$$f(\theta) = -\sum_{i=1}^{N} \log Z_\theta(\mathbf{d}_i), \tag{5.7}$$

with a feasibility constraint that the partition function Z_θ equals some constant α. The following result tells us that it suffices to optimize Equation 5.7 under the given constraint, to optimize Equation 5.6.

Theorem 7 *Let α be a positive constant, and let $g(\theta)$ be a (feasibility) function satisfying $Z_{g(\theta)} = \alpha$ and $g(\theta_{\mathbf{x}_a}) \propto \theta_{\mathbf{x}_a}$ for all $\theta_{\mathbf{x}_a}$.*[7] *For every point θ, if $g(\theta)$ is optimal for Equation 5.7, subject to its constraint, then it is also optimal for Equation 5.6.*

[7] *Here, $g(\theta_{\mathbf{x}_a})$ denotes the component of $g(\theta)$ corresponding to $\theta_{\mathbf{x}_a}$. Moreover, the function $g(\theta)$ can be constructed, e.g., by simply multiplying all entries of one parameter set by α/Z_θ. In our experiments, we normalize each parameter set to sum-to-one, but then update the constant $\alpha = Z_{\theta^t}$ for the subsequent iteration.*

Moreover, a point θ is stationary for Equation 5.6 iff the point $g(\theta)$ is stationary for Equation 5.7, subject to its constraint.

With Equation 5.7 as a new (constrained) objective function for estimating the parameters of a Markov network, we can now cast it in the terms of Section 5.2. We start by characterizing its sub-functions.

Theorem 8 *For a given parameter set $\theta_{\mathbf{X}_a}$, the objective function of Equation 5.7 has sub-functions:*

$$f_{\theta^\star}(\theta_{\mathbf{X}_a}) = -\sum_{i=1}^{N} \log \sum_{\mathbf{x}_a} C^i_{\mathbf{x}_a} \cdot \theta_{\mathbf{x}_a} \qquad \textit{subject to} \qquad \sum_{\mathbf{x}_a} C_{\mathbf{x}_a} \cdot \theta_{\mathbf{x}_a} = \alpha \tag{5.8}$$

where $C^i_{\mathbf{x}_a}$ and $C_{\mathbf{x}_a}$ are constants that are independent of the parameter set $\theta_{\mathbf{X}_a}$:

$$C^i_{\mathbf{x}_a} = Z_{\theta^\star}(\mathbf{x}_a, \mathbf{d}_i)/\theta^\star_{\mathbf{x}_a} \qquad \textit{and} \qquad C_{\mathbf{x}_a} = Z_{\theta^\star}(\mathbf{x}_a)/\theta^\star_{\mathbf{x}_a}.$$

Note that computing these constants *requires inference* on a Markov network with parameters $\theta^\star$.[8]

Interestingly, this sub-function is convex, as well as the constraint (which is now linear), resulting in a unique optimum, as in Bayesian networks. However, even when $\theta^\star$ is a feasible point, the unique optima of these sub-functions may not be feasible when combined. Thus, the feasibility function $g(\theta)$ of Theorem 7 must be utilized in this case.

We now have another instance of the iterative algorithm proposed in Section 5.2, but for undirected graphical models. That is, *we have just derived an EDML algorithm for such models.*

[8]Theorem 8 assumes that $\theta^\star_{\mathbf{x}_a} \neq 0$. In general, $C^i_{\mathbf{x}_a} = \frac{\partial Z_{\theta^\star}(\mathbf{d}_i)}{\partial \theta_{\mathbf{x}_a}}$, and $C_{\mathbf{x}_a} = \frac{\partial Z_{\theta^\star}}{\partial \theta_{\mathbf{x}_a}}$. See also Footnote 3.

5.5 EDML under Complete Data

We consider now how EDML simplifies under complete data for both Bayesian and Markov networks, identifying forms and properties of the corresponding sub-functions under complete data.

We start with Bayesian networks. Consider a variable X, and a parent instantiation $\mathbf{u}$; and let $\mathcal{D}\#(x\mathbf{u})$ represent the number of examples that contain $x\mathbf{u}$ in the complete dataset $\mathcal{D}$. Equation 5.2 of Theorem 5 then reduces to:

$$f_{\theta^\star}(\theta_{X|\mathbf{u}}) = -\sum_x \mathcal{D}\#(x\mathbf{u}) \log \theta_{x|\mathbf{u}} + C,$$

where C is a constant that is independent of parameter set $\theta_{X|\mathbf{u}}$. Assuming that $\theta^\star$ is feasible (i.e., each $\theta_{X|\mathbf{u}}$ satisfies the sum-to-one constraint), the unique optimum of this sub-function is $\theta_{x|\mathbf{u}} = \frac{\mathcal{D}\#(x\mathbf{u})}{\mathcal{D}\#(\mathbf{u})}$, which is guaranteed to yield a feasible point θ, globally. Hence, EDML produces the unique optimal estimates in its first iteration and terminates immediately thereafter.

The situation is different, however, for Markov networks. Under a complete dataset $\mathcal{D}$, Equation 5.8 of Theorem 8 reduces to:

$$f_{\theta^\star}(\theta_{\mathbf{X}_a}) = -\sum_{\mathbf{x}_a} \mathcal{D}\#(\mathbf{x}_a) \log \theta_{\mathbf{x}_a} + C,$$

where C is a constant that is independent of parameter set $\theta_{\mathbf{X}_a}$. Assuming that $\theta^\star$ is feasible (i.e., satisfies $Z_{\theta^\star} = \alpha$), the unique optimum of this sub-function has the closed form: $\theta_{\mathbf{x}_a} = \frac{\alpha}{N} \frac{\mathcal{D}\#(\mathbf{x}_a)}{C_{\mathbf{x}_a}}$, which is equivalent to the unique optimum one would obtain in a sub-function for Equation 5.6 (Pietra, Pietra, & Lafferty, 1997; Murphy, 2012). Contrary to Bayesian networks, the collection of these optima for different parameter sets do not necessarily yield a feasible point θ. Hence, the feasibility function g of Theorem 7 must be applied here. The resulting feasible point, however, may no longer be a stationary point for the corresponding sub-functions, leading EDML to iterate further. Hence, under complete data, EDML for Bayesian networks converges immediately, while EDML for Markov networks may take multiple iterations.

Both results are consistent with what is already known in the literature on parameter estimation for Bayesian and Markov networks. The result on Bayesian networks is useful in confirming that EDML performs optimally in this case. *The result for Markov networks, however, gives rise to a new algorithm for parameter estimation under complete data.* We evaluate the performance of this new EDML algorithm after considering the following example.

Let $\mathcal{D}$ be a complete dataset over three variables A, B and C, specified in terms of the number of times that each instantiation a, b, c appears in $\mathcal{D}$. In particular, we have the following counts:
$\mathcal{D}\#(a, b, c) = 4$, $\mathcal{D}\#(a, b, \bar{c}) = 18$, $\mathcal{D}\#(a, \bar{b}, c) = 2$, $\mathcal{D}\#(a, \bar{b}, \bar{c}) = 13$, $\mathcal{D}\#(\bar{a}, b, c) = 1$, $\mathcal{D}\#(\bar{a}, b, \bar{c}) = 1$, $\mathcal{D}\#(\bar{a}, \bar{b}, c) = 42$, and $\mathcal{D}\#(\bar{a}, \bar{b}, \bar{c}) = 19$.
Suppose we want to learn, from this dataset, a Markov network with 3 edges, (A, B), (B, C) and (A, C), with the corresponding parameter sets θ_{AB}, θ_{BC} and θ_{AC}. If the initial set of parameters $\theta^\star = (\theta^\star_{AB}, \theta^\star_{BC}, \theta^\star_{AC})$ is uniform, i.e., $\theta^\star_{XY} = (1, 1, 1, 1)$, then Equation 5.8 gives the sub-function $f_{\theta^\star}(\theta_{AB}) = -22 \cdot \log \theta_{ab} - 15 \cdot \log \theta_{a\bar{b}} - 2 \cdot \log \theta_{\bar{a}b} - 61 \cdot \log \theta_{\bar{a}\bar{b}}$. Moreover, we have $Z_{\theta^\star} = 2 \cdot \theta_{ab} + 2 \cdot \theta_{a\bar{b}} + 2 \cdot \theta_{\bar{a}b} + 2 \cdot \theta_{\bar{a}\bar{b}}$. Minimizing $f_{\theta^\star}(\theta_{AB})$ under $Z_{\theta^\star} = \alpha = 2$ corresponds to solving a convex optimization problem, which has the unique solution: $(\theta_{ab}, \theta_{a\bar{b}}, \theta_{\bar{a}b}, \theta_{a\bar{b}}) = (\frac{22}{100}, \frac{15}{100}, \frac{2}{100}, \frac{61}{100})$. We solve similar convex optimization problems for the other parameter sets θ_{BC} and θ_{AC}, to update estimates $\theta^\star$. We then apply an appropriate feasibility function g (see Footnote 7), and repeat until convergence.

5.6 Experimental Results

We evaluate now the efficiency of EDML, conjugate gradient (CG) and Limited-memory BFGS (L-BFGS), when learning Markov networks under complete data.[9] We first learned grid-structured pairwise MRFs from the CEDAR dataset of handwritten digits,

[9] We also considered Iterative Proportional Fitting (IPF) as a baseline. However, IPF does not scale to our benchmarks, as it invokes inference many times more often than the methods we considered.

which has 10 datasets (one for each digit) of 16×16 binary images. We also simulated datasets from networks used in the probabilistic inference evaluations of UAI-2008, 2010 and 2012, that are amenable to jointree inference.[10] For each network, we simulated 3 datasets of size 2^{10} examples each, and learned parameters using the original structure. Experiments were run on a 3.6GHz Intel i5 CPU with access to 8GB RAM.

We used the CG implementation in the Apache Commons Math library, and the L-BFGS implementation in Mallet.[11] Both are Java libraries, and our implementation of EDML is also in Java. More importantly, all of the CG, L-BFGS, and EDML methods rely on the same underlying engine for exact inference.[12] For EDML, we damped parameter estimates at each iteration, which is typical for algorithms like loopy belief propagation, which EDML was originally inspired by (Choi et al., 2011).[13] We used Brent's method with default settings for line search in CG, which was the most efficient over all univariate solvers in Apache's library, which we evaluated in initial experiments.

We first run CG until convergence (or after exceeding 30 minutes) to obtain parameter estimates of some quality q_{cg} (in log likelihood), recording the number of iterations i_{cg} and time t_{cg} required in minutes. EDML is then run next until it obtains an estimate of the same quality q_{cg}, or better, recording also the number of iterations i_{edml} and time t_{edml} in minutes. The time speed-up S of EDML over CG is computed as $t_{\text{cg}}/t_{\text{edml}}$. We also performed the same comparison with L-BFGS instead of CG, recording the corresponding number of iterations ($i_{\text{l-bfgs}}$, i'_{edml}) and time taken ($t_{\text{l-bfgs}}$, t'_{edml}), giving us the speed-up of EDML over L-BFGS as $S' = t_{\text{l-bfgs}}/t'_{\text{edml}}$.

Table 5.1 shows results for both sets of experiments. It shows the number of vari-

[10]Network 54.wcsp is a weighted CSP problem; or-chain-$\{42, 45, 147, 148, 225\}$ are from the Promedas suite; rbm-20 is a restricted Boltzmann machine; Seg2-17, Seg7-11 are from the Segmentation suite; family2-dominant, family2-recessive are genetic linkage analysis networks; and grid10x10.f5.wrap, grid10x10.10.wrap are 10x10 grid networks.

[11]Available at `http://commons.apache.org/` and `http://mallet.cs.umass.edu/`.

[12]For exact inference in Markov networks, we employed a jointree algorithm from the SAMIAM inference library, `http://reasoning.cs.ucla.edu/samiam/`.

[13]We start with an initial factor of $\frac{1}{2}$, which we tighten as we iterate.

ables in each network (#vars), the average number of iterations taken by each algorithm, and the average speed-up achieved by EDML over CG (L-BFGS).[14] On the given benchmarks, we see that on average EDML was roughly $13.5\times$ faster than CG, and $4.5\times$ faster than L-BFGS. EDML was up to an order-of-magnitude faster than L-BFGS in some cases. In many cases, EDML required more iterations but was still faster in time. This is due in part by the number of times inference is invoked by CG and L-BFGS (in line search), whereas EDML only needs to invoke inference once per iteration.

5.7 Related Work

As an iterative fixed-point algorithm, we can view EDML as a Jacobi-type method, where updates are performed in parallel (Bertsekas & Tsitsiklis, 1989). Alternatively, a version of EDML using Gauss-Seidel iterations would update each parameter set in sequence using the most recently computed update. This leads to an algorithm that monotonically improves the log likelihood at each update. In this case, we obtain a coordinate descent algorithm, Iterative Proportional Fitting (IPF) (Jirousek & Preucil, 1995), as a special case of EDML.

The notion of fixing all parameters, except for one, has been exploited before for the purposes of optimizing the log likelihood of a Markov network, as a heuristic for structure learning (Pietra et al., 1997). This notion also underlies the IPF algorithm; see, e.g., (Murphy, 2012), Section 19.5.7. In the case of complete data, the resulting sub-function is convex, yet for incomplete data, it is not necessarily convex.

Optimization methods such as conjugate gradient, and L-BFGS (Liu & Nocedal, 1989), are more commonly used to optimize the parameters of a Markov network. For relational Markov networks or Markov networks that otherwise assume a feature-

[14]For CG, we used a threshold based on relative change in the likelihood at 10^{-4}. We used Mallet's default convergence threshold for L-BFGS.

based representation (Domingos & Lowd, 2009), evaluating the likelihood is typically intractable, in which case one typically optimizes instead the pseudo-log-likelihood (Besag, 1975). For more on parameter estimation in Markov networks, see (Koller & Friedman, 2009; Murphy, 2012).

5.8 Conclusion

In this chapter, we provided an abstract and simple view of the EDML algorithm, originally proposed for parameter estimation in Bayesian networks, as a particular method for continuous optimization. One consequence of this view is that it is immediate that fixed-points of EDML are stationary points of the log-likelihood, and vice-versa (Refaat et al., 2012). A more interesting consequence, is that it allows us to propose an EDML algorithm for a new class of models, Markov networks. Empirically, we find that EDML can more efficiently learn parameter estimates for Markov networks under complete data, compared to conjugate gradient and L-BFGS, sometimes by an order-of-magnitude.

