Graphical Models and Symmetries: Loopy Belief Propagation Approaches

Dissertation
zur
Erlangung des Doktorgrades (Dr. rer. nat.)
der
Mathematisch-Naturwissenschaftlichen Fakultät
der
Rheinischen Friedrich-Wilhelms-Universität Bonn

vorgelegt
von
Babak Ahmadi
aus
Teheran

Bonn, 2013

Angefertigt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Rheinischen Friedrich-Wilhelms-Universität Bonn

1. Gutachter: Prof. Dr. Stefan Wrobel
2. Gutachter: Prof. Dr. Kristian Kersting
Tag der Promotion: 09.07.2014
Erscheinungsjahr: 2014

Babak Ahmadi

Fraunhofer Institut für Intelligente Analyse-
und Informationssysteme IAIS

und

Rheinische Friedrich-Wilhelms-Universität Bonn,
Institut für Informatik III

Acknowledgements

First of all I wish to express my gratitude towards Prof. Dr. Stefan Wrobel and Prof. Dr. Kristian Kersting for providing me with the opportunity to work in the supportive environment of the Knowledge Discovery Department at Fraunhofer IAIS and the Computer Science Department at the University of Bonn.

Although I was the first PhD student in his group at Fraunhofer, or maybe because of this, I really enjoyed working with Kristian. Kristian is not only extremly knowledgable and passionate about what he does, but all his interest lies in the good of the group and his students. His micro- and macromanagement has been outstanding. Kristian has been the best possible match as an advisor and a friend you can always talk to.

I would like to thank all the current and former members of the STREAM group at KDD Ahmed Jawad, Fabian Hadiji, Marion Neumann, Martin Mladenov, Mirwaes Wahabzada, Salah Zayakh, and Zhao Xu, honorary member Novi Quadrianto, fellow Phd students in the KDD group, Anja Pilz, Christine Kopp, Daniel 'Gecko' Paurat, Dennis Wegener, Hannes Korte, Katrin Ullrich, Michael Kamp, Olana Missura, and Thomas Liebig, and other dear members of KDD Florian Schulz, Friederike von Rantzau, Mario Boley, Roman Garnett, and Thomas Gärtner for many interesting discussions, exchange of ideas and lots of fun "on and off the pitch".

Fabian Hadiji has spent a tremendous amount of time and effort to realize our implementation of libSTREAM which was used and extended throughout the work on this thesis. Martin Mladenov has taught me a lot in the course of his master's thesis work and is continuing to use his impressive mathematical understanding for a clear mathematical characterization of lifting (also outlined in the future work section) for his Phd. The discussions with both have tremendously helped to carve out and understand "all this lifted stuff".

I would also like to thank Scott Sanner, Marc Toussaint, and Luc De Raedt for giving me the the opportunity of not only visiting and making new experiences, but also for the support and valuable collaborations with them and their gifted students.

Special thanks go to Sriraam Natarajan, the advice and the joint projects have not only been fruitful but have greatly influenced me through the course of my research, and Tushar Khot for the valuable discussions and insights and for providing the Wumpus example.

This work would not have been possible without my co-authors Fabian Hadiji, Kristian Kersting, Marion Neumann, Martin Mladenov, Salah Zayakh, Scott Sanner, Sriraam Natarajan, and Youssef El Massaoudi.

Lastly, I would like to thank my family and friends for supporting me the last years and helping me to keep up the pace. You know who you are.

This work was supported by the Fraunhofer ATTRACT fellowship STREAM and by the European Commission under contract number FP7-248258-First-MM.

Abstract

Whenever a person or an automated system has to reason in uncertain domains, probability theory is necessary. Probabilistic graphical models allow us to build statistical models that capture complex dependencies between random variables. Inference in these models, however, can easily become intractable. Typical ways to address this scaling issue are inference by approximate message-passing, stochastic gradients, and MapReduce, among others. Exploiting the symmetries of graphical models, however, has not yet been considered for scaling statistical machine learning applications. One instance of graphical models that are inherently symmetric are statistical relational models. These have recently gained attraction within the machine learning and AI communities and combine probability theory with first-order logic, thereby allowing for an efficient representation of structured relational domains. The provided formalisms to compactly represent complex real-world domains enable us to effectively describe large problem instances. Inference within and training of graphical models, however, have not been able to keep pace with the increased representational power.

