Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI 2015)

Data Compression for Learning MRF Parameters

Khaled S. Refaat and **Adnan Darwiche**
Computer Science Department
University of California, Los Angeles
{krefaat,darwiche}@cs.ucla.edu

Abstract

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.

1 Introduction

Markov Random Fields (MRFs) are probabilistic graphical models that have been useful to many fields, including computer vision, bioinformatics, natural language processing, and statistical physics; for example see [Li, 2001; Yanover *et al.*, 2007; Lafferty *et al.*, 2001; Marinari *et al.*, 1997]. An MRF represents a joint probability distribution compactly using a structure populated with parameters. In brief, 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 and Snell, 1980; Koller and Friedman, 2009; Murphy, 2012]. This paper is concerned with learning factors from incomplete data given a fixed structure.

Learning MRF parameters from data is typically reduced to finding the maximum likelihood parameters: ones that maximize the probability of a dataset, due to their attractive statistical properties [Fisher, 1922]. However, due to the complexity of learning maximum likelihood parameters, other simplified methods have also been proposed in literature such as pseudo-likelihood [Besag, 1975], ratio matching [Hyva rinen, 2005], composite maximum likelihood [Varin *et al.*, 2011], contrastive divergence [Hinton, 2000], and more recently the Linear and Parallel algorithm (LAP) [Mizrahi *et al.*, 2014].

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 the dataset, whereas in an incomplete dataset, some variables may have missing values. Computationally, learning from incomplete data can be much harder than learning from complete data, as we discuss next.

If the data is complete, learning maximum likelihood parameters can be formulated as a convex optimization problem. However, evaluating the objective or computing the gradient requires doing inference (i.e. summation over all possible variable instantiations) to compute the partition function, which is #P-hard [Roth, 1996]. Iterative algorithms, such as gradient descent [Russel *et al.*, 1995], conjugate gradient (CG) [Hestenes and Stiefel, 1952], limited-memory Broyden-Fletcher-Goldfarb-Shanno algorithm (L-BFGS) [Liu and Nocedal, 1989], iterative proportional fitting (IPF) [Jirousek and Preucil, 1995], and more recently the edge-deletion maximum-likelihood algorithm (EDML) [Refaat *et al.*, 2013], can be used to get the global optimal solution. Ideally, one inference is required per algorithm iteration, unless line search is used, which may require multiple function evaluations; for more about line search, see Chapter 9 in [Boyd and Vandenberghe, 2004].

On the other hand, if the data is incomplete, the optimization problem is generally non-convex, i.e. has multiple local optima. Iterative algorithms, such as expectation maximization (EM) [Dempster *et al.*, 1977; Lauritzen, 1995] and gradient descent, can be used to get a locally optimal solution; see Chapter 19 in [Murphy, 2012]. Unfortunately, non-convexity is not the only difficulty introduced by the data being incomplete. As pointed out in Koller and Friedman (2009) [Koller and Friedman, 2009], at every iteration, besides doing inference to compute the partition function, we now need to run inference separately conditioned on every unique data example. In this paper, 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 paper is organized as follows. In Section 2, we define our notation and give an introduction to the problem of learning MRF parameters. We motivate the problem we tackle in Section 3. In Section 4, we show how the data decomposition technique works. The experimental results are given in Section 5. We prove that our method is sound in Section 6. We review some of the related work in Section 7, and conclude in Section 8.

2 Learning Parameters

In this section, we define our notation, and review how parameter estimation for MRFs is formulated as an optimization problem. We use upper case letters (X) to denote variables and lower case letters (x) to denote their values. Variable sets are denoted by bold-face upper case letters ($\mathbf{X}$) and their instantiations by bold-face lower case letters ($\mathbf{x}$). 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 $\theta = (\dots, \theta_{\mathbf{X}_f}, \dots)$. 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 $\mathcal{D} = \{\dots, \mathbf{d}_i, \dots\}$ is denoted by $-\ell\ell(\theta|\mathcal{D})$, and given by:

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

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 paper, 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 1 is called the data term, whereas the second term is called the model term. If the data is complete, Equation 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 paper, 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.

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 consists of the data term and the model term, as explained in Section 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 paper, 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.

Not surprisingly, in today's world, with the growing use of low-cost computers and sensors for data collec-

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

tion [Cherkassky and Mulier, 2007], large datasets are widely available. Furthermore, it is not uncommon to have a significant number of variables that are always observed. For example, in the UCI repository: the internet advertisements database has 1558 variables, only 3 of which have missing values; the Automobile database has 26 variables, where 7 have missing values; the Dermatology database has 34 variables, where only age can be missing; the Mushroom dataset has 22 variables, where only one variable has missing values; and so on [Bache and Lichman, 2013]. Thus, data compression can potentially benefit a non-trivial spectrum of problems in different fields.