Table 5.1: Speed-up results of EDML over CG and L-BFGS

problem	#vars	i_{cg}	i_{edml}	t_{cg}	(S)	$i_{\text{l-bfgs}}$	i'_{edml}	$t_{\text{l-bfgs}}$	(S')
zero	256	45	105	3.6	3.9x	24	74	1.6	2x
one	256	104	73	8.3	13.3x	58	42	3.9	8.1x
two	256	46	154	3.7	2.8x	21	87	1.5	1.5x
three	256	43	169	3.6	2.5x	52	169	3.6	1.9x
four	256	56	126	4.6	4.3x	61	115	3.9	3.2x
five	256	43	155	3.5	2.7x	49	155	3.2	1.9x
six	256	48	150	3.9	3.1x	20	90	1.5	1.4x
seven	256	57	147	4.6	3.4x	23	89	1.7	1.6x
eight	256	48	155	3.8	2.8x	57	154	3.8	2.3x
nine	256	56	168	4.5	3.15x	45	141	2.9	1.9x
54.wcsp	67	107.3	160.3	6.6	2.8x	68.3	172	1.8	0.7x
or-chain-42	385	120.3	27	0.1	31.3x	110	54.3	0.1	6.4x
or-chain-45	715	151	33.7	0.1	12.5x	94.3	36.3	0.1	4.9x
or-chain-147	410	107.7	18.7	3.3	80.7x	105	58.3	1.6	12.8x
or-chain-148	463	122.7	42.3	1	49.0x	80	32	0.3	14.2x
or-chain-225	467	181.3	58	0.8	44.1x	137.7	69	0.3	10.8x
rbm20	40	9	41	31	2.4x	30	107.2	30.2	1x
Seg2-17	228	63	83.7	1.8	7x	46.7	64.7	0.7	4.1x
Seg7-11	235	54.3	84	1.9	2.8x	48.7	73.3	1.3	2.3x
Family2Dominant	385	117.3	88	2.4	5.9x	85.7	78.3	1	2.7x
Family2Recessive	385	111.6	89.7	1.31	3.9x	86.3	81.7	0.7	2.2x
grid10x10.f5.wrap	100	136.7	239	17.4	6.3x	142	180.3	10.3	4.6x
grid10x10.f10.wrap	100	101.3	62.3	12.4	20.9x	92.7	59	5.9	9.7x
average	**275.7**	**83.9**	**101.3**	**5.4**	**13.6x**	**66.8**	**94.9**	**3.6**	**4.5x**

CHAPTER 6

Decomposing Parameter Estimation Problems in Bayesian Networks

We propose in this chapter a technique for decomposing the parameter learning problem in Bayesian networks into independent learning problems. Our technique applies to incomplete datasets and exploits variables that are either hidden or observed in the given dataset. We show empirically that the proposed technique can lead to orders-of-magnitude savings in learning time. We explain, analytically and empirically, the reasons behind our reported savings, and compare the proposed technique to related ones that are sometimes used by inference algorithms. This chapter is based on (Refaat et al., 2014).

6.1 Introduction

When learning the parameters of a graphical model from data, a key distinction is commonly drawn between complete and incomplete datasets. In a complete dataset, the value of each variable is known in every example. In this case, maximum likelihood parameters are unique and can be easily estimated using a single pass on the dataset. However, when the data is incomplete, the optimization problem is generally non-convex, has multiple local optima, and is commonly solved by iterative methods, such as EM (Dempster et al., 1977; Lauritzen, 1995), gradient descent (Russel, Binder, Koller, & Kanazawa, 1995) and, more recently, EDML (Choi et al., 2011; Refaat et al., 2012, 2013).

Incomplete datasets may still exhibit a certain structure. In particular, certain variables may always be observed in the dataset, while others may always be unobserved (hidden). We exploit this structure by decomposing the parameter learning problem into smaller learning problems that can be solved independently. In particular, we show that the stationary points of the likelihood function can be characterized by the ones of the smaller problems. This implies that algorithms such as EM and gradient descent can be applied to the smaller problems while preserving their guarantees. Empirically, we show that the proposed decomposition technique can lead to orders-of-magnitude savings. Moreover, we show that the savings are amplified when the dataset grows in size. Finally, we explain these significant savings analytically by examining the impact of our decomposition technique on the dynamics of the used convergence test, and on the properties of the datasets associated with the smaller learning problems.

The chapter is organized as follows. In Section 6.2, we present the decomposition technique and then prove its soundness in Section 6.3. Section 6.4 is dedicated to empirical results and to analyzing the reported savings. We discuss related work in Section 6.5 and finally close with some concluding remarks in Section 6.6. The proofs are moved to the appendix in the supplementary material.

6.2 Decomposing the Learning Problem

We now show how the problem of learning Bayesian network parameters can be decomposed into independent learning problems. The proposed technique exploits two aspects of a dataset: hidden and observed variables.

Proposition 3 *The likelihood function $L(\theta|\mathcal{D})$ does not depend on the parameters of variable X if X is hidden in dataset $\mathcal{D}$ and is a leaf of the network structure.*

If a hidden variable appears as a leaf in the network structure, it can be removed from the structure while setting its parameters arbitrarily (assuming no prior). This process

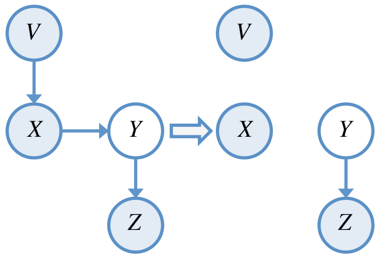


Figure 6.1: Identifying components of network G given $\mathbf{O} = \{V, X, Z\}$.

can be repeated until there are no leaf variables that are also hidden. The soundness of this technique follows from (Shachter, 1986, 1990).

Our second decomposition technique will exploit the observed variables of a dataset. In a nutshell, we will (a) decompose the Bayesian network into a number of sub-networks, (b) learn the parameters of each sub-network independently, and then (c) assemble parameter estimates for the original network from the estimates obtained in each sub-network.

Definition 1 (Component) *Let G be a network, $\mathbf{O}$ be some observed variables in G and let $G|\mathbf{O}$ be the network which results from deleting all edges from G which are outgoing from $\mathbf{O}$. A* <u>*component*</u> *of $G|\mathbf{O}$ is a maximal set of nodes that are connected in $G|\mathbf{O}$.*

Consider the network G in Figure 6.1, with observed variables $\mathbf{O} = \{V, X, Z\}$. Then $G|\mathbf{O}$ has three components in this case: $\mathbf{S}_1 = \{V\}$, $\mathbf{S}_2 = \{X\}$, and $\mathbf{S}_3 = \{Y, Z\}$.

The components of a network partition its parameters into groups, one group per component. In the above example, the network parameters are partitioned into the following groups:

$$\begin{aligned}
\mathbf{S}_1 : &\quad \{\theta_v, \theta_{\overline{v}}\} \\
\mathbf{S}_2 : &\quad \{\theta_{x|v}, \theta_{\overline{x}|v}, \theta_{x|\overline{v}}, \theta_{\overline{x}|\overline{v}}\} \\
\mathbf{S}_3 : &\quad \{\theta_{y|x}, \theta_{\overline{y}|x}, \theta_{y|\overline{x}}, \theta_{\overline{y}|\overline{x}}, \theta_{z|y}, \theta_{\overline{z}|y}, \theta_{z|\overline{y}}, \theta_{\overline{z}|\overline{y}}\}.
\end{aligned}$$

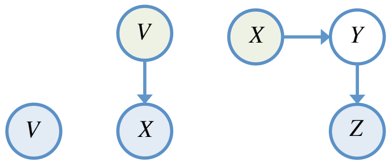


Figure 6.2: The sub-networks induced by adding boundary variables to components.

We will later show that the learning problem can be decomposed into independent learning problems, each induced by one component. To define these independent problems, we need some definitions.

Definition 2 (Boundary Node) *Let* $\mathbf{S}$ *be a component of* $G|\mathbf{O}$. *If edge* $B \to S$ *appears in* G, $B \notin \mathbf{S}$ *and* $S \in \mathbf{S}$, *then* B *is called a* <u>*boundary*</u> *for component* $\mathbf{S}$.

Considering Figure 6.1, node X is the only boundary for component $\mathbf{S}_3 = \{Y, Z\}$. Moreover, node V is the only boundary for component $\mathbf{S}_2 = \{X\}$. Component $\mathbf{S}_1 = \{V\}$ has no boundary nodes.

The independent learning problems are based on the following sub-networks.

Definition 3 (Sub-Network) *Let* $\mathbf{S}$ *be a component of* $G|\mathbf{O}$ *with boundary variables* $\mathbf{B}$. *The* <u>*sub-network*</u> *of component* $\mathbf{S}$ *is the subset of network* G *induced by variables* $\mathbf{S} \cup \mathbf{B}$.

Figure 6.2 depicts the three sub-networks which correspond to our running example.

The parameters of a sub-network will be learned using projected datasets.

Definition 4 *Let* $\mathcal{D} = \mathbf{d}_1, \ldots, \mathbf{d}_N$ *be a dataset over variables* $\mathbf{X}$ *and let* $\mathbf{Y}$ *be a subset of variables* $\mathbf{X}$. *The* <u>*projection*</u> *of dataset* $\mathcal{D}$ *on variables* $\mathbf{Y}$ *is the set of examples* $\mathbf{e}_1, \ldots, \mathbf{e}_N$, *where each* $\mathbf{e}_i$ *is the subset of example* $\mathbf{d}_i$ *which pertains to variables* $\mathbf{Y}$.

We show below a dataset for the full Bayesian network in Figure 6.1, followed by three projected datasets, one for each of the sub-networks in Figure 6.2.

	V	X	Y	Z
$\mathbf{d}_1$	$\overline{v}$	x	?	$\overline{z}$
$\mathbf{d}_2$	v	$\overline{x}$	?	z
$\mathbf{d}_3$	v	x	?	$\overline{z}$

	V	*count*
$\mathbf{e}_1$	$\overline{v}$	1
$\mathbf{e}_2$	v	2

	V	X	*count*
$\mathbf{e}_1$	$\overline{v}$	x	1
$\mathbf{e}_2$	v	$\overline{x}$	1
$\mathbf{e}_3$	v	x	1

	X	Y	Z	*count*
$\mathbf{e}_1$	x	?	$\overline{z}$	2
$\mathbf{e}_2$	$\overline{x}$	?	z	1

The projected datasets are "compressed" as we only represent unique examples, together with a count of how many times each example appears in a dataset. Using compressed datasets is crucial to realizing the full potential of decomposition, as it ensures that the size of a projected dataset is at most exponential in the number of variables appearing in its sub-network (more on this later).

We are now ready to describe our decomposition technique. Given a Bayesian network structure G and a dataset $\mathcal{D}$ that observes variables $\mathbf{O}$, we can get the stationary points of the likelihood function for network G as follows:

1. Identify the components $\mathbf{S}_1, \ldots, \mathbf{S}_M$ of $G|\mathbf{O}$ (Definition 1).

2. Construct a sub-network for each component $\mathbf{S}_i$ and its boundary variables $\mathbf{B}_i$ (Definition 3).

3. Project the dataset $\mathcal{D}$ on the variables of each sub-network (Definition 4).

4. Identify a stationary point for each sub-network and its projected dataset (using, e.g., EM, EDML or gradient descent).

5. Recover the learned parameters *of non-boundary variables* from each sub-network.

We will next prove that (a) these parameters are a stationary point of the likelihood function for network G, and (b) every stationary point of the likelihood function can be generated this way (using an appropriate seed).

6.3 Soundness

The soundness of our decomposition technique is based on three steps. We first introduce the notion of a *parameter term,* on which our proof rests. We then show how the likelihood function for the Bayesian network can be decomposed into component likelihood functions, one for each sub-network. We finally show that the stationary points of the likelihood function (network) can be characterized by the stationary points of component likelihood functions (sub-networks).

Two parameters are *compatible* iff they agree on the state of their common variables. For example, parameters $\theta_{z|y}$ and $\theta_{y|x}$ are compatible, but parameters $\theta_{z|\overline{y}}$ and $\theta_{y|x}$ are not compatible, as $y \neq \overline{y}$. Moreover, a parameter is compatible with an example iff they agree on the state of their common variables. Parameter $\theta_{\overline{y}|x}$ is compatible with example $x, \overline{y}, z$, but not with example x, y, z.

Definition 5 (Parameter Term) *Let* $\mathbf{S}$ *be network variables and let* $\mathbf{d}$ *be an example. A* $\underline{\textit{parameter term}}$ *for* $\mathbf{S}$ *and* $\mathbf{d}$*, denoted* $\Theta_{\mathbf{S}}^{\mathbf{d}}$*, is a product of compatible network parameters, one for each variable in* $\mathbf{S}$*, that are also compatible with example* $\mathbf{d}$.

Consider the network $X \to Y \to Z$. If $\mathbf{S} = \{Y, Z\}$ and $\mathbf{d} = x, z$, then $\Theta_{\mathbf{S}}^{\mathbf{d}}$ will denote either $\theta_{y|x}\theta_{z|y}$ or $\theta_{\overline{y}|x}\theta_{z|\overline{y}}$. Moreover, if $\mathbf{S} = \{X, Y, Z\}$, then $\Theta_{\mathbf{S}}^{\mathbf{d}}$ will denote either $\theta_x\theta_{y|x}\theta_{z|y}$ or $\theta_x\theta_{\overline{y}|x}\theta_{z|\overline{y}}$. In this case, $Pr(\mathbf{d}) = \sum_{\Theta_{\mathbf{S}}^{\mathbf{d}}} \Theta_{\mathbf{S}}^{\mathbf{d}}$. This holds more generally, whenever $\mathbf{S}$ is the set of all network variables.

We will now use parameter terms to show how the likelihood function can be decomposed into component likelihood functions.