This thesis tackles two major aspects of graphical models and shows that both inference and training can indeed benefit from exploiting symmetries. It first deals with efficient inference exploiting symmetries in graphical models for various query types. We introduce lifted loopy belief propagation (lifted LBP), the first lifted parallel inference approach for relational as well as propositional graphical models. Lifted LBP can effectively speed up marginal inference, but cannot straightforwardly be applied to other types of queries. Thus we also demonstrate efficient lifted algorithms for MAP inference and higher order marginals, as well as the efficient handling of multiple inference tasks.

Then we turn to the training of graphical models and introduce the first lifted online training for relational models. Our training procedure and the MapReduce lifting for loopy belief propagation combine lifting with the traditional statistical approaches to scaling, thereby bridging the gap between statistical relational learning and traditional statistical machine learning.

Contents

List of Figures

List of Algorithms

1 Introduction

1.1 Statistical Relational AI

Machine learning thrives on large datasets and models, and one can anticipate substantial growth in the diversity and the scale of impact of machine learning applications over the coming decade. Such datasets and models originate, for example, from social networks and media, online books at Google, image collections at Flickr, and robots entering the real life. As storage capacity, computational power, and communication bandwidth continue to expand, today's ”large” is certainly tomorrow's ”medium” and next week's ”small”. This exciting new opportunity, however, also raises many challenges.

To be able to reason in uncertain domains our models need to deal with probabilities. Probabilistic graphical models, such as Bayesian networks or Markov networks, provide the tools to model the problems capture the dependencies of the variables involved and reason in that uncertain domain. Scaling inference within and training of graphical models, however, is a major challenge as the amount of data and the size of our problems grow. Thus, statistical learning provides a rich toolbox for scaling. Actually, statistical learning is unthinkable for many practical applications without such techniques. Exact inference, for example, is not applicable to many large-scale applications, therefore one often resorts to approximate inference by message-passing, which in turn can naturally be parallelized in a framework such as MapReduce. Training, i.e., learning the parameters of the model, let alone the structure, is even more challenging as it involves repeatedly carrying out inference. One approach out of the statistical machine learning toolbox to tackle training in large models is to do stochastic updates for gradient based approaches.

Often, however, we face inference and training problems with symmetries and redundancies in the graph structure. A prominent example are relational models, see [39, 53] for overviews, that tackle a long standing goal of AI, namely unifying first-order logic — capturing regularities and symmetries — and probability — capturing uncertainty. These models usually draw upon probabilistic graphical models and use a subset of first-order logic to describe relations within the domain of interest. They often encode large, complex models using few weighted rules only and, hence, symmetries and redundancies abound. One can easily define a model for millions of entities and relations among them. Since all the relations origin from a few rules with shared parameters we can expect substantial amounts of symmetry. Thus one major interest of statistical relational learning has been efficient model representation, inference and learning in the presence of symmetries (see [50] for an overview).

However, symmetries exist in propositional models as well. Our world is inherently symmetrical and, unlike humans who naturally make use of that knowledge, automated systems generally are not able to exploit this. To better understand which symmetries we refer to, consider the following example of a robot trying to set a table.