4 Data Decomposition

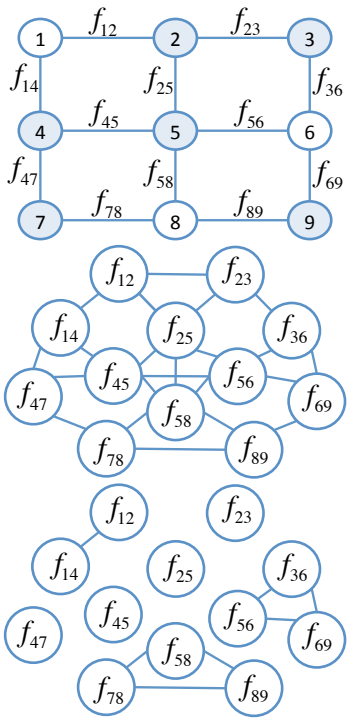

Figure 1: The process of identifying graph sub-networks given observed nodes: 2, 3, 4, 5, 7, and 9. **Top**: 3×3 MRF grid. **Middle**: A graph of factors, where an edge between two factors exists if they have common variables. **Bottom**: The sub-networks obtained by deleting every edge between two factors 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 6. Figure 1 (top) 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 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 1 (bottom).

Now, the decomposed factor graph in Figure 1 (bottom) 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 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 $2^2 \times 3^1 = 12$.

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

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 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.

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 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 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 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 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.

[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 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×

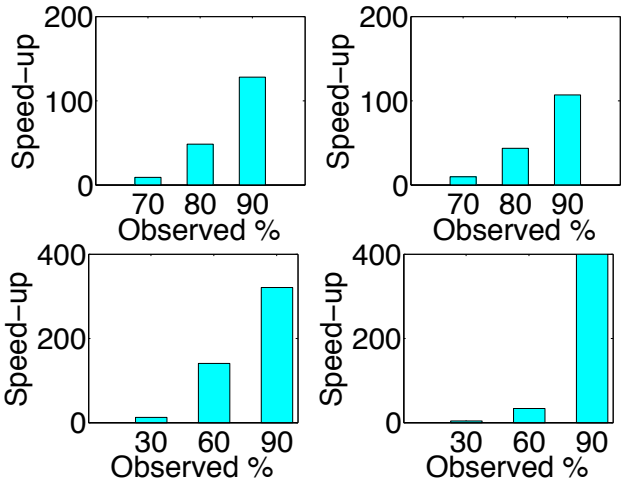

Figure 2: The speed-up obtained by data decomposition on different percentages of always observed variables, for 4 different network structures: 9x9 grid, alarm, chain (50 variables), tree (63 variables).

Figure 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.

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)

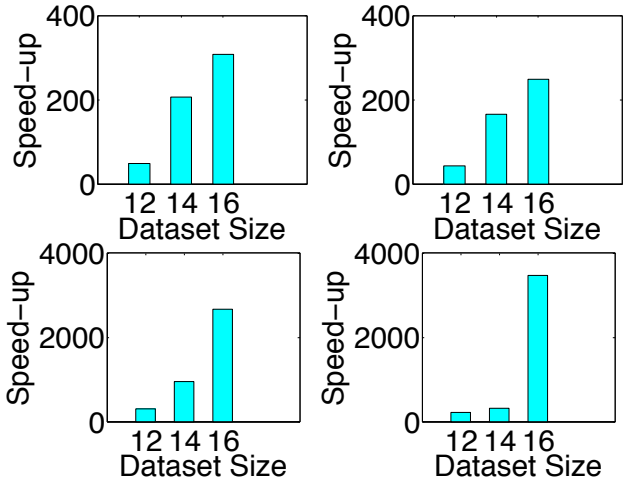

Figure 3: The speed-up obtained by data decomposition on different dataset sizes in log-scale, for 4 different network structures: 9x9 grid, alarm, chain (50 variables), tree (63 variables).

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 *et al.*, 2010]. We provided our system and FastInf with the same initial parameters, same incomplete datasets (hiding 20%), and the same MRF structure.