Theorem 9 *Let* $\mathbf{S}$ *be a component of* $G|\mathbf{O}$ *and let* $\mathbf{R}$ *be the remaining variables of network* G*. If variables* $\mathbf{O}$ *are observed in example* $\mathbf{d}$*, we have*

$$Pr_\theta(\mathbf{d}) = \left[\sum_{\Theta^{\mathbf{d}}_{\mathbf{S}}} \Theta^{\mathbf{d}}_{\mathbf{S}}\right] \left[\sum_{\Theta^{\mathbf{d}}_{\mathbf{R}}} \Theta^{\mathbf{d}}_{\mathbf{R}}\right].$$

If θ denotes all network parameters, and $\mathbf{S}$ is a set of network variables, then $\theta\!:\!\mathbf{S}$ will denote the subset of network parameters which pertain to the variables in $\mathbf{S}$. Each component $\mathbf{S}$ of a Bayesian network induces its own likelihood function over parameters $\theta\!:\!\mathbf{S}$.

Definition 6 (Component Likelihood) *Let* $\mathbf{S}$ *be a component of* $G|\mathbf{O}$*. For dataset* $\mathcal{D} = \mathbf{d}_1, \ldots, \mathbf{d}_N$*, the* <u>*component likelihood*</u> *for* $\mathbf{S}$ *is defined as*

$$L(\theta\!:\!\mathbf{S}|\mathcal{D}) = \prod_{i=1}^{N} \sum_{\Theta^{\mathbf{d}_i}_{\mathbf{S}}} \Theta^{\mathbf{d}_i}_{\mathbf{S}}.$$

In our running example, the components are $\mathbf{S}_1 = \{V\}$, $\mathbf{S}_2 = \{X\}$ and $\mathbf{S}_3 = \{Y, Z\}$. Moreover, the observed variables are $\mathbf{O} = \{V, X, Z\}$. Hence, the component likelihoods are

$$\begin{aligned}
L(\theta\!:\!\mathbf{S}_1|\mathcal{D}) &= [\theta_{\overline{v}}]\,[\theta_v]\,[\theta_v] \\
L(\theta\!:\!\mathbf{S}_2|\mathcal{D}) &= \left[\theta_{x|\overline{v}}\right]\left[\theta_{\overline{x}|v}\right]\left[\theta_{x|v}\right] \\
L(\theta\!:\!\mathbf{S}_3|\mathcal{D}) &= \left[\theta_{y|x}\theta_{\overline{z}|y} + \theta_{\overline{y}|x}\theta_{\overline{z}|\overline{y}}\right]\left[\theta_{y|\overline{x}}\theta_{z|y} + \theta_{\overline{y}|\overline{x}}\theta_{z|\overline{y}}\right]\left[\theta_{y|x}\theta_{\overline{z}|y} + \theta_{\overline{y}|x}\theta_{\overline{z}|\overline{y}}\right]
\end{aligned}$$

The parameters of component likelihoods partition the network parameters. That is, the parameters of two component likelihoods are always non-overlapping. Moreover, the parameters of component likelihoods account for all network parameters.[1]

We can now state our main decomposition result, which is a direct corollary of Theorem 9.

[1]The sum-to-one constraints that underlie each component likelihood also partition the sum-to-one constraints of the likelihood function.

Corollary 1 *Let* $\mathbf{S}_1, \ldots, \mathbf{S}_M$ *be the components of* $G|\mathbf{O}$*. If variables* $\mathbf{O}$ *are observed in dataset* $\mathcal{D}$*,*

$$L(\theta|\mathcal{D}) = \prod_{i=1}^{M} L(\theta : \mathbf{S}_i|\mathcal{D}).$$

Hence, the network likelihood decomposes into a product of component likelihoods. This leads to another important corollary (see Lemma 1 in the Appendix):

Corollary 2 *Let* $\mathbf{S}_1, \ldots, \mathbf{S}_M$ *be the components of* $G|\mathbf{O}$*. If variables* $\mathbf{O}$ *are observed in dataset* $\mathcal{D}$*, then* $\theta^\star$ *is a stationary point of the likelihood* $L(\theta|\mathcal{D})$ *iff, for each* i*,* $\theta^\star : \mathbf{S}_i$ *is a stationary point for the component likelihood* $L(\theta : \mathbf{S}_i|\mathcal{D})$*.*

The search for stationary points of the network likelihood is now decomposed into independent searches for stationary points of component likelihoods.

We will now show that the stationary points of a component likelihood can be identified using any algorithm that identifies such points for the network likelihood.

Theorem 10 *Consider a sub-network* G *which is induced by component* $\mathbf{S}$ *and boundary variables* $\mathbf{B}$*. Let* θ *be the parameters of sub-network* G*, and let* $\mathcal{D}$ *be a dataset for* G *that observes boundary variables* $\mathbf{B}$*. Then* $\theta^\star$ *is a stationary point for the sub-network likelihood,* $L(\theta|\mathcal{D})$*, only if* $\theta^\star : \mathbf{S}$ *is a stationary point for the component likelihood* $L(\theta : \mathbf{S}|\mathcal{D})$*. Moreover, every stationary point for* $L(\theta : \mathbf{S}|\mathcal{D})$ *is part of some stationary point for* $L(\theta|\mathcal{D})$*.*

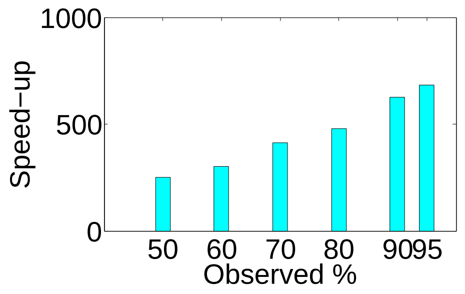
Given an algorithm that identifies stationary points of the likelihood function of Bayesian networks (e.g., EM), we can now identify all stationary points of a component likelihood. That is, we just apply this algorithm to the sub-network of each component $\mathbf{S}$, and then extract the parameter estimates of variables in $\mathbf{S}$ while *ignoring* the parameters of boundary variables. This proves the soundness of our proposed decomposition technique.

6.4 The Computational Benefit of Decomposition

We will now illustrate the computational benefits of the proposed decomposition technique, showing orders-of-magnitude reductions in learning time. Our experiments are structured as follows. Given a Bayesian network G, we generate a dataset $\mathcal{D}$ while ensuring that a certain percentage of variables are observed, with all others hidden. Using dataset $\mathcal{D}$, we estimate the parameters of network G using two methods. The first uses the classical EM on network G and dataset $\mathcal{D}$. The second decomposes network G into its sub-networks $G_1, \ldots, G_M$, projects the dataset $\mathcal{D}$ on each sub-network, and then applies EM to each sub-network and its projected dataset. This method is called D-EM (for Decomposed EM). We use the same seed for both EM and D-EM.

Before we present our results, we have the following observations on our data generation model. First, we made all unobserved variables hidden (as opposed to missing at random) as this leads to a more difficult learning problem, especially for EM (even with the pruning of hidden leaf nodes). Second, it is not uncommon to have a significant number of variables that are always observed in real-world datasets. For example, in the UCI repository: the internet advertisements dataset has 1558 variables, only 3 of which have missing values; the automobile dataset has 26 variables, where 7 have missing values; the dermatology dataset has 34 variables, where only age can be missing; and the mushroom dataset has 22 variables, where only one variable has missing values (Bache & Lichman, 2013).

We performed our experiments on three sets of networks: synthesized chains, synthesized complete binary trees, and some benchmarks from the UAI 2008 evaluation with other standard benchmarks (called UAI networks): alarm, win95pts, andes, diagnose, water, and pigs. Figure 6.3 and Table 6.1 depict the obtained time savings. As can be seen from these results, decomposing chains and trees lead to two orders-of-magnitude speed-ups for almost all observed percentages. For UAI networks, when observing 70% of the variables or more, one obtains one-to-two orders-of-magnitude



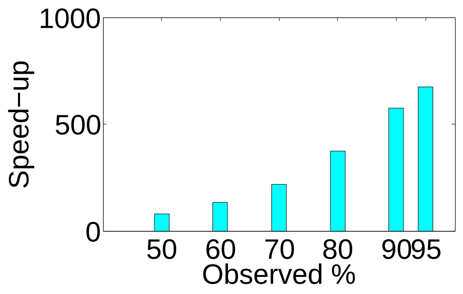


Figure 6.3: Speed-up of D-EM over EM on chain networks: three chains (180, 380, and 500 variables) (left), and tree networks (63, 127, 255, and 511 variables) (right), with three random datasets per network/observed percentage, and 2^{10} examples per dataset.

speed-ups. We note here that the time used for D-EM includes the time needed for decomposition (i.e., identifying the sub-networks and their projected datasets). Similar results for EDML are shown in the supplementary material.

The reported computational savings appear quite surprising. We now shed some light on the culprit behind these savings. We also argue that some of the most prominent tools for Bayesian networks do not appear to employ the proposed decomposition technique when learning network parameters.

Our first analytic explanation for the obtained savings is based on understanding the role of data projection, which can be illustrated by the following example. Consider a chain network over binary variables $X_1, \ldots, X_n$, where n is even. Consider also a dataset $\mathcal{D}$ in which variable X_i is observed for all odd i. There are $n/2$ sub-networks in this case. The first sub-network is X_1. The remaining sub-networks are in the form $X_{i-1} \rightarrow X_i \rightarrow X_{i+1}$ for $i = 2, 4, \ldots, n-2$ (node X_n will be pruned). The dataset $\mathcal{D}$ can have up to $2^{n/2}$ distinct examples. If one learns parameters without decomposition, one would need to call the inference engine once for each distinct example, in each iteration of the learning algorithm. With m iterations, the inference engine may be called up to $m2^{n/2}$ times. When learning with decomposition, however, each projected dataset will have at most 2 distinct examples for sub-network X_1, and at most 4 distinct

Observed %	Network	Speed-up D-EM	Network	Speed-up D-EM	Network	Speed-up D-EM
95.0%	alarm	267.67x	diagnose	43.03x	andes	155.54x
90.0%	alarm	173.47x	diagnose	17.16x	andes	52.63x
80.0%	alarm	115.4x	diagnose	11.86x	andes	14.27x
70.0%	alarm	87.67x	diagnose	3.25x	andes	2.96x
60.0%	alarm	92.65x	diagnose	3.48x	andes	0.77x
50.0%	alarm	12.09x	diagnose	3.73x	andes	1.01x
95.0%	win95pts	591.38x	water	811.48x	pigs	235.63x
90.0%	win95pts	112.57x	water	110.27x	pigs	37.61x
80.0%	win95pts	22.41x	water	7.23x	pigs	34.19x
70.0%	win95pts	17.92x	water	1.5x	pigs	16.23x
60.0%	win95pts	4.8x	water	2.03x	pigs	4.1x
50.0%	win95pts	7.99x	water	4.4x	pigs	3.16x

Table 6.1: Speed-up of D-EM over EM on UAI networks. Three random datasets per network/observed percentage with 2^{10} examples per dataset.

examples for sub-network $X_{i-1} \rightarrow X_i \rightarrow X_{i+1}$ (variable X_i is hidden, while variables X_{i-1} and X_{i+1} are observed). Hence, if sub-network i takes m_i iterations to converge, then the inference engine would need to be called at most $2m_1+4(m_2+m_4+\ldots+m_{n-2})$ times. We will later show that m_i is generally significantly smaller than m. Hence, with decomposed learning, the number of calls to the inference engine can be significantly smaller, which can contribute significantly to the obtained savings. [2]

Our analysis suggests that the savings obtained from decomposing the learning problem would amplify as the dataset gets larger. This can be seen clearly in Figure 6.4 (left), which shows that the speed-up of D-EM over EM grows linearly with

[2]The analysis in this section was restricted to chains to make the discussion concrete. This analysis, however, can be generalized to arbitrary networks if enough variables are observed in the corresponding dataset.

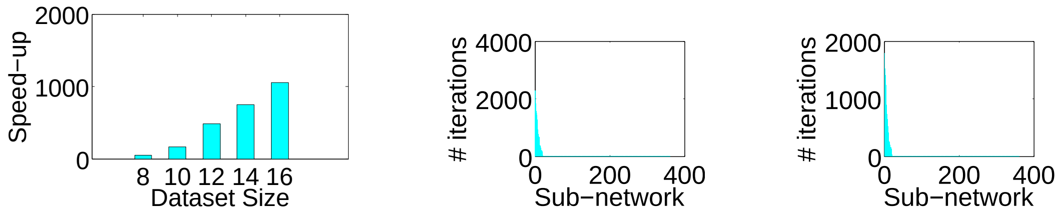


Figure 6.4: Left: Speed-up of D-EM over EM as a function of dataset size. This is for a chain network with 180 variables, while observing 50% of the variables. Right Pair: Graphs showing the number of iterations required by each sub-network, sorted descendingly. The problem is for learning Network Pigs while observing 90% of the variables, with convergence based on parameters (left), and on likelihood (right).

the dataset size. Hence, decomposition can be critical when learning with very large datasets.

Interestingly, two of the most prominent (non-commercial) tools for Bayesian networks *do not exhibit this behavior* on the chain network discussed above. This is shown in Figure 6.5, which compares D-EM to the EM implementations of the GENIE/SMILE and SAMIAM systems,[3] both of which were represented in previous inference evaluations (Darwiche, Dechter, Choi, Gogate, & Otten, 2008). In particular, we ran these systems on a chain network $X_0 \to \cdots \to X_{100}$, where each variable has 10 states, and using datasets with alternating observed and hidden variables. Each plot point represents an average over 20 simulated datasets, where we recorded the time to execute each EM algorithm (excluding the time to read networks and datasets from file, which was negligible compared to learning time).

Clearly, D-EM scales better in terms of time than both SMILE and SAMIAM, as the size of the dataset increases. As explained in the above analysis, the number of

[3]The GENIE/SMILE and SAMIAM systems are available at `http://genie.sis.pitt.edu/` and `http://reasoning.cs.ucla.edu/samiam/`. SMILE's C++ API was used to run EM, using default options, except we suppressed the randomized parameters option. SAMIAM's Java API was used to run EM (via the CodeBandit feature), also using default options, and the Hugin algorithm as the underlying inference engine.