Example 1.1.1. *A household robot is confronted with the task of setting the table for a meal. The robot requires a lot of information to be able to autonomously complete this task. Some of the questions can be*

- *How many people are going to attend the meal?*
- *Is it going to be for breakfast, lunch or dinner?*
- *What is the required setup of the table?*
- *Which objects are needed?*
- *What sequence of actions has to be carried out to fulfill the task?*
- *and many more...*

Given all information the robot is able to plan and carry out the actions necessary to reach its goal. If there is information missing, however, it has to **infer** *what it wants to know from what it knows already. It could infer the type of the meal, i.e. breakfast, lunch or dinner, from the time of the day. It is unlikely to be dinner in the morning. In turn, the type of the meal determines the objects that are necessary. A cereal bowl is only needed for breakfast and not for lunch or dinner.*

The robot usually determines its action from some rules that have either been learned or given by the programmer in advance, in addition to some background knowledge about the world and measurements about its current state.

A set of rules stating that a cereal bowl is needed in case of a breakfast could look as follows:

$$
\begin{aligned}
\texttt{breakfast} &\rightarrow \texttt{NeedCerealBowl}\\
\texttt{NeedCerealBowl} &\wedge \texttt{InCupboard(CerealBowl\_1)}\\
&\rightarrow \texttt{Put(CerealBowl\_1)}, !\texttt{NeedCerealBowl}\\
\texttt{NeedCerealBowl} &\wedge \texttt{InCupboard(CerealBowl\_2)}\\
&\rightarrow \texttt{Put(CerealBowl\_2)}, !\texttt{NeedCerealBowl}\\
\texttt{NeedCerealBowl} &\wedge \texttt{InCupboard(CerealBowl\_3)}\\
&\rightarrow \texttt{Put(CerealBowl\_3)}, !\texttt{NeedCerealBowl}\\
\ldots
\end{aligned}
$$

The rules read as follows: if there is a need for a cereal bowl and cereal bowl number 1 *is in the cupboard, put cereal bowl* 1 *on the table. Now there is no need for a cereal bowl anymore (assuming we are setting up for one person). Applying these rules would be necessary to make sure at least one of the bowls is on the table. One can see that these rules look very much the same and thus the local structure is* **symmetric**. *It does not matter whether there is cereal bowl number* 1, 2 *or* 3 *on the table. A relational version could be stated as follows:*

$$
\begin{aligned}
\texttt{NeedCerealBowl} \wedge \exists X\ (\texttt{CerealBowl}(X) \wedge \texttt{InCupboard}(X))\\
\rightarrow \texttt{Put}(X), !\texttt{NeedCerealBowl}
\end{aligned}
$$

This quantified rule subsumes the above propositional rules and applies to all objects in the cupboard that are cereal bowls. Also, this rule nicely shows the symmetries inherent in the task. It only matters whether the objects are cereal bowls and if so, whether they are in the cupboard or not. Cereal bowls can also be placed in the dishwasher, for example, in which case they would not be available for the robot. With this task in mind, however, one does not need to distinguish between five or ten different bowls, only whether they are in the cupboard or not. In that sense they are indistinguishable and the model is symmetric. The above description also shows that the symmetries are not introduced by a relational representation. The symmetries have already been present in the propositional problem but are now only made explicit.

Reconsidering the above example, if we have to reason on the propositional level there are as many rules as there are cereal bowls. In the relational representation, on the other hand, there is only one rule for all cereal bowls. Ideally, we would like to carry out the reasoning as efficient as possible, i.e., on the smaller relational representation. The only distinction we would have to make in this case is between the group of bowls that are located in the cupboard and those that are not. This process of staying as abstract as possible, while getting to the propositional level (grounding) as much as necessary, is called **lifting**.

Therefore, symmetries have been explored in AI tasks, such as, symmetry-aware approaches in (mixed–)integer programming [18, 97], linear programming [64], SAT and CSP [142], and MDPs [44, 131]. Surprisingly, however, symmetries — stemming from

		Statistical Learning		
		single core	online	MapReduce
Loopy BP	standard	x	[1]	[59]
	lifted	**Ch. 3**	**Ch. 5**	**Ch. 8**
Training	standard	x	x	[172]
	lifted	-	-	-

Loopy BP	standard	x	[42]	[59]
	lifted	[146], **Ch. 3**	[116], **Ch. 5**	**Ch. 8**
Training	standard	x	[68]	**Ch. 7**
	lifted	**Ch. 7**	**Ch. 7**	**Ch. 7**
		single core	online	MapReduce
		Statistical Relational Learning		