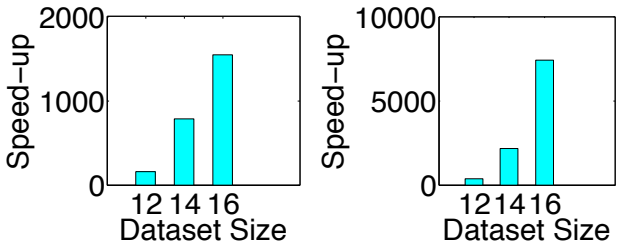

Figure 4: Speed-up of Gradient and EM methods (respectively) that use data decomposition, (allowed 100 iterations) over FastInf EM (allowed 2 iterations) on different dataset sizes in log-scale. Network alarm was used; 20% of the nodes have missing values in the data.

Figure 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 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.

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

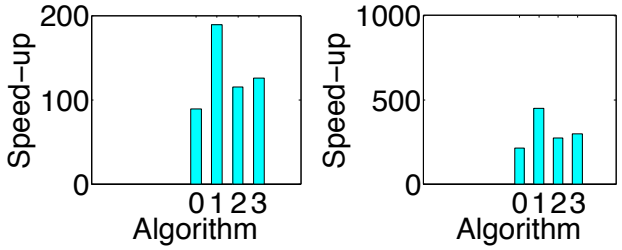

Figure 5: Speed-up of Gradient and EM methods, that use data decomposition, over different FastInf algorithms on 2^{12} data examples. Network alarm was used; 20% of the nodes have missing values in the data.

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 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.

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].

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 1 (Parameter Term) *Let* $\mathbf{F}$ *be a set of network factors and let* $\mathbf{d}$ *be a data example. A parameter term 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 $Z_\theta(\mathbf{d}) = \sum_{\Theta^{\mathbf{d}}_{\mathbf{F}}} \Theta^{\mathbf{d}}_{\mathbf{F}}$. Armed with parameter terms, we are now ready to state our decomposition theorem.

Theorem 1 *The data term is decomposed into a number of smaller functions corresponding to sub-networks. The log-likelihood takes the form:*

$$\ell\ell(\theta|\mathcal{D}) = \sum_s \sum_{i=1}^{N^s} n_i^s \log Z_\theta^s(\mathbf{d}_i) - N \log Z_\theta \tag{2}$$

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:

$$\sum_{i=1}^{N} \log Z_\theta(\mathbf{d}_i) = \sum_{i=1}^{N} \log \sum_{\Theta^{\mathbf{d}_i}_{\mathbf{F}}} \Theta^{\mathbf{d}_i}_{\mathbf{F}} \tag{3}$$

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

$$\sum_{i=1}^{N} \log Z_\theta(\mathbf{d}_i) = \sum_{i=1}^{N} \log \Big(\sum_{\Theta^{\mathbf{d}_i}_{\mathbf{F}^s}} \Theta^{\mathbf{d}_i}_{\mathbf{F}^s} \sum_{\Theta^{\mathbf{d}_i}_{\mathbf{F}^{s'}} \sim \Theta^{\mathbf{d}_i}_{\mathbf{F}^s}} \Theta^{\mathbf{d}_i}_{\mathbf{F}^{s'}}\Big) \tag{4}$$

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

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, $\Theta^{\mathbf{d}_i}_{\mathbf{F}^{s'}}$ 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:

$$\begin{aligned}\sum_{i=1}^{N} \log Z_\theta(\mathbf{d}_i) = \sum_{i=1}^{N} \log\Big(\sum_{\Theta^{\mathbf{d}_i}_{\mathbf{F}^s}} \Theta^{\mathbf{d}_i}_{\mathbf{F}^s} \sum_{\Theta^{\mathbf{d}_i}_{\mathbf{F}^{s'}}} \Theta^{\mathbf{d}_i}_{\mathbf{F}^{s'}}\Big) = \\ \sum_{i=1}^{N} \log \sum_{\Theta^{\mathbf{d}_i}_{\mathbf{F}^s}} \Theta^{\mathbf{d}_i}_{\mathbf{F}^s} + \sum_{i=1}^{N} \log \sum_{\Theta^{\mathbf{d}_i}_{\mathbf{F}^{s'}}} \Theta^{\mathbf{d}_i}_{\mathbf{F}^{s'}}\end{aligned} \tag{5}$$

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

$$\sum_{i=1}^{N} \log Z_\theta(\mathbf{d}_i) = \sum_{i=1}^{N^s} n_i^s \log \sum_{\Theta^{\mathbf{d}_i}_{\mathbf{F}^s}} \Theta^{\mathbf{d}_i}_{\mathbf{F}^s} + \sum_{i=1}^{N} \log \sum_{\Theta^{\mathbf{d}_i}_{\mathbf{F}^{s'}}} \Theta^{\mathbf{d}_i}_{\mathbf{F}^{s'}} \tag{6}$$

[5] In this case, not necessarily distinct.