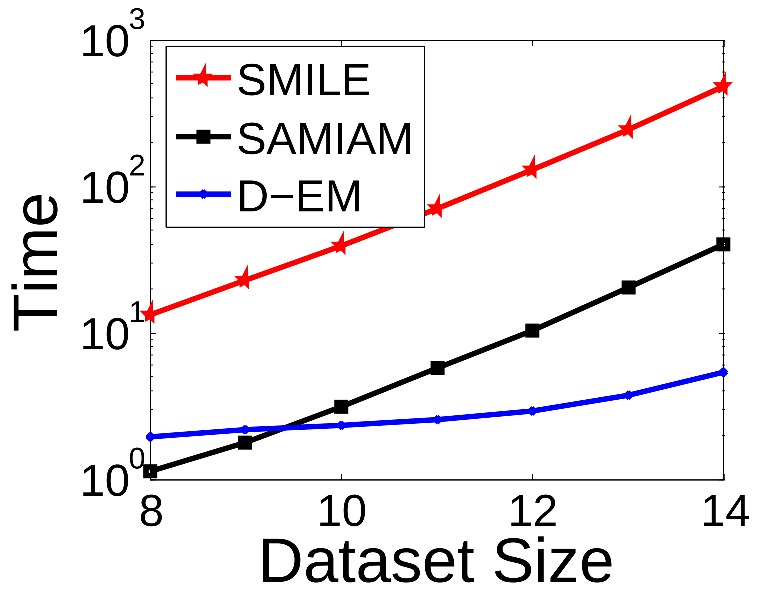


Figure 6.5: Effect of dataset size (log-scale) on learning time in seconds.

calls to the inference engine by D-EM is not necessarily linear in the dataset size. Note here that D-EM used a stricter convergence threshold and obtained better likelihoods, than both SMILE and SAMIAM, in all cases. Yet, D-EM was able to achieve one-to-two orders-of-magnitude speed-ups as the dataset grows in size. On the other hand, SAMIAM was more efficient than SMILE, but got worse likelihoods in all cases, using their default settings (the same seed was used for all algorithms).

Our second analytic explanation for the obtained savings is based on understanding the dynamics of the convergence test, used by iterative algorithms such as EM. Such algorithms employ a convergence test based on either parameter or likelihood change. According to the first test, one compares the parameter estimates obtained at iteration i of the algorithm to those obtained at iteration $i-1$. If the estimates are close enough, the algorithm converges. The likelihood test is similar, except that the likelihood of estimates is compared across iterations. In our experiments, we used a convergence test based on parameter change. In particular, when the absolute change in every parameter falls below the set threshold of 10^{-4}, convergence is declared by EM.

When learning with decomposition, each sub-network is allowed to converge independently, which can contribute significantly to the obtained savings. In particular, with enough observed variables, we have realized that the vast majority of sub-networks converge very quickly, sometimes in one iteration (when the projected dataset is complete). In fact, due to this phenomenon, the convergence threshold for sub-networks can be further tightened without adversely affecting the total running time. In our experiments, we used a threshold of 10^{-5} for D-EM, which is tighter than the threshold used for EM. Figure 6.4 (right pair) illustrates decomposed convergence, by showing the number of iterations required by each sub-network to converge, sorted decreasingly, with convergence test based on parameters (left) and likelihood (right). The vast majority of sub-networks converged very quickly. Here, convergence was declared when the change in parameters or log-likelihood, respectively, fell below the set threshold of 10^{-5}.

6.5 Related Work

The decomposition techniques we discussed in this chapter have long been utilized in the context of inference, but apparently not in learning. In particular, leaf nodes that do not appear in evidence $\mathbf{e}$ have been called *Barren nodes* in (Shachter, 1986), which showed the soundness of their removal during inference with evidence $\mathbf{e}$. Similarly, deleting edges outgoing from evidence nodes has been called *evidence absorption* and its soundness was shown in (Shachter, 1990). Interestingly enough, both of these techniques are employed by the inference engines of SamIam and SMILE,[4] even though neither seem to employ them when learning network parameters as we propose here (see earlier experiments). When employed during inference, these techniques simplify the network to *reduce the time* needed to compute queries (e.g., conditional marginals which are needed by learning algorithms). However, when employed in the context of learning, these techniques *reduce the number of calls* that need to be made to an infer-

[4]SMILE actually employs a more advanced technique known as relevance reasoning (Lin & Druzdzel, 1997).

ence engine. The difference is therefore fundamental, and the effects of the techniques are orthogonal. In fact, the inference engine we used in our experiments does employ decomposition techniques. Yet, we were still able to obtain orders-of-magnitude speed-ups when decomposing the learning problem. On the other hand, our proposed decomposition techniques do not apply fully to Markov random fields (MRFs) as the partition function cannot be decomposed, even when the data is complete (evaluating the partition function is independent of the data). However, distributed learning algorithms have been proposed in the literature. For example, the recently proposed LAP algorithm is a consistent estimator for MRFs under complete data (Mizrahi, Denil, & de Freitas, 2014). A similar method to LAP was independently introduced by (Meng, Wei, Wiesel, & III, 2013) in the context of Gaussian graphical models.

6.6 Conclusion

We proposed a technique for decomposing the problem of learning Bayesian network parameters into independent learning problems. The technique applies to incomplete datasets and is based on exploiting variables that are either hidden or observed. Our empirical results suggest that orders-of-magnitude speed-up can be obtained from this decomposition technique, when enough or particular variables are hidden or observed in the dataset. The proposed decomposition technique is orthogonal to the one used for optimizing inference as one reduces the *time* of inference queries, while the other reduces the *number* of such queries. The latter effect is due to decomposing the dataset and the convergence test. The decomposition process incurs little overhead as it can be performed in time that is linear in the structure size and dataset size. Hence, given the potential savings it may lead to, it appears that one must always try to decompose before learning network parameters.

CHAPTER 7

Data Compression in Learning MRFs

We propose in this chapter a technique for decomposing and compressing the dataset in the parameter learning problem in Markov random fields. Our technique applies to incomplete datasets and exploits variables that are always observed in the given dataset. We show that our technique allows exact computation of the gradient and the likelihood, and can lead to orders-of-magnitude savings in learning time. This chapter is based on (Refaat & Darwiche, 2015).

7.1 Introduction

This chapter is concerned with learning factors from incomplete data given a fixed structure. As pointed out in Koller and Friedman (2009) (Koller & Friedman, 2009), at every iteration, besides doing inference to compute the partition function, we need to run inference separately conditioned on every unique data example. In this chapter, we propose a technique that can significantly decrease the number of required inferences per iteration, without any loss of quality, which we highlight next.

Our goal is to alleviate the need for an inference for each unique data example. We decompose the dataset into smaller datasets each of which is over a subset of the variables, and is associated with some part of the MRF structure. We prove that to compute the objective function or its gradient, one can operate on the decomposed datasets rather than the original dataset. So why can operating on the decomposed datasets be better? We show that our proposed decomposition can create room for compressing the datasets. Accordingly, the decomposed datasets can be much smaller than the original dataset. This leads to decreasing the number of inferences required by the optimization algorithm, and can significantly decrease the learning time.

Our proposed technique exploits variables that are always observed in the dataset, and it can be performed in time that is linear in the MRF structure and dataset size.

The chapter is organized as follows. In Section 7.2, we define our notation and give an introduction to the problem of learning MRF parameters. We motivate the problem we tackle in Section 7.3. In Section 7.4, we show how the data decomposition technique works. The experimental results are given in Section 7.5. We prove that our method is sound in Section 7.6. We review some of the related work in Section 7.7, and conclude in Section 7.8.

7.2 Learning Parameters

In this section, we review how parameter estimation for MRFs is formulated as an optimization problem. The set of all parameters of the MRF is denoted by θ. Variables and their instantiations are used as subscripts for θ to denote a subset of the parameters. Namely, the network parameters are given by the vector $\theta = (\ldots, \theta_{\mathbf{X}_f}, \ldots)$. Component $\theta_{\mathbf{X}_f}$ is a parameter set for a factor f, assigning a number $\theta_{\mathbf{x}_f} > 0$ for each instantiation $\mathbf{x}_f$ of variable set $\mathbf{X}_f$.

We say that an instantiation $\mathbf{x}$ and a data example $\mathbf{d}$ are compatible, denoted by $\mathbf{x} \sim \mathbf{d}$, iff they agree on the state of their common variables. For example $\mathbf{x} = a, b, \bar{c}$ is compatible with $\mathbf{d}_1 = a, \bar{c}$ but not with $\mathbf{d}_2 = a, c$ as $c \neq \bar{c}$.

The negative log-likelihood of a dataset $\mathcal{D} = \{\ldots, \mathbf{d}_i, \ldots\}$ is denoted by $-\ell\ell(\theta|\mathcal{D})$, and given by:

$$-\ell\ell(\theta|\mathcal{D}) = -\sum_{i=1}^{N} \log Z_\theta(\mathbf{d}_i) + N \log Z_\theta \tag{7.1}$$

where N is the number of data examples, and Z_θ is the partition function. The partition function is given by $Z_\theta = \sum_{\mathbf{x}} \prod_f \theta_{\mathbf{x}_f}$, where $\sum_{\mathbf{x}}$ is a summation over all possible instantiations of $\mathbf{x}$, which determines an instantiation $\mathbf{x}_f$ for each factor f. Similarly, $\log Z_\theta(\mathbf{d}_i) = \sum_{\mathbf{x} \sim \mathbf{d}_i} \prod_f \theta_{\mathbf{x}_f}$ is the partition function conditioned on example $\mathbf{d}_i$, i.e. the summation is over the instantiations that agree with the observed values of $\mathbf{d}_i$. For simplicity, we assume, throughout the chapter, a tabular representation as given in (Refaat et al., 2013), as opposed to an exponential representation as given in Chapter 19 in (Murphy, 2012). In our experiments, however, we use the exponential representation, to avoid the need for explicit non-negativity constraints.

The first term in Equation 7.1 is called the data term, whereas the second term is called the model term. If the data is complete, Equation 7.1 is convex, if the exponential representation is used; and the data term becomes trivial to evaluate. Thus, in every optimization iteration, a single inference is typically needed to evaluate the model term.

However, when the data is incomplete, the data term is non-trivial to evaluate as it requires running inference conditioned on every data example $\mathbf{d}_i$. Thus, the number of inferences needed per iteration would be $N + 1$ [1]. An efficient package would however detect identical data examples, and do inference for every distinct data example. However, the number of distinct data examples can still be substantially large. In this chapter, we propose a technique that decomposes the data term and compresses the dataset. As a result, the number of inferences required to evaluate the data term can decrease leading to high speed-ups. We next give a motivation and explain how the technique works.

7.3 Motivation

We highlight the proposed technique by taking a closer look at the underlying optimization problem. When learning maximum likelihood parameters, the objective function

[1] N iterations are needed for doing inference conditioned on each data example, and one inference is needed for the model term.

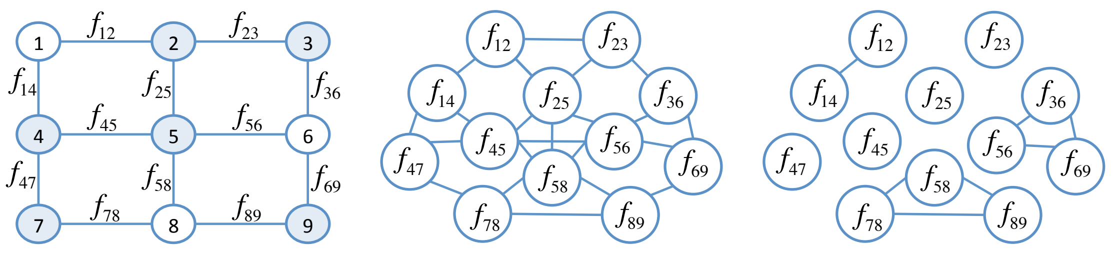
consists of the data term and the model term, as explained in Section 7.2. Unlike the data term, the model term does not depend on the dataset. In case of complete data, the data term evaluation is trivial during optimization. Thus, only one inference per iteration is required for the model term. However, when the data is incomplete, evaluation of the data term requires a number of inferences equal to the number of distinct data examples in the dataset. In this chapter, we exploit variables that are always observed in the dataset, and decompose the data term into independent terms, each of which is over a possibly much smaller dataset.

When inference is done conditioned on a data example, the graph can be pruned using the observed values, to make inference more efficient; see Chapter 6 in (Darwiche, 2009). Most learning packages, that use efficient inference engines, use such techniques that date back to Shachter (Shachter, 1986, 1990). Namely, the graph is pruned given every data example, before doing inference. The key observation that we exploit is that the pruned graphs, given all the data examples, share something in common, if some variables are always observed. This commonality is the heart of the proposed method.

To capture this commonality, we decompose the graph conditioned *only* on the variables that are always observed in the dataset. As a result, the graph is decomposed into a number of sub-graphs each of which is over a subset of the variables. We then project the dataset onto the variables of each sub-graph, by discarding the variables not in the sub-graph. We prove that evaluating the data term or computing its gradient can now be computed by doing inference in the sub-graphs and their projected datasets independently. This begs the question: *Why is this decomposition useful?*

Now, the number of variables and, therefore, unique data examples in each projected dataset is much smaller than in the original dataset. As a result, each projected dataset can be compressed significantly by detecting repetitions, and accordingly, the number of inferences required in each iteration decreases. We show empirically that the proposed method can lead to orders-of-magnitude speed-ups. In fact, data compression becomes particularly useful as the size of the dataset grows, and as the number of observed variables increases.

7.4 Data Decomposition



1 2 3 4 5 6 7 8 9
f_{12} f_{23} f_{14} f_{25} f_{36} f_{45} f_{56} f_{47} f_{58} f_{69} f_{78} f_{89}

Figure 7.1: The process of identifying graph sub-networks given observed nodes: 2, 3, 4, 5, 7, and 9. **Left**: 3×3 MRF grid. **Middle**: A graph of factors, where an edge between two factors exists if they have common variables. **Right**: The sub-networks obtained by deleting every edge between two factors if all their common variables are always observed in the data.

In this section, we explain how the data term is decomposed and, accordingly, the dataset is compressed. The proof will be given in Section 7.6. Figure 7.1 (left) shows a 3×3 grid MRF that we use as a running example. Factors in the grid are binary, i.e. involves two variables, and variables can take 2 states: $true$ (t) or $false$ (f). Suppose that Variables 1, 6, and 8 have missing values in the dataset (denoted by $?$), whereas Variables 2, 3, 4, 5, 7, and 9 are always observed (cannot take $?$). The dataset takes the form:

example/variable	1	2	3	4	5	6	7	8	9
1	t	f	f	f	t	$?$	t	$?$	t
2	$?$	t	t	f	t	$?$	f	$?$	f
3	t	f	t	f	f	t	t	$?$	t
4	$?$	t	t	f	f	$?$	f	f	t
.	.	.	.	.	.	.	.	.	.