Table 1.1: Machine learning thrives on large-scale datasets and models. Several approaches have been developed to scale traditional statistical learning such as approximate message passing, stochastic gradients, and MapReduce, among others (denoted by "x" if it is main stream or by a representative citation). Scaling by lifting, i.e., exploiting symmetries within the graphical model structure, however, has not received a lot of attention. In this thesis, we aim at closing this gap (denoted as "**Ch.**") in order to boost cross-fertilization. Please note that the references are naturally not exhaustive but representatives of the two worlds.

relational models or not — have not been the subject of great interest within statistical learning and for practical applications for inference.

Tab. 1.1 shows some approaches that have been developed to scale traditional statistical learning and statistical relational learning, respectively. For instance, (loopy) belief propagation [124] is an efficient approximate inference algorithm that has been proven successful in many real-world applications where exact inference is intractable. Gonzalez *et al.* [58, 59] have presented parallel versions for large factor graphs in shared memory as well as the distributed memory setting of computer clusters. Unfortunately,

(Limitation 1) *loopy belief propagation does not exploit symmetries.*

Indeed, for relational models, lifted loopy belief propagation has been proposed that exploits symmetries, see e.g. [146]. It often renders large, previously intractable probabilistic inference problems quickly solvable by employing symmetries to handle whole sets of indistinguishable random variables. However, symmetries are present in abundance even in traditional, non-relational models, as we have seen in Ex. 1.1.1. In Ch. 3 we will introduce *lifted LBP* that can be applied to relational as well as propositional models, as is also shown in Tab. 1.1. Moreover, in practical applications one is faced with a variety of different queries. Loopy belief propagation prescribes a way to compute (max-) marginal probabilities and joint probabilities for neighboring random variables, however, the range of different types of queries is very limited. Consequently,

(Limitation 2) *we can only benefit from lifting for marginal queries,* as it is not directly applicable to other forms. We widen the range and applicability of lifted inference and introduce lifted approaches for maximum a posteriori (MAP) queries and joint probabilities of distant nodes, tackled in Ch. 4 and Ch. 5 respectively, and a novel lifted message-passing technique that exploits symmetries across multiple evidence cases (Ch. 6). Furthermore, although we have increasing access to computer clusters due to the availability of affordable commodity hardware and high performance networking,

> **(Limitation 3)** *lifted loopy belief propagation is still carried out on a single core.*

That is, other than the parallel lifting introduced in Sec. 8.2 there are no inference approaches that exploit both symmetries and MapReduce for scaling. Still, in many if not most situations, training relational models will not benefit from scalable lifting.

> **(Limitation 4)** *Symmetries within models easily break since variables become correlated by virtue of asymmetrically depending on evidence*,

so that lifting produces models that are often not far from propositionalized, therefore canceling the benefits of lifting for training. In relational learning we typically face a single mega-example [101] only, a single large set of inter-connected facts. Consequently, many if not all standard statistical learning methods do not naturally carry over to the relational case. Consider, for example, stochastic gradient methods. Similar to the perceptron method [136], stochastic gradient approaches update the weight vector in an online setting. We essentially assume that the training examples are given one at a time. The algorithms examine the current training example and then update the parameter vector accordingly. Stochastic gradient approaches often scale sub-linearly with the amount of training data, making them very attractive for large training data as targeted by statistical relational learning. Empirically, they are even often found to be more resilient to errors made when approximating the gradient. Unfortunately, stochastic gradient methods do not naturally carry over to the relational cases:

> **(Limitation 5)** *Stochastic gradients coincide with batch gradients in the relational case since there is only a single mega-example.*

Consequently, while there are efficient parallelized gradient approaches, such as the one developed by Zinkevich *et al.* [172], that impose very little I/O overhead and are well suited for a MapReduce implementation, these have not been used for lifted training.