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:

$$\sum_{i=1}^{N} \log Z_\theta(\mathbf{d}_i) = \sum_{i=1}^{N^s} n_i^s \log Z_\theta^s(\mathbf{d}_i) + \sum_{i=1}^{N} \log \sum_{\Theta^{\mathbf{d}_i}_{\mathbf{F}^{s'}}} \Theta^{\mathbf{d}_i}_{\mathbf{F}^{s'}} \tag{7}$$

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

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 paper, 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.

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 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

[6] Pruning edges migrates to MRFs.

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.

Acknowledgments

This work was supported by ONR grant #N00014-12-1-0423 and NSF grant #IIS-1118122.

References

[Bache and Lichman, 2013] K. Bache and M. Lichman. Uci machine learning repository. Technical report, Irvine, CA: University of California, School of Information and Computer Science, 2013.

[Besag, 1975] J. Besag. Statistical Analysis of Non-Lattice Data. *The Statistician*, 24:179–195, 1975.

[Boyd and Vandenberghe, 2004] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[Cherkassky and Mulier, 2007] Vladimir Cherkassky and Filip M. Mulier. *Learning from Data: Concepts, Theory, and Methods*. Wiley-IEEE Press, 2007.

[Darwiche, 2009] Adnan Darwiche. *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press, 2009.

[Dempster *et al.*, 1977] A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society B*, 39:1–38, 1977.

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

[Hestenes and Stiefel, 1952] Magnus R. Hestenes and Eduard Stiefel. Methods of conjugate gradients for solving linear systems. *Research of the National Bureau of Standards*, 1952.

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

[Hyva rinen, 2005] A Hyva rinen. Estimation of non-normalized statistical models using score matching. *The Journal of Machine Learning Research*, 2005.

[Jaimovich *et al.*, 2010] A. Jaimovich, O. Meshi, I. McGraw, and G. Elidan. Fastinf: An efficient approximate inference library. *The Journal of Machine Learning Research*, 2010.

[Jirousek and Preucil, 1995] Radim Jirousek and Stanislav Preucil. On the effective implementation of the iterative proportional fitting procedure. *Computational Statistics & Data Analysis*, 19(2):177–189, 1995.

[Kindermann and Snell, 1980] R. Kindermann and J. L. Snell. *Markov Random Fields and their Applications*. American Mathematical Society, 1980.

[Koller and Friedman, 2009] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.

[Lafferty *et al.*, 2001] J. D. Lafferty, A. McCallum, and F. C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML*, 2001.

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

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

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

[Marinari *et al.*, 1997] E. Marinari, G. Parisi, and J.J. Ruiz-Lorenzo. Numerical simulations of spin glass systems. *Spin Glasses and Random Fields*, 1997.

[Meng *et al.*, 2013] Z. Meng, D. Wei, A. Wiesel, and A. O. Hero III. Distributed learning of gaussian graphical models via marginal likelihoods. In *AIStats*, 2013.

[Mizrahi *et al.*, 2014] Yariv Dror Mizrahi, Misha Denil, and Nando de Freitas. Linear and parallel learning of markov random fields. In *In International Conference on Machine Learning (ICML)*, 2014.

[Murphy, 2012] Kevin Patrick Murphy. *Machine Learning: A Probabilistic Perspective*. MIT Press, 2012.

[Refaat *et al.*, 2013] Khaled S. Refaat, Arthur Choi, and Adnan Darwiche. EDML for learning parameters in directed and undirected graphical models. In *Advances in Neural Information Processing Systems 26*, pages 1502–1510, 2013.

[Refaat *et al.*, 2014] Khaled S. Refaat, Arthur Choi, and Adnan Darwiche. Decomposing parameter estimation problems. In *Advances in Neural Information Processing Systems 27*, pages 1565–1573, 2014.

[Roth, 1996] Dan Roth. On the hardness of approximate reasoning. *Artificial Intelligence*, 1996.

[Russel *et al.*, 1995] S. Russel, J. Binder, D. Koller, and K. Kanazawa. Local learning in probabilistic networks with hidden variables. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, 1995.

[Shachter, 1986] R. Shachter. Evaluating influence diagrams. *Operations Research*, 1986.

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

[Varin *et al.*, 2011] C. Varin, N. Reid, and D Firth. An overview of composite likelihood methods. *Statistica Sinica*, 2011.

[Yanover *et al.*, 2007] C. Yanover, O. Schueler-Furman, and Y. Weiss. 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*, 2007.