Firstly, we create a factor graph, as given in Figure 7.1 (middle), which has a node for every factor in the original MRF. An edge between two factors exists if and only if they share a common variable. For example, f_{12} and f_{25} share Variable 2, and therefore has an edge between them in the factor graph. The decomposition will be performed on the created factor graph as shown next.

Secondly, we delete any edge, in the factor graph, if the common variables between its nodes are always observed in the dataset. For example, the edge between f_{12} and f_{25} is deleted as the common Variable 2 is always observed in the dataset. On the other hand, f_{58} and f_{89} remains intact as the common Variable 8 has missing values in the dataset. The result of this decomposition is given in Figure 7.1 (right).

Now, the decomposed factor graph in Figure 7.1 (right) decomposes the MRF into multiple sub-networks, each of which is over a subset of factors. For example, factors f_{12} and f_{14} forms a sub-network. As we will prove in Section 7.6, the data term decomposes into a summation of independent terms corresponding to each sub-network.

Thirdly, we project the dataset onto the variables of each sub-network, by discarding the variables not in the sub-network. For example, we project the dataset onto the variables of the sub-network, $f_{12} - f_{14}$, by discarding all variables except 1, 2, and 4. Thus, this projected dataset will have only 3 columns, and will take the form:

example/variable	1	2	4
1	t	f	f
2	$?$	t	f
3	t	f	f
4	$?$	t	f
.	.	.	.

Now, we have a projected dataset for each sub-network. Finally, we will compress every projected dataset by detecting repetitions and adding a count field to every distinct data example. For example, the compressed dataset for $f_{12} - f_{14}$ may take the form:

example/variable	1	2	4	Count
1	t	f	f	5
2	?	t	f	15
.	.	.	.	.

where the count keeps record of how many times the distinct data example was repeated in the dataset. The key observation here is that the number of repetitions increases in the projected datasets. By a simple counting argument, one can show that the maximum number of possible distinct data examples that can appear in the original dataset is $2^6 \times 3^3 = 1728$. [2] On the other hand, the maximum number of distinct data examples in the projected dataset of $f_{12} - f_{14}$, for instance, can be at most only $2^2 \times 3^1 = 12$.

Every sub-network with its own projected, and potentially compressed, dataset now induces an independent data term, defined exactly as the original data term, as we will show in Section 7.6. We will prove that the original data term is equivalent to the summation of all the independent sub-network data terms. Thus, to evaluate or compute the gradient of the original data term, one needs to evaluate or compute the gradient of the data term of each sub-network independently, and then combine them by addition.

The decomposition above suggests that as more variables are always observed, the MRF is decomposed into much smaller sub-networks leading to more repetitions. Moreover, as the dataset size gets larger, the difference between the data before and after decomposition is magnified. Now, we will show that, experimentally, data decomposition works as expected and can achieve orders-of-magnitude savings in time.

7.5 Experimental Results

We compare the time taken by the gradient method if data decomposition is used, versus if the original dataset is used. In particular, using a fixed network structure, we simulate a dataset, then make the data incomplete by randomly selecting a certain percentage of

[2]Note that a variable with missing values can take true, false, or be missing.

variables to have missing values. After that, we learn the parameters from the data using the gradient method with and without data decomposition, to obtain a local optimum.

For 11 different networks, [3] and with hiding 20% of the variables, Table 7.1 shows the time taken by the gradient method without data decomposition t_{grad}, and with data decomposition t_{d-grad}, together with the speed-up achieved by data decomposition, computed as $\frac{t_{grad}}{t_{d-grad}}$. In all cases, the same learned parameters were returned by both techniques, which we do not show in the table.

One can see that data decomposition achieved one-to-two orders-of-magnitude speed-up in learning time in most cases. The decomposition technique has almost left the dataset of network 54.wcsp without much decomposition leading to little speed-up.

In this experiment, we did not vary the percentage of observed variables nor the dataset size. We selected the dataset in each case to be as small as possible without making computing the data term much easier than the model term. We next analyze the behavior of the decomposition technique when the percentage of observed variables or the dataset size changes.

As the motivating example in Section 7.4 suggests, we expect that data decomposition will behave favorably as the dataset gets larger, and as more variables are always observed in the dataset.

Figure 7.2 shows the speed-up achieved with a dataset of 2^{12} examples, while always observing different percentages of variables, using 4 different structures. As expected, as less nodes have missing values, the speed-up increases.

Figure 7.3 shows the speed-up achieved while using a hiding percentage of 20%, for different dataset sizes, in log-scale. Indeed, the speed-up is directly proportional to the dataset size. In this case, orders-of-magnitude speed-up was achieved starting from 2^{14} examples.

[3]Among the used structures are randomly generated chains (Chain-50 and Chain-100), and randomly generated binary trees (Tree-63, Tree-127, Tree-225). Grid9x9 and Grid10x10 are grid networks. Network 54.wcsp is a weighted CSP problem, whereas Network win95pts is an expert system for printer troubleshooting in Windows 95. Network smokers is a relational Markov network.

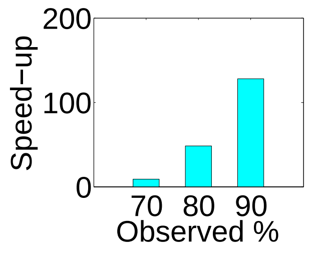
Table 7.1: The execution time taken by the gradient method (without and with data decomposition), together with the speed-up achieved when data decomposition is used.

Network	#vars	data size	t_{grad}	t_{d-grad}	speed-up
Chain − 50	50	4096	4.7 mins	1.7 secs	165×
Chain − 100	100	4096	12.2 mins	2.9 secs	253×
Tree − 63	63	4096	6 mins	2.36 secs	152×
Tree − 127	127	4096	18 mins	5.77 secs	187×
Tree − 255	255	4096	53 mins	13 secs	236×
Grid9x9	81	8192	61 mins	73 secs	50×
Grid10x10	100	8192	74 mins	34 secs	130×
alarm	37	4096	3.4 mins	13.6 secs	15×
54.wcsp	67	1024	1.45 mins	1.32 mins	1.1×
win95pts	76	1024	4.3 mins	7.8 secs	33×
smokers	120	2048	16 mins	1.64 mins	9.7×

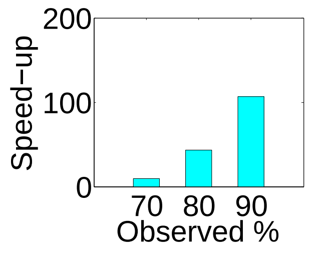
Given the difficulty of the problem, the speed-up achieved by data decomposition can be indispensable. For example, for a 9×9 Grid, 2^{16} data examples, and 20% hiding percentage, the gradient method took about 3 minutes by data decomposition versus about 17 hours by detecting repetitions in the original dataset.

Decomposition can also make learning feasible in large problems. For example, we were able to learn the parameters of a network from the field of genetics (Family2Recessive), that has 385 factors, from 2^{12} examples with 20% hiding, in about 55 minutes, by data decomposition. However, we were not able to get results, in less than a day, without data decomposition, for this network.

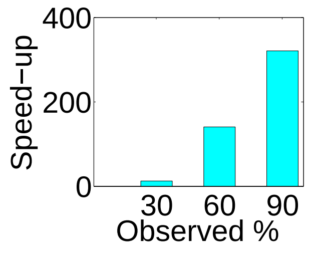
We next show that an existing prominent package (FastInf) does not appear to use the proposed data decomposition. We compare the speed-up obtained of our basic implementation of gradient descent and EM, that use data decomposition, against FastInf (Jaimovich, Meshi, McGraw, & Elidan, 2010). We provided our system and FastInf with the same initial parameters, same incomplete datasets (hiding 20%), and



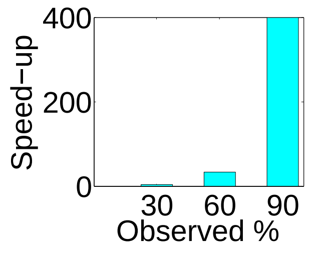
(a) Grid



(b) Alarm



(c) Chain



(d) Tree

Figure 7.2: The speed-up obtained by data decomposition on different percentages of always observed variables, for 4 different network structures: 9x9 grid, alarm, chain (50 variables), tree (63 variables).

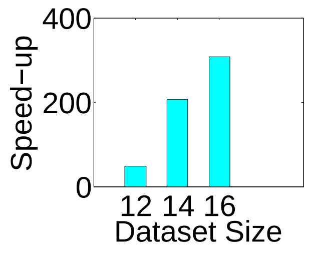
the same MRF structure.

Figure 7.4 shows the speed-up obtained by our Gradient and EM methods, that use data decomposition (allowed 100 iterations), over FastInf EM (with the gradient option allowed only 2 iterations), for different dataset sizes in log-scale. One can see that as the data increases, more speed-up is achieved. We were able to get a better likelihood too as we run our system for more iterations, and still achieve high speed-ups.

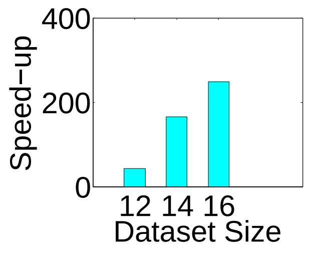
In Figure 7.5, we fix the dataset size to 2^{12} and show the speed-up obtained by our technique over FastInf EM with different algorithm options: 0-FR, 1-PR, 2-BFGS, 3-STEEP, 4-NEWTON [4]. Newton method was not successful and, therefore, not shown in the figure.

Although FastInf uses approximate inference, and our implementation is based on exact inference, we were still able to realize orders-of-magnitude speed-ups over

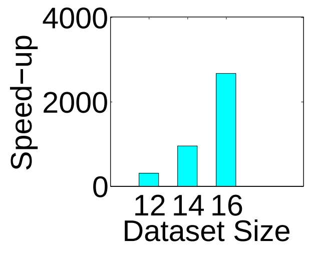
[4]For details about different algorithm options in FastInf, see (Jaimovich et al., 2010)



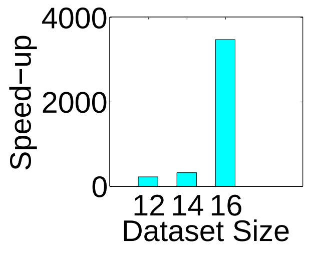
(a) Grid



(b) Alarm



(c) Chain



(d) Tree

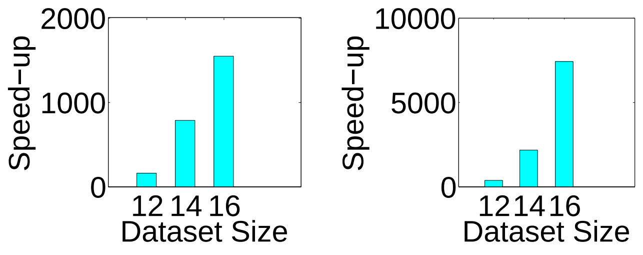
Figure 7.3: The speed-up obtained by data decomposition on different dataset sizes in log-scale, for 4 different network structures: 9x9 grid, alarm, chain (50 variables), tree (63 variables).

FastInf, as the dataset size increases.

We note too that data decomposition is done once and can be performed in time that is linear in the MRF structure size and dataset size. The execution time of our methods used to compute the speed-ups did involve the time needed to decompose the graph and decompose the data. In the next section, we prove that data decomposition is exact, and does not compromise quality.

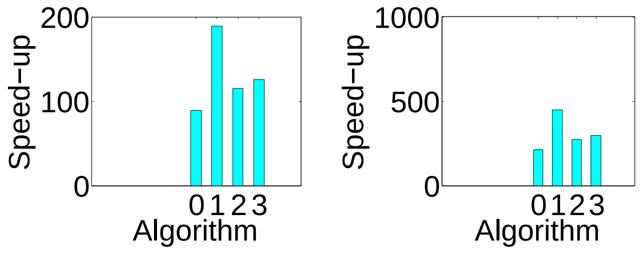
7.6 Soundness

In this section, we prove that our decomposition technique is sound. Before we give our decomposition theorem, we review the notion of parameter terms, initially introduced in the context of Bayesian networks in (Refaat et al., 2014).



(a) Gradient (b) EM

Figure 7.4: Speed-up of Gradient and EM methods (respectively) that use data decomposition, (allowed 100 iterations) over FastInf EM (allowed 2 iterations) on different dataset sizes in log-scale. Network alarm was used; 20% of the nodes have missing values in the data.



(a) Gradient (b) EM

Figure 7.5: Speed-up of Gradient and EM methods, that use data decomposition, over different FastInf algorithms on 2^{12} data examples. Network alarm was used; 20% of the nodes have missing values in the data.

7.6.1 Parameter Terms

Two parameters are *compatible*, denoted by $\theta_{\mathbf{x}_f} \sim \theta_{\mathbf{x}_{f'}}$, iff they agree on the state of their common variables. For example, parameters $\theta_{\mathbf{x}_f=xy}$ and $\theta_{\mathbf{x}_{f'}=zy}$ are compatible, but parameters $\theta_{\mathbf{x}_f=x\overline{y}}$ and $\theta_{\mathbf{x}_{f'}=zy}$ are not compatible, as $y \neq \overline{y}$.

Moreover, a parameter is compatible with an example iff they agree on the state of their common variables. For example, parameter $\theta_{\mathbf{x}_f=x\overline{y}}$ is compatible with example $x, \overline{y}, z, v$, but not with example x, y, z, v. The definition of a parameter term is given as follows:

Definition 7 (Parameter Term) *Let* $\mathbf{F}$ *be a set of network factors and let* $\mathbf{d}$ *be a data example. A* <u>*parameter term*</u> *for* $\mathbf{F}$ *and* $\mathbf{d}$*, denoted* $\Theta_{\mathbf{F}}^{\mathbf{d}}$*, is a product of compatible network parameters, one for each factor in* $\mathbf{F}$*, that are also compatible with example* $\mathbf{d}$*.*

For example, consider an MRF with 3 factors $\{\theta_{\mathbf{X}_{f_1}=X}, \theta_{\mathbf{X}_{f_2}=XY}, \theta_{\mathbf{X}_{f_3}=YZ}\}$; and let $\mathbf{F}$ be the subset of factors $\{\theta_{\mathbf{X}_{f_1}=X}, \theta_{\mathbf{X}_{f_2}=XY}\}$ and $\mathbf{d} = \overline{x}, z$. Then, $\Theta_{\mathbf{F}}^{\mathbf{d}}$ will denote either $\theta_{\mathbf{x}_{f_1}=\overline{x}}.\theta_{\mathbf{x}_{f_2}=\overline{x}y}$ or $\theta_{\mathbf{x}_{f_1}=\overline{x}}.\theta_{\mathbf{x}_{f_2}=\overline{x}.\overline{y}}$.