In this thesis, we demonstrate how to overcome all five limitations, that is, we fill in most gaps in Tab. 1.1, and thereby move statistical and statistical relational learning closer together in order to boost cross-fertilization.

1.2 Contributions and Outline of the Thesis

Message-passing algorithms, such as loopy belief propagation (LBP), can be very efficient in computing marginal distributions of random variables and joint marginals of neighboring nodes, thus often are the method of choice. However, we frequently encounter inference and training problems with symmetries and redundancies in the graph structure. A prominent example of such graphs are relational models. Exploiting these symmetries, however, has not been considered for scaling yet. These symmetries and redundancies arise from various sources, for example, from the graphical model structure or the task at hand and we show how to exploit these for various types of probabilistic queries. The thesis consists of two parts and makes a number of novel contributions to efficient inference in probabilistic graphical models for various query types (Part I) and to the training of relational models (Part II) by showing that both inference and training can indeed benefit from exploiting symmetries.

Part I

Ch. 3 As a *first contribution* we show that message-passing algorithms can be lifted. We introduce **lifted loopy belief propagation** that exploits symmetries and hence often scales much better than standard loopy belief propagation. Its underlying idea is rather simple: group together nodes that are indistinguishable in terms of messages sent and received given the evidence. The lifted graph is often significantly smaller and can be used to perform a modified loopy belief propagation yielding the same results as loopy belief propagation applied to the unlifted graph. This overcomes **Limitation 1**, and our experimental results show that considerable efficiency gains are obtainable. This mainly draws upon work that has previously been published as

> K. Kersting, B. Ahmadi, and S. Natarajan. Counting belief propagation. In *Proceedings of the Twenty–Fifth Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI–09)*, pages 277–284, Montreal, Canada, 2009. AUAI Press.

In many situations, this purely syntactic criterion of lifting, however, may yield lifted networks that are too pessimistic: lifting cannot make use of the fact that LBP message errors decay along paths [70] and one does not benefit from lifting in some practical applications. To overcome this, we propose **informed LLBP** a novel, easy-to-implement, informed LLBP approach that interleaves lifting and modified LBP iterations. In turn, we can efficiently monitor the true BP messages sent and received in each iteration and group nodes accordingly. This is based on the following work:

> K. Kersting, Y. El Massaoudi, B. Ahmadi, and F. Hadiji. Informed lifting for message–passing. In D. Poole M. Fox, editor, *Twenty–Fourth AAAI Conference on Artificial Intelligence (AAAI–10)*, Atlanta, USA, AAAI Press, 2010.

Furthermore, lifting is not limited to the discrete case. Many application domains involve continuous random variables and we show how the color-passing procedure for lifting can be applied to this case as well, that is, we introduce **lifted Gaussian belief propagation**, previously published in

> B. Ahmadi, K. Kersting, and S. Sanner. Multi-Evidence Lifted Message Passing, with Application to PageRank and the Kalman Filter. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI–2011)*, pages 11521–158, AAAI Press, 2011.

Ch. 4 The remainder of this part is dedicated to the various query types and how these can be handled efficiently in a lifted fashion, overcoming **Limitation 2**. Maximum a posteriori probability (MAP) queries, i.e., finding the most probable joint configuration of the random variables, are important in a number of fields and often formulated as linear programs (LP). We show how the symmetries carry over to the LP-relaxation and introduce **lifted linear programs**. The work in this chapter was previously published as

> M. Mladenov, B. Ahmadi, and K. Kersting. Lifted linear programming. In *15th International Conference on Artificial Intelligence and Statistics (AISTATS–2012)*, pages 788–797, La Palma, Canary Islands, Spain, 2012. Volume 22 of JMLR: W&CP 22.