When $\mathbf{F}$ has all the MRF factors, i.e. $\mathbf{F} = \{\theta_{\mathbf{X}_{f_1}=X}, \theta_{\mathbf{X}_{f_2}=XY}, \theta_{\mathbf{X}_{f_3}=YZ}\}$, then $\Theta_{\mathbf{F}}^{\mathbf{d}}$ will denote either $\theta_{\mathbf{x}_{f_1}=\overline{x}}.\theta_{\mathbf{x}_{f_2}=\overline{x}y}.\theta_{\mathbf{x}_{f_3}=yz}$ or $\theta_{\mathbf{x}_{f_1}=\overline{x}}.\theta_{\mathbf{x}_{f_2}=\overline{x}\,\overline{y}}.\theta_{\mathbf{x}_{f_3}=\overline{y}z}$; in this case $Z_\theta(\mathbf{d}) = \sum_{\Theta_{\mathbf{F}}^{\mathbf{d}}} \Theta_{\mathbf{F}}^{\mathbf{d}}$. Armed with parameter terms, we are now ready to state our decomposition theorem.

Theorem 11 *The data term is decomposed into a number of smaller functions corresponding to sub-networks. The log-likelihood takes the form:*

$$\ell\ell(\theta|\mathcal{D}) = \sum_s \sum_{i=1}^{N^s} n_i^s \log Z_\theta^s(\mathbf{d}_i) - N \log Z_\theta \tag{7.2}$$

where n_i^s *is the number of times that distinct* d_i *appears in the projected dataset of Sub-network* s*,* N^s *is the total number of distinct data examples in the projected dataset of Sub-network* s*, and* $Z_\theta^s(\mathbf{d}_i)$ *is the partition function of Sub-network* s *conditioned on example* d_i*.*

Proof We will proceed by induction, decomposing one sub-network, and operating inductively on the rest of the network to decompose all the sub-networks.

We note that the data term in the log-likelihood function, $\sum_{i=1}^N \log Z_\theta(\mathbf{d}_i)$, can be written as:

$$\sum_{i=1}^N \log Z_\theta(\mathbf{d}_i) = \sum_{i=1}^N \log \sum_{\Theta_{\mathbf{F}}^{\mathbf{d}_i}} \Theta_{\mathbf{F}}^{\mathbf{d}_i} \tag{7.3}$$

Where N is the number of data points in the dataset [5], and $\mathbf{F}$ is the set of all factors in the MRF. Let $\mathbf{F}^s$ be the set of factors in Sub-network s, and $\mathbf{F}^{s'}$ be the set of all the rest

[5]In this case, not necessarily distinct.

of the factors. By definition of parameter terms, the data term can be re-written as:

$$\sum_{i=1}^{N} \log Z_\theta(\mathbf{d}_i) = \sum_{i=1}^{N} \log \big(\sum_{\Theta^{\mathbf{d}_i}_{\mathbf{F}^s}} \Theta^{\mathbf{d}_i}_{\mathbf{F}^s} \sum_{\Theta^{\mathbf{d}_i}_{\mathbf{F}^{s\prime}} \sim \Theta^{\mathbf{d}_i}_{\mathbf{F}^s}} \Theta^{\mathbf{d}_i}_{\mathbf{F}^{s\prime}}\big) \tag{7.4}$$

where the fourth summation is over all $\Theta^{\mathbf{d}_i}_{\mathbf{F}^{s\prime}}$ that agree on the state of their common variables with $\Theta^{\mathbf{d}_i}_{\mathbf{F}^s}$, which is denoted by the compatibility: $\Theta^{\mathbf{d}_i}_{\mathbf{F}^{s\prime}} \sim \Theta^{\mathbf{d}_i}_{\mathbf{F}^s}$.

By the decomposition procedure, the common variables between Sub-network s and the rest of the network are always observed. Otherwise, the sub-network would not have been separated from the rest. Therefore, $\Theta^{\mathbf{d}_i}_{\mathbf{F}^{s\prime}}$ and $\Theta^{\mathbf{d}_i}_{\mathbf{F}^s}$ always agree on the common variables, which are determined by $\mathbf{d}_i$. Thus, there is no need to ensure compatibility, and the data term can be written as:

$$\begin{aligned} \sum_{i=1}^{N} \log Z_\theta(\mathbf{d}_i) &= \sum_{i=1}^{N} \log\big(\sum_{\Theta^{\mathbf{d}_i}_{\mathbf{F}^s}} \Theta^{\mathbf{d}_i}_{\mathbf{F}^s} \sum_{\Theta^{\mathbf{d}_i}_{\mathbf{F}^{s\prime}}} \Theta^{\mathbf{d}_i}_{\mathbf{F}^{s\prime}}\big) = \\ \sum_{i=1}^{N} \log \sum_{\Theta^{\mathbf{d}_i}_{\mathbf{F}^s}} \Theta^{\mathbf{d}_i}_{\mathbf{F}^s} &+ \sum_{i=1}^{N} \log \sum_{\Theta^{\mathbf{d}_i}_{\mathbf{F}^{s\prime}}} \Theta^{\mathbf{d}_i}_{\mathbf{F}^{s\prime}} \end{aligned} \tag{7.5}$$

Now the distinct data points with respect to Sub-network s can be detected, to get:

$$\sum_{i=1}^{N} \log Z_\theta(\mathbf{d}_i) = \sum_{i=1}^{N^s} n_i^s \log \sum_{\Theta^{\mathbf{d}_i}_{\mathbf{F}^s}} \Theta^{\mathbf{d}_i}_{\mathbf{F}^s} + \sum_{i=1}^{N} \log \sum_{\Theta^{\mathbf{d}_i}_{\mathbf{F}^{s\prime}}} \Theta^{\mathbf{d}_i}_{\mathbf{F}^{s\prime}} \tag{7.6}$$

By observing that $\sum_{\Theta^{\mathbf{d}_i}_{\mathbf{F}^s}} \Theta^{\mathbf{d}_i}_{\mathbf{F}^s}$ is equivalent to the partition function of Sub-network s conditioned on d_i, we get:

$$\sum_{i=1}^{N} \log Z_\theta(\mathbf{d}_i) = \sum_{i=1}^{N^s} n_i^s \log Z^s_\theta(\mathbf{d}_i) + \sum_{i=1}^{N} \log \sum_{\Theta^{\mathbf{d}_i}_{\mathbf{F}^{s\prime}}} \Theta^{\mathbf{d}_i}_{\mathbf{F}^{s\prime}} \tag{7.7}$$

We continue inductively on the rest of the network to decompose all the sub-networks.
□

7.7 Related Work

Some work on decomposing MRFs and Bayesian networks (BNs) exist in literature. In the context of inference in BNs, pruning *Barren* nodes and edges outgoing from observed variables was initially proposed in (Shachter, 1986, 1990) [6].

In the context of parameter learning from incomplete data, decomposing the BN optimization problem was proposed in (Refaat et al., 2014), where the notion of *parameter terms* was introduced. Namely, it was shown that fully observed variables may be exploited to decompose the optimization problem into independent problems, leading to both data decomposition and independent convergence. In this chapter, we migrate this concept to the context of MRFs. While the partition function makes decomposing the optimization problem exactly, as in (Refaat et al., 2014), hard, we showed here that similar decomposition techniques can be used to decompose the data term, leading to decomposing, and potentially, compressing the dataset.

For MRFs, the LAP algorithm (Mizrahi et al., 2014) deals with approximately decomposing MRFs in the case of complete data, where they showed that LAP behaves similarly to pseudo-likelihood and maximum likelihood, for large sample sizes, while being more efficient. A similar method was independently introduced by (Meng et al., 2013) in the context of *Gaussian graphical models*. Our work stands out from the LAP algorithm in dealing with incomplete data, and in being equivalent to maximizing the likelihood. However, our proposed technique does not help in the case of complete data, as the data term becomes trivial.

7.8 Conclusion

We proposed a technique for decomposing the dataset to learn MRF parameters from incomplete data. The technique works by decomposing the MRF to sub-networks based

[6]Pruning edges migrates to MRFs.

on variables that are always observed in the incomplete dataset. The dataset is then projected on each sub-network, and compressed by detecting repetitions.

The key observation, that data compression relies on, is that sub-networks typically have a small number of variables. Thus, it is likely that more repetitions, and, accordingly, compression can take place. Our empirical results suggest that orders-of-magnitude speed-ups may be obtained using data decomposition.

The decomposition process incurs very little overhead as it can be performed in time that is linear in the MRF structure size and dataset size. Hence, given the potential savings it may lead to, it appears that one must always try to decompose the incomplete dataset before learning maximum likelihood MRF parameters.

CHAPTER 8

Summary of Contributions

In this thesis, we showed that special structure in the dataset could be exploited to learn probabilistic graphical models from incomplete data more efficiently. Namely, fully observed variables can be used to decompose the learning problem in Bayesian networks into independent learning problems, without any loss of quality. Furthermore, in Markov Random Fields, fully observed variables can be exploited to significantly compress the dataset leading to orders-of-magnitude speed-ups in learning time. We also proposed EDML, as an alternative to EM, in the context of learning graphical models. We argued that EDML is more sensitive to the degree of completeness in the data, and can have better convergence properties than EM, both theoretically and practically.

When we solve a general optimization problem, all we know is the objective function and the constraints. However, when learning probabilistic graphical model parameters, we have a graph structure and a dataset that impose a specific structure on the underlying optimization problem. We have presented in this thesis a number of techniques that exploit this structure in order to solve the corresponding optimization problem more efficiently.

APPENDIX A

Proofs of Chapter 4

Proof of Theorem 2

Proof We have:

$$Pr(\theta_X|\eta^1, \ldots, \eta^n) \propto \rho(\theta_X) \prod_{i=1}^{n} \sum_x \lambda_x^i \theta_x \tag{A.1}$$

where $1 \leq i \leq n$. The product of log concave functions where at least one is strictly log concave is strictly log concave (Boyd & Vandenberghe, 2004). Thus, in order to prove that $\rho(\theta_X) \prod_{i=1}^{n} \sum_x \lambda_x^i \theta_x$ is strictly log concave, it is sufficient to show that $\rho(\theta_X)$ and $\sum_x \lambda_x^i \theta_x$ for every i are log concave and at least one of them is strictly log concave. In fact, $\sum_x \lambda_x^i \theta_x$ for all i are strictly log concave. Consider $f_i(\theta_X) = \sum_x \lambda_x^i \theta_x$ for some arbitrary i. Firstly, $f_i(\theta_X)$ is an affine function. Moreover, $f_i(\theta_X)$ has a range of values from the smallest λ_x^i to the largest λ_x^i. Furthermore, this range is continuous since any number in the range can be achieved by a weighted average of the maximum and the minimum. Since $\log(x)$ is strictly concave over the domain defined by the range of $\sum_x \lambda_x^i \theta_x$, $\log f_i(\theta_X)$ is strictly concave. It follows that $f_i(\theta_X)$ is strictly log concave. On the other hand, we have:

$$\log \rho(\theta_X) = \sum_x (\psi_x - 1) \log \theta_x \tag{A.2}$$

The hessian of $\log \rho(\theta_X)$ is a diagonal matrix because all non-diagonal elements vanish due to differentiating with respect to θ_x and then with respect to $\theta_{x'}$, where x and x' correspond to the row and column indices of the hessian, respectively. Moreover, it is easy to show that all the diagonal elements are negative and, therefore, the hessian is negative definite (Boyd & Vandenberghe, 2004) Thus, $\log \rho(\theta_X)$ is concave. Finally, If all λ_x^i for some i are equal, $f_i(\theta_X)$ becomes a constant function, and, therefore, not strictly log concave. For the proof to hold, λ_x^i for all x should not be all equal for at least one $f_i(\theta_X)$. $\square$

Proof of Theorem 3

We want MAP estimates $\theta^\star$ maximizing $\rho(\theta \mid \eta)$, where $\eta = \{\eta_1, \ldots, \eta_N\}$ is soft (virtual) evidence on a soft-evidence island.

Say we have a Dirichlet prior $\rho(\theta) = \alpha \prod_x [\theta_x]^{\psi_x - 1}$, where α is a normalizing constant. Given current estimates θ^t we want new estimates θ maximizing the expected log posterior:

$$\begin{aligned}
ELP(\theta \mid \theta^t) &= \sum_{\mathbf{x}} Pr(\mathbf{x} \mid \eta, \theta^t) \log \rho(\mathbf{x}, \theta \mid \eta) \\
&= \sum_{\mathbf{x}} Pr(\mathbf{x} \mid \eta, \theta^t) \log \frac{Pr(\mathbf{x}, \eta \mid \theta)\rho(\theta)}{\rho(\eta)} \\
&= -\log \rho(\eta) + \log \rho(\theta) + \sum_{\mathbf{x}} Pr(\mathbf{x} \mid \eta, \theta^t) \log Pr(\mathbf{x}, \eta \mid \theta) \\
&= -\log \rho(\eta) + \log \rho(\theta) + ELL(\theta \mid \theta^t) \\
&= -\log \rho(\eta) + \log \alpha + \sum_x (\psi_x - 1) \log \theta_x + ELL(\theta \mid \theta^t)
\end{aligned}$$

where $ELL(\theta \mid \theta^t)$ is the expected log likelihood:

$$\begin{aligned}
ELL(\theta \mid \theta^t) &= \sum_{\mathbf{x}} Pr(\mathbf{x} \mid \eta, \theta^t) \log Pr(\mathbf{x}, \eta \mid \theta) \\
&= \sum_{\mathbf{x}} Pr(\mathbf{x} \mid \eta, \theta^t) \log \prod_i Pr(x_i, \eta_i \mid \theta) \\
&= \sum_i \sum_{\mathbf{x}} Pr(\mathbf{x} \mid \eta, \theta^t) \log Pr(x_i, \eta_i \mid \theta) \\
&= \sum_i \sum_{x_i} Pr(x_i \mid \eta_i, \theta^t) \log Pr(x_i, \eta_i \mid \theta) \\
&= \sum_x \sum_i Pr(x_i \mid \eta_i, \theta^t) \log Pr(x_i, \eta_i \mid \theta) \\
&= \sum_x \sum_i Pr(x_i \mid \eta_i, \theta^t) \log \lambda^i_{x_i} \theta_x \\
&= \sum_x \sum_i Pr(x_i \mid \eta_i, \theta^t) \log \lambda^i_{x_i} + \sum_x \sum_i Pr(x_i \mid \eta_i, \theta^t) \log \theta_x
\end{aligned}$$

where only the second term mentions θ. Thus it suffices to maximize:

$$\sum_x (\psi_x - 1) \log \theta_x + \sum_x \sum_i Pr(x_i \mid \eta_i, \theta^t) \log \theta_x$$

We can multiply this equation by the constant $[\psi_X - |X| + N]^{-1}$ and maximize:

$$\sum_x \left[\frac{(\psi_x - 1) + \sum_{i=1}^N Pr(x_i \mid \eta_i, \theta^t)}{\psi_X - |X| + N} \right] \log \theta_x$$

Note that the bracketed expression is a normalized distribution $Q(X)$, so to maximize this expression, we maximize $\sum_x Q(x) \log \theta_x$, which is uniquely maximized by $\theta_x = Q(x)$. Note that $Pr(x_i \mid \eta_i, \theta^t) = \frac{\lambda^i_x \theta_x}{\sum_y \lambda^i_y \theta_y}$. Thus, our expected log posterior is maximized by our fixed-point update for optimization in a soft-evidence island:

$$Q(X) = \frac{(\psi_x - 1) + \sum_{i=1}^N \frac{\lambda^i_x \theta_x}{\sum_y \lambda^i_y \theta_y}}{\psi_X - |X| + N}$$

To show that this update monotonically increases the log posterior, observe that

$ELP(\theta \mid \theta^t)$ is also maximized by:

$$
\begin{aligned}
f(\theta \mid \theta^t) &= \sum_{\mathbf{x}} Pr(\mathbf{x} \mid \eta, \theta^t) \log \frac{\rho(\mathbf{x}, \theta \mid \eta)}{\rho(\mathbf{x}, \theta^t \mid \eta)} \\
&= \sum_{\mathbf{x}} Pr(\mathbf{x} \mid \eta, \theta^t) \log \frac{Pr(\mathbf{x} \mid \eta, \theta)}{Pr(\mathbf{x} \mid \eta, \theta^t)} \frac{\rho(\theta \mid \eta)}{\rho(\theta^t \mid \eta)} \\
&= \log \frac{\rho(\theta \mid \eta)}{\rho(\theta^t \mid \eta)} + \sum_{\mathbf{x}} Pr(\mathbf{x} \mid \eta, \theta^t) \log \frac{Pr(\mathbf{x} \mid \eta, \theta)}{Pr(\mathbf{x} \mid \eta, \theta^t)} \\
&= \log \frac{\rho(\theta \mid \eta)}{\rho(\theta^t \mid \eta)} - KL(Pr(\mathbf{x} \mid \eta, \theta^t), Pr(\mathbf{x} \mid \eta, \theta))
\end{aligned}
$$

Now, let:

$$\theta^{t+1} = \operatorname*{argmax}_{\theta} ELP(\theta \mid \theta^t) = \operatorname*{argmax}_{\theta} f(\theta \mid \theta^t)$$

Note that $\operatorname{argmax}_{\theta} f(\theta \mid \theta^t) \geq 0$ since $f(\theta \mid \theta^t) = 0$ when $\theta = \theta^t$. Moreover, since the KL–divergence is non-negative:

$$\log \frac{\rho(\theta^{t+1} \mid \eta)}{\rho(\theta^t \mid \eta)} \geq \log \frac{\rho(\theta^{t+1} \mid \eta)}{\rho(\theta^t \mid \eta)} - KL(Pr(\mathbf{x} \mid \eta, \theta^t), Pr(\mathbf{x} \mid \eta, \theta)) \geq 0$$

and thus the posterior is increasing.

APPENDIX B

Proofs of Chapter 5

Proof of Theorem 5 First, the probability of an example $\mathbf{d}_i \in \mathcal{D}$ is:

$$Pr_\theta(\mathbf{d}_i) = \sum_{\mathbf{x} \sim \mathbf{d}_i} \prod_{x|\mathbf{u} \sim \mathbf{x}} \theta_{x|\mathbf{u}}$$

where operator $\sim$ denotes compatibility between two instantiations (they set the same value to common variables). For a fixed parameter set $\theta_{X|\mathbf{u}}$, the probability $Pr_\theta(\mathbf{d}_i)$ is a linear function with respect to the parameters of $\theta_{X|\mathbf{u}}$:

$$\begin{aligned} Pr_\theta(\mathbf{d}_i) &= Pr_\theta(\neg\mathbf{u}, \mathbf{d}_i) + \sum_x Pr_\theta(x\mathbf{u}, \mathbf{d}_i) \\ &= Pr_\theta(\neg\mathbf{u}, \mathbf{d}_i) + \sum_x \frac{\partial Pr_\theta(\mathbf{d}_i)}{\partial \theta_{x|\mathbf{u}}} \theta_{x|\mathbf{u}} \\ &= C^i_{\mathbf{u}} + \sum_x C^i_{x|\mathbf{u}} \cdot \theta_{x|\mathbf{u}} \end{aligned}$$

where $C^i_{\mathbf{u}}$ and $C^i_{x|\mathbf{u}}$ are constants with respect to $\theta_{X|\mathbf{u}}$. Moreover $Pr_\theta(\neg\mathbf{u}, \mathbf{d}_i) = Pr_\theta(\mathbf{d}_i) - Pr_\theta(\mathbf{u}, \mathbf{d}_i)$. Thus our sub-function, the negative log-likelihood with respect to parameter set $\theta_{X|\mathbf{u}}$, has the form:

$$f_{\theta^\star}(\theta_{X|\mathbf{u}}) = -\sum_{i=1}^{N} \log\Big(C^i_{\mathbf{u}} + \sum_x C^i_{x|\mathbf{u}} \cdot \theta_{x|\mathbf{u}}\Big).$$

□

Proof of Theorem 6 The log-likelihood of soft evidence in this model is:

$$\begin{aligned}\log \mathbb{P}(\eta|\theta_{X|\mathbf{u}}) &= \sum_{i=1}^{N} \log \mathbb{P}(\eta_i|\theta_{X|\mathbf{u}}) \\ &= \sum_{i=1}^{N} \log \sum_{x_i} \mathbb{P}(\eta_i|x_i, \theta_{X|\mathbf{u}}) \mathbb{P}(x_i|\theta_{X|\mathbf{u}}) \\ &= \sum_{i=1}^{N} \log \sum_{x_i} \mathbb{P}(\eta_i|x_i) \cdot \theta_{x|\mathbf{u}}.\end{aligned}$$

If we substitute $\mathbb{P}(\eta_i|x_i) = C^i_{\mathbf{u}} + C^i_{x|\mathbf{u}}$, we have

$$\begin{aligned}\log \mathbb{P}(\eta|\theta_{X|\mathbf{u}}) &= \sum_{i=1}^{N} \log \sum_{x} \left(C^i_{\mathbf{u}} + C^i_{x|\mathbf{u}}\right) \cdot \theta_{x|\mathbf{u}} \\ &= \sum_{i=1}^{N} \log \left(C^i_{\mathbf{u}} + \sum_{x} C^i_{x|\mathbf{u}} \theta_{x|\mathbf{u}}\right)\end{aligned}$$

which is Equation 5.2, negated. □

Proof of Theorem 7 Suppose that θ^* is optimal for Equation 5.6. Multiplying an arbitrary $\theta^*_{\mathbf{X}_a}$ by a constant, results in multiplying both Z_θ, and $Z_\theta(\mathbf{d}_i)$, by the same constant, which cancels out in each pair of terms, $\log Z_\theta - \log Z_\theta(\mathbf{d}_i)$, preserving the same optimal objective value. Thus, one could always find an optimal θ where $\theta_{\mathbf{X}_a} \propto \theta^*_{\mathbf{X}_a}$, that is optimal for Equation 5.6, and where $Z_\theta = \alpha$.

Thus, fixing $Z_\theta = \alpha$ does not exclude the optimal solution for Equation 5.6, which can be now reduced to:

$$f(\theta) = N \log \alpha - \sum_{i=1}^{N} \log Z_\theta(\mathbf{d}_i) \qquad \text{(B.1)}$$

with a feasibility constraint that $Z_\theta = \alpha$.

Equation B.1 is equivalent to Equation 5.7, since $N \log \alpha$ is a constant. As a result, if $g(\theta)$ is feasible and optimal for Equation 5.7, then any θ, where $\theta_{\mathbf{X}_a} \propto g(\theta_{\mathbf{X}_a}) \ \forall \ \mathbf{X}_a$ is optimal for Equation 5.6. We will next prove the second part of the theorem.

The partial derivative of the log likelihood $\ell\ell(\theta|\mathcal{D})$ w.r.t. parameter $\theta_{\mathbf{x}_a}$ is:

$$\frac{\partial \ell\ell}{\partial \theta_{\mathbf{x}_a}} = -\frac{N}{Z_\theta}\frac{\partial Z_\theta}{\partial \theta_{\mathbf{x}_a}} + \sum_{i=1}^{N} \frac{1}{Z_\theta(\mathbf{d}_i)}\frac{\partial Z_\theta(\mathbf{d}_i)}{\partial \theta_{\mathbf{x}_a}}.$$

First, note that:

$$\tfrac{1}{Z_\theta}\tfrac{\partial Z_\theta}{\partial \theta_{\mathbf{x}_a}}\theta_{\mathbf{x}_a} = Pr(\mathbf{x}_a), \tfrac{1}{Z_\theta(\mathbf{d}_i)}\tfrac{\partial Z_\theta(\mathbf{d}_i)}{\partial \theta_{\mathbf{x}_a}}\theta_{\mathbf{x}_a} = Pr_\theta(\mathbf{x}_a|\mathbf{d}_i)$$

Thus, with some re-arranging, we obtain:

$$Pr_\theta(\mathbf{x}_a) = \frac{1}{N}\sum_{i=1}^{N} Pr_\theta(\mathbf{x}_a|\mathbf{d}_i) \tag{B.2}$$

which is the "moment matching" condition for parameter estimation in Markov networks. Second, consider the simplified objective: $f(\theta) = -\sum_{i=1}^{N}\log Z_\theta(\mathbf{d}_i)$ which is subject to the constraint $Z = \alpha$. We construct the Lagrangian $L(\theta, \overline{\mathbf{u}}) = f(\theta) + \overline{\mathbf{u}}(Z - \alpha)$. Setting to zero the partial derivative w.r.t. $\overline{\mathbf{u}}$, we obtain our constraint $Z = \alpha$. The partial derivative w.r.t. parameter $\theta_{\mathbf{x}_a}$ is:

$$-\sum_{i=1}^{N} \frac{1}{Z_\theta(\mathbf{d}_i)}\frac{\partial Z_\theta(\mathbf{d}_i)}{\partial \theta_{\mathbf{x}_a}} + \overline{\mathbf{u}}\frac{\partial Z_\theta}{\partial \theta_{\mathbf{x}_a}}.$$

We set the partial derivative to zero, multiply the second term by $\frac{\alpha}{Z} = 1$, and re-arrange, giving us:

$$\overline{\mathbf{u}}\alpha Pr_\theta(\mathbf{x}_a) = \sum_{i=1}^{N} Pr_\theta(\mathbf{x}_a \mid \mathbf{d}_i).$$

Summing each equation for all instantiations $\mathbf{x}_a$, we identify $\overline{\mathbf{u}} = \frac{N}{\alpha}$, which after substitution, gives us a condition equivalent to Equation B.2.

Note that the stationary condition given by Equation B.2 depends only on marginals, not the absolute value of the partition function. Moreover, applying a proper feasibility function $g(\theta)$, where $\theta_{\mathbf{x}_a} \propto g(\theta_{\mathbf{x}_a})\ \forall\ \mathbf{X}_a$, will not change the marginals implied by θ, as the multiplicative factors cancel out in each pair of terms, $\log Z_\theta - \log Z_\theta(\mathbf{d}_i)$. Thus if a point θ satisfies Equation B.2, then $g(\theta)$ must also satisfy it. Similarly, if $g(\theta)$ satisfies Equation B.2, the original point θ must also satisfy it. $\square$

Proof of Theorem 8 First, the partition function conditioned on an example $\mathbf{d}_i \in \mathcal{D}$ is:

$$Z_\theta(\mathbf{d}_i) = \sum_{\mathbf{x} \sim \mathbf{d}_i} \prod_{\mathbf{x}_a \sim \mathbf{x}} \theta_{\mathbf{x}_a}$$

where operator $\sim$ denotes compatibility between two instantiations (they set the same value to common variables). For a given parameter set $\theta_{\mathbf{X}_a}$, the partition function $Z_\theta(\mathbf{d}_i)$ is a linear function with respect to the parameters $\theta_{\mathbf{X}_a}$:

$$\begin{aligned} Z_\theta(\mathbf{d}_i) &= \sum_{\mathbf{x}_a} Z_\theta(\mathbf{x}_a, \mathbf{d}_i) = \sum_{\mathbf{x}_a} \frac{\partial Z_\theta(\mathbf{d}_i)}{\partial \theta_{\mathbf{x}_a}} \theta_{\mathbf{x}_a} \\ &= \sum_{\mathbf{x}_a} C^i_{\mathbf{x}_a} \cdot \theta_{\mathbf{x}_a} \end{aligned}$$

where $C^i_{\mathbf{x}_a}$ is a constant with respect to $\theta_{\mathbf{X}_a}$. Thus, our sub-function, has the form:

$$f_{\theta^\star}(\theta_{\mathbf{X}_a}) = -\sum_{i=1}^{N} \log \sum_{\mathbf{x}_a} C^i_{\mathbf{x}_a} \cdot \theta_{\mathbf{x}_a}.$$

On the other hand, the constraint $Z_\theta = \alpha$ takes the form:

$$Z_\theta = \sum_{\mathbf{x}_a} Z_\theta(\mathbf{x}_a) = \sum_{\mathbf{x}_a} \frac{\partial Z_\theta}{\partial \theta_{\mathbf{x}_a}} \theta_{\mathbf{x}_a} = \sum_{\mathbf{x}_a} C_{\mathbf{x}_a} \theta_{\mathbf{x}_a} = \alpha$$

□

Theorem 12 *Suppose we have a feasibility function*

$$g(y_1, \dots, y_n) = (x_1, \dots, x_n)$$

where $x_i \neq y_i$ implies that the point $(x_1, \dots, y_i, \dots, x_n)$ is infeasible (e.g., Euclidean projection satisfies this condition). Suppose now that the algorithm produces a sequence $x^t, y^{t+1}, x^{t+1} = x^t$. Then x^t must be a feasible and stationary point.