Ch. 5 Often we are interested in computing higher order marginals, e.g., the joint distribution of distant nodes. This usually requires repeated inference with a slightly modified query. Lifting the network every time from scratch is wasteful and we show how to avoid this to efficiently perform sequential inference tasks to obtain **higher-order marginals**. This chapter mainly draws upon the following two works:

> B. Ahmadi, K. Kersting, and F. Hadiji. Lifted belief propagation: Pairwise marginals and beyond. In *Proceedings of the 5th European Workshop on Probabilistic Graphical Models (PGM10)*, pages 9–16, Helsinki, Finland, 2010.

> F. Hadiji, B. Ahmadi, and K. Kersting. Efficient sequential clamping for lifted message passing. In *Proceedings of the 34th Annual German Conference on Artificial Intelligence (KI–11)*, Berlin, 2011, Springer.

Ch. 6 Our *fifth main contribution* is a novel lifted message-passing technique that exploits symmetries across multiple evidence cases. The benefits of this **multi-evidence lifted inference** are shown for several important AI tasks such as computing personalized PageRanks and Kalman filters via multi-evidence lifted Gaussian belief propagation, previously published as

> B. Ahmadi, K. Kersting, and S. Sanner. Multi-Evidence Lifted Message Passing, with Application to PageRank and the Kalman Filter. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI–2011)*, pages 11521–158, AAAI Press, 2011.

Part II

Ch. 7 Still, in many if not most situations training relational models will not benefit from this (scalable) lifting: symmetries within models easily break since variables become correlated by virtue of asymmetrically depending on evidence. An appealing idea for such situations is to train and recombine local models. This breaks long-range dependencies and allows to exploit lifting within and across the local training tasks. Thus, in this chapter we develop the first **lifted training** approach. More specifically, we shatter a model into local pieces. In each iteration, we then train the pieces independently and re-combine the learned parameters from each piece. This breaks long-range dependencies and allows one to exploit lifting across the local training tasks. Hence, it overcomes **Limitation 4**.

Moreover, the shattering naturally paves the way for the first scalable lifted training approaches based on **stochastic gradients**. Based on the lifted piece-wise training, we introduce the first online training approach for relational models that can deal with partially observed training data. The idea is, we treat (mini-batches of) pieces as training examples and process one piece after the other. This overcomes **Limitation 5**. Our experimental results on several datasets demonstrate that the online training converges to the same quality solution over an order of magnitude faster than batch learning, simply because it starts optimizing long before having seen the entire mega-example even once. Indeed, the way we shatter the full model into pieces greatly effects the learning quality: important influences between variables might get broken. To overcome this, we randomly grow relational piece patterns that form trees. Our experimental results show that *tree pieces* can balance lifting and quality of the online training.

As has been shown before, general stochastic gradient approaches naturally extend to MapReduced training [172]. Lifted inference, however, has not been possible in a lifted fashion. We subsequently present the first **MapReduce lifted loopy belief propagation** approach. More precisely, we establish a link between color-passing, the specific way of lifting the graph, and radix sort, which is well-known to be MapReduce-able. Together with Gonzalez *et al.* [58, 59] MapReduce loopy belief propagation approach, this overcomes **Limitation 3**. Our experimental results show that MapReduced lifting scales much better than single-core lifting.

Moreover, this MapReduce lifting is the missing piece for a fully parallel lifted approach. We hereby bridge the gap between the two orthogonal approaches for scaling, namely lifting, i.e., exploiting symmetries in the model, and parallel computations.

The work in **Ch. 8.2** is based on the following two papers:

B. Ahmadi, K. Kersting, M. Mladenov, and S. Natarajan. Exploiting symmetries for scaling loopy belief propagation and relational training. *Machine Learning Journal*, 92(1): 91–132, July 2013.

B. Ahmadi, K. Kersting, and S. Natarajan. Lifted online training of relational models with stochastic gradient methods. In *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases (ECML–PKDD 2012)*, Bristol, UK, 2012. Springer.

Other work that is not covered in these chapters is listed in the references at the end of the thesis. A concluding chapter summarizes the thesis, discusses its implications and provides an outlook on future research.