Proof By the statement of the iterative procedure, x^t is guaranteed to be feasible. Suppose that $g(y^{t+1}) = x^{t+1} = x^t$. First, it must be that $y^{t+1} = x^t$. Suppose instead that $y^{t+1} \neq x^t$, and thus for some component, $y^{t+1}_i \neq x^t_i$. By our feasibility function, $(x^t_1, \dots, y^{t+1}_i, \dots, x^t_n)$ must be infeasible. However, Step 2(a) ensures that $(x_1^t, \dots, y_i^{t+1}, \dots, x_n^t)$ is feasible. Hence, it must be that $y^{t+1} = x^t$. Further, by Step 2(a) and Claim 1, x^t must also be stationary. $\square$

APPENDIX C

Proofs of Chapter 6

Proof of Proposition 3 If $\mathbf{d}_i$ is an example of dataset $\mathcal{D}$, then $Pr_\theta(\mathbf{d}_i)$ does not depend on the parameters of variable X; see (Darwiche, 2009, Chapter 6). Hence, the likelihood function $L(\theta|\mathcal{D}) = \prod_{i=1}^{N} Pr_\theta(\mathbf{d}_i)$ does not depend on the parameters of variable X. $\square$

Proof of Theorem 9 Let $\mathbf{N} = \mathbf{S} \cup \mathbf{R}$ be all network variables. One can show that the product $\Theta^{\mathbf{d}}_{\mathbf{S}}\Theta^{\mathbf{d}}_{\mathbf{R}}$ is a parameter term for $\mathbf{N}$ and $\mathbf{d}$. Moreover, one can show that every parameter term for $\mathbf{N}$ and $\mathbf{d}$ can be written as $\Theta^{\mathbf{d}}_{\mathbf{S}}\Theta^{\mathbf{d}}_{\mathbf{R}}$. The key observation here is that if variable X is shared by some parameter in $\Theta^{\mathbf{d}}_{\mathbf{S}}$ and some parameter in $\Theta^{\mathbf{d}}_{\mathbf{R}}$, then $X \in \mathbf{O}$ and its value must be set by example $\mathbf{d}$. Hence, the parameters of $\Theta^{\mathbf{d}}_{\mathbf{S}}$ and those of $\Theta^{\mathbf{d}}_{\mathbf{R}}$ must be compatible. Hence, one can enumerate all parameter terms $\Theta^{\mathbf{d}}_{\mathbf{N}}$ by enumerating all products $\Theta^{\mathbf{d}}_{\mathbf{S}}\Theta^{\mathbf{d}}_{\mathbf{R}}$:

$$Pr_\theta(\mathbf{d}) = \sum_{\Theta^{\mathbf{d}}_{\mathbf{N}}} \Theta^{\mathbf{d}}_{\mathbf{N}} = \sum_{\Theta^{\mathbf{d}}_{\mathbf{S}}} \sum_{\Theta^{\mathbf{d}}_{\mathbf{R}}} \Theta^{\mathbf{d}}_{\mathbf{S}}\Theta^{\mathbf{d}}_{\mathbf{R}} = \left[\sum_{\Theta^{\mathbf{d}}_{\mathbf{S}}} \Theta^{\mathbf{d}}_{\mathbf{S}}\right] \left[\sum_{\Theta^{\mathbf{d}}_{\mathbf{R}}} \Theta^{\mathbf{d}}_{\mathbf{R}}\right].$$

$\square$

Proof of Theorem 10 By definition of a sub-network, $\mathbf{S}$ must be a component of $G|\mathbf{B}$. Hence, by Theorem 9, $L(\theta|\mathcal{D}) = L(\theta\!:\!\mathbf{S}|\mathcal{D})L(\theta\!:\!\mathbf{B}|\mathcal{D})$. Since $\mathbf{S}$ and $\mathbf{B}$ partition the variables of sub-network G, the parameters in $\theta\!:\!\mathbf{S}$ do not overlap with those in $\theta\!:\!\mathbf{B}$, and their union accounts for all sub-network parameters, θ. The theorem then follows immediately from Lemma 1. $\square$

C.1 Decomposing Stationary Points

A stationary point for function $f(x_1, \ldots, x_n)$ is a point $x_1^\star, \ldots, x_n^\star$ at which the gradient of $f(x_1, \ldots, x_n)$ evaluates to zero. That is,

$$\left.\frac{\partial f}{\partial x_i}\right|_{x_i = x_i^\star} = 0 \text{ for } i = 1, \ldots, n.$$

Lemma 1 *Consider the non-zero function*

$$f(x_1, \ldots, x_n, y_1, \ldots, y_m) = g(x_1, \ldots, x_n)h(y_1, \ldots, y_m).$$

Then $x_1^\star, \ldots, x_n^\star, y_1^\star, \ldots, y_m^\star$ is a stationary point of f iff $x_1^\star, \ldots, x_n^\star$ is a stationary point of g and $y_1^\star, \ldots, y_m^\star$ is a stationary point of h.

Proof Consider the following elementary identities:

$$\begin{aligned}
\frac{\partial f}{\partial x_i} &= g(x_1, \ldots, x_n)\frac{\partial h}{\partial x_i} + h(y_1, \ldots, y_m)\frac{\partial g}{\partial x_i} \\
&= h(y_1, \ldots, y_m)\frac{\partial g}{\partial x_i} \\
\frac{\partial f}{\partial y_i} &= g(x_1, \ldots, x_n)\frac{\partial h}{\partial y_i} + h(y_1, \ldots, y_m)\frac{\partial g}{\partial y_i} \\
&= g(x_1, \ldots, x_n)\frac{\partial h}{\partial y_i}.
\end{aligned}$$

The lemma follows immediately from these identities since function f is non-zero (which implies that g and h are non-zero). □

BIBLIOGRAPHY

Aitkin, M., & Aitkin, I. (1996). A hybrid EM/Gauss-Newton algorithm for maximum likelihood in mixture distributions. *Statistics and Computing*, *6*, 127–130.

Bache, K., & Lichman, M. (2013). Uci machine learning repository. Tech. rep., Irvine, CA: University of California, School of Information and Computer Science.

Bertsekas, D. P., & Tsitsiklis, J. N. (1989). *Parallel and Distributed Computation: Numerical Methods*. Prentice-Hall.

Besag, J. (1975). Statistical Analysis of Non-Lattice Data. *The Statistician*, *24*, 179–195.

Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent Dirichlet allocation. *JMLR*, *3*, 993–1022.

Boyd, S., & Vandenberghe, L. (2004). *Convex Optimization*. Cambridge University Press.

Chan, H., & Darwiche, A. (2005). On the revision of probabilistic beliefs using uncertain evidence. *AIJ*, *163*, 67–90.

Cherkassky, V., & Mulier, F. M. (2007). *Learning from Data: Concepts, Theory, and Methods*. Wiley-IEEE Press.

Chickering, G., & Heckerman (1995). Learning bayesian networks: Search methods and experimental results. In *Proceedings of the Fifth International Workshop on Artificial Intelligence and Statistics*, pp. 112–128.

Choi, A., & Darwiche, A. (2006). An edge deletion semantics for belief propagation and its practical impact on approximation quality. In *AAAI*, pp. 1107–1114.

Choi, A., Refaat, K. S., & Darwiche, A. (2011). EDML: A method for learning parameters in Bayesian networks. In *UAI*.

Darwiche, A. (2003). A differential approach to inference in Bayesian networks. *JACM*, *50*(3), 280–305.

Darwiche, A. (2009). *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press.

Darwiche, A., Dechter, R., Choi, A., Gogate, V., & Otten, L. (2008). Results from the probabilistic inference evaluation of uncertainty in artificial intelligence UAI-08. `http://graphmod.ics.uci.edu/uai08/Evaluation/Report`.

Dempster, A., Laird, N., & Rubin, D. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society B*, *39*, 1–38.

Domingos, P., & Lowd, D. (2009). *Markov Logic: An Interface Layer for Artificial Intelligence*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers.

Elidan, G., & Friedman, N. (2005). Learning hidden variable networks: The information bottleneck approach. *JMLR*, *6*, 81–127.

Elidan, G., & Gould, S. (2008). Learning bounded treewidth Bayesian networks. *JMLR*, *9*, 2699–2731.

Elidan, G., Ninio, M., Friedman, N., & Shuurmans, D. (2002). Data perturbation for escaping local maxima in learning. In *AAAI/IAAI*, pp. 132–139.

Fisher, R. A. (1922). On the mathematical foundations of theoretical statistics. *Philosophical Transactions of the Royal Society of London Series*, *222*.

Good, I. J. (1950). *Probability and the Weighing of Evidence*. Charles Griffin, London.

Heckerman, D. (1998). A tutorial on learning with Bayesian networks. In Jordan, M. I. (Ed.), *Learning in Graphical Models*, pp. 301–354. MIT Press.

Hestenes, M. R., & Stiefel, E. (1952). Methods of conjugate gradients for solving linear systems. *Research of the National Bureau of Standards*, *49*(6).

Hinton, G. (2000). Training products of experts by minimizing contrastive divergence. In *Neural Computation*.

Hyvarinen, A., & Dayan, P. (2005). Estimation of non-normalized statistical models using score matching. *JMLR*, *6*.

Jaimovich, A., Meshi, O., McGraw, I., & Elidan, G. (2010). Fastinf: An efficient approximate inference library. *The Journal of Machine Learning Research*, *11*.

Jiang, J., Rai, P., & III, H. D. (2011). Message-passing for approximate MAP inference with latent variables. In *NIPS*, pp. 1197–1205.

Jirousek, R., & Preucil, S. (1995). On the effective implementation of the iterative proportional fitting procedure. *Computational Statistics & Data Analysis*, *19*(2), 177–189.

Kindermann, R., & Snell, J. L. (1980). *Markov Random Fields and their Applications*. American Mathematical Society.

Koller, D., & Friedman, N. (2009). *Probabilistic Graphical Models: Principles and Techniques*. MIT Press.

Lafferty, J. D., McCallum, A., & Pereira, F. C. N. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data.. In *ICML*.

Lauritzen, S. (1995). The EM algorithm for graphical association models with missing data. *Computational Statistics and Data Analysis*, *19*, 191–201.

Li, S. Z. (2001). *Markov random field modeling in image analysis.* Springer-Verlag.

Lin, Y., & Druzdzel, M. (1997). Computational advantages of relevance reasoning in Bayesian belief networks. In *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence*.

Liu, D. C., & Nocedal, J. (1989). On the Limited Memory BFGS Method for Large Scale Optimization. *Mathematical Programming*, *45*(3), 503–528.

Liu, Q., & Ihler, A. T. (2011). Variational algorithms for marginal MAP. In *UAI*, pp. 453–462.

Marinari, E., Parisi, G., & Ruiz-Lorenzo, J. (1997). Numerical simulations of spin glass systems.. *Spin Glasses and Random Fields*, *12*.

Meng, Z., Wei, D., Wiesel, A., & III, A. O. H. (2013). Distributed learning of gaussian graphical models via marginal likelihoods. In *AIStats*.

Minka, T. P. (2001). Expectation propagation for approximate Bayesian inference. In *UAI*, pp. 362–369.

Minka, T. P., & Lafferty, J. D. (2002). Expectation-propogation for the generative aspect model. In *UAI*, pp. 352–359.

Mizrahi, Y. D., Denil, M., & de Freitas, N. (2014). Linear and parallel learning of markov random fields. In *In International Conference on Machine Learning (ICML)*.

Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective*. MIT Press.

Park, J., & Darwiche, A. (2004). A differential semantics for jointree algorithms. *AIJ*, *156*, 197–216.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers, Inc., San Mateo, California.

Pietra, S. D., Pietra, V. J. D., & Lafferty, J. D. (1997). Inducing features of random fields. *IEEE Trans. Pattern Anal. Mach. Intell.*, *19*(4), 380–393.

Refaat, K. S., Choi, A., & Darwiche, A. (2012). New advances and theoretical insights into EDML. In *UAI*, pp. 705–714.

Refaat, K. S., Choi, A., & Darwiche, A. (2013). EDML for learning parameters in directed and undirected graphical models. In *Advances in Neural Information Processing Systems 26*, pp. 1502–1510.

Refaat, K. S., Choi, A., & Darwiche, A. (2014). Decomposing parameter estimation problems. In *Advances in Neural Information Processing Systems 27*, pp. 1565–1573.

Refaat, K. S., & Darwiche, A. (2015). Data compression for learning mrf parameters. In *International Joint Conference on Artificial Intelligence*. To appear.

Roth, D. (1996). On the hardness of approximate reasoning. *Artificial Intelligence*, *82*.

Russel, S., Binder, J., Koller, D., & Kanazawa, K. (1995). Local learning in probabilistic networks with hidden variables. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*.

Salakhutdinov, R., Roweis, S. T., & Ghahramani, Z. (2003). Optimization with EM and expectation-conjugate-gradient. In *ICML*, pp. 672–679.

Shachter, R. (1986). Evaluating influence diagrams. *Operations Research*, *34*(6), 871–882.

Shachter, R. (1990). Evidence absorption and propagation through evidence reversals. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*.

Thiesson, B., Meek, C., & Heckerman, D. (2001). Accelerating EM for large databases. *Machine Learning*, *45*(3), 279–299.

Varin, C., Reid, N., & Firth, D. (2011). An overview of composite likelihood methods.. *Statistica Sinica*, *21*.

Yanover, C., Schueler-Furman, O., & Weiss, Y. (2007). Minimizing and learning energy functions for side-chain prediction.. *In Speed, Terry and Huang, Haiyan (eds.), Research in Computational Molecular Biology, volume 4453 of Lecture Notes in Computer Science*, *11*.