1.3 Related Work

As argued above, in its two parts this thesis aims at developing efficient inference for probabilistic graphical models and efficient training approaches for relational models by employing symmetries in combination with MapReduce and stochastic gradients. Thus, we are getting mainstream statistical and statistical relational learning closer together. Consequently, there are several related lines of research.

Lifted Probabilistic Inference

In 2003, Poole published his seminal paper on “First-Order Probabilistic Inference” [126]. where he presents an algorithm to reason about multiple individuals where we may know particular facts about some of them. About others, however, we have no information. Thus, they are indistinguishable and we want to make use of this symmetry and treat them as a group.

This was the starting point of the very active research field trying to employ symmetries in graphical models for speeding up probabilistic inference, called **lifted probabilistic inference**, that has recently received a lot of attention.

Braz *et al.* [40, 41], Milch *et al.* [103], Kisynski and Poole [83], and Taghipour *et al.* [151] have developed lifted versions of the variable elimination (VE) algorithm. They typically employ a counting elimination operator that is equivalent to counting indistinguishable random variables and then summing them out immediately. The different variations of these algorithms improve upon the work of Poole [126] and Braz *et al.* [40, 41] by introducing counting formulas [103], aggregation operators [31, 83], handling arbitrary constraints [151, 152] and joint formulas [7]. Choi *et al.* [30] developed a variable elimination algorithm when the underlying distributions are continuous random variables and lifted variational inference for hybrid models [29]. These exact inference approaches do not easily scale to realistic domains, and hence have only been applied to rather small artificial problems.

The closest work to our approach for exploiting symmetries in approximate inference by message passing is the work by Singla and Domingos [146]. Singla and Domingos’ *lifted first-order belief propagation* (LFOBP) builds upon the work of Jaimovich *et al.* [72] and also groups random variables, i.e., nodes that send and receive identical messages. Given a Markov logic network LFOBP does so by grouping together nodes and factors that communicate via the same clauses in the network.

Sen *et al.* [143] presented another “clustered” inference approach based on bisimulation. Like lifted LBP, introduced later in Sec. 3, which can also be viewed as running a bisimulation-like algorithm on the factor graph, Sen *et al.*’s approach does not require a first-order logical specification. In contrast to lifted LBP, it is guaranteed to converge but is also much more complex. Its efficiency in dynamic relational domains, in which variables easily become correlated over time by virtue of sharing common influences in the past, is unclear and its evaluation is an interesting future work.

Recently, Van den Broeck *et al.* [155] proposed a method that relaxes first-order conditions to perform exact lifted inference on the model and then incrementally improve the approximation by adding more constraints back into the model. This can be viewed as bridging the lifted VE and the lifted LBP methods presented above. Again, as for LFOBP, a first-order logical specification of the model is required.

An alternative to LBP and VE is to use search-based methods based on recursive conditioning. That is, by conditioning on parameterized variables we decompose a lifted network into smaller networks that can be solved independently. Each of these subnetworks is then solved recursively using the same method, until we reach a simple enough network that can be solved [37]. Recently, several lifted search-based methods have been proposed [56, 57, 127, 157] that assume a relational model given. Gogate and Domingos [57] reduced the problem of lifted probabilistic inference to weighted model counting in a lifted graph. Van den Broeck *et al.* [157] employ circuits in first-order deterministic decomposable negation normal form to do the same, also for higher order marginals [156]. Both these approaches were developed in parallel and have promising potential to lifted inference. These approaches have been followed up by works providing a more detailed analysis leading to completeness results of the approaches [154] and the application to a wider range of problem such as maximum weighted utility (MEU) queries [8].

There are also sampling methods that employ ideas of lifting. Milch and Russell developed an MCMC approach where states are only partial descriptions of possible worlds [102]. Zettlemoyer *et al.* [170] extended particle filters to a logical setting. Gogate and Domingos introduced a lifted importance sampling [57], and Venugopal and Gogate introduced a lifted Gibbs sampling procedure [158].

Niepert proposed permutation groups and group theoretical algorithms to represent and manipulate symmetries in probabilistic models, which can be used for MCMC [119, 120]. And Bui *et al.* [24] have shown that for MAP inference we can basically divide the inference task into an evidence free and an evidence prone part, thus exploiting the symmetries of the model before evidence is incorporated.

It is an interesting question whether one can characterize these symmetries more precisely. Poon and Domingos identified the partition function as the key limiting factor for efficient inference and introduced an approach to obtain tractable networks [129]. Work on symmetry in exponential families [25] shows that one can create supernodes using the automorphism group of the graph, using the notion of orbit partitions. Mladenov and Kersting [106] analyze the space between the lifting obtainable by the different lifted approaches — exact and approximate — and show that there is a whole hierarchy of possible liftings.

As of today, none of these approaches have been shown to be MapReduce-able. Moreover, many of them have exponential costs in the treewidth of the graph, making them infeasible for most real-world applications, and none of them have been used for training relational models.

Symmetries in AI

Symmetries in general have been explored in many AI tasks. For instance, there are symmetry-aware approaches in (mixed–)integer programming [18, 97], linear programming [64], SAT and CSP [49, 142]. The general view in these domains is that, the cause of symmetries lies in the problem formulation, which in turn causes an exponential growth of the size of the search space. To cope with symmetrical CSPs, for example, there are different approaches such as adding global constraints that break the symmetry, or modifying the search either by pruning symmetric states or by using symmetry-breaking search heuristics.

Markov Decision Processes (MDPs) and partially observable MDPs (POMDPs) can easily grow to intractable problem sizes, thus there have been various approaches to first-order and relational versions that avoid propositionalizing the domain. Dean and Givan [44] make use of redundancies by grouping states that behave similar in a preprocessing step and thus reduce the size of the model. This work was later extended by Ravindran and Barto [131] to additionally include symmetries that would not have been exploited before. Boutillier *et al.* [20] introduced symbolic dynamic programming for first-order MDPs. Their algorithm solves first-order MDPs without explicit state space or action enumeration. Sanner and Kersting [138] introduced symbolic dynamic programming for POMDPs which was later extended to continuous states and observations by Zamani *et al.* [169]. Classical algorithms for MDPs have been developed for the first-order and relational case such as the first-order value iteration (FOVIA) [66] and the relational Bellman algorithm (ReBel) [79].

Surprisingly, symmetries have for a long time not been the subject of great interest within statistical learning and for practical applications for inference. As we have seen, besides exact inference using lifted variable elimination and its extensions, there is a lot of work on approximate inference which have shown successful in many AI tasks and applications, and only recently have some of these been applied in a lifted fashion and have shown to often be faster, more compact and provide more structure for optimization.

These tasks include information retrieval [117], satisfiability [61], Boolean model counting (Sec. 3.1), particle [145, 170] and Kalman filtering ([32] and Sec. 6.2), PageRank (Sec. 6.2), label propagation [117], citation matching, entity resolution, link prediction in social networks [134], information broadcasting (Sec. 3.2), and various sampling approaches [57, 102, 119, 120, 158], among others.

Local Training

Our lifted training approach is related to local training methods well known for propositional graphical models. Besag [12] presented a pseudolikelihood (PL) approach for training an Ising model with a rectangular array of variables. PL, however, tends to introduce a bias and is not necessarily a good approximation of the true likelihood with a smaller number of samples. In the limit, however, the maximum pseudolikelihood coincides with that of the true likelihood [165]. Hence, it is a very popular method for training models such as conditional random fields (CRF) where the normalization can become intractable while PL requires normalizing over only one node. An alternative approach is to decompose the factor graph into tractable subgraphs (or pieces) that are trained independently [149]. This *piecewise training* can be understood as approximating the exact likelihood using a propagation algorithm such as LBP. Sutton and McCallum [149] also combined the two ideas of PL and piecewise training to propose piecewise pseudolikelihood (PWPL) which, in spite of being a double approximation, has the benefit of being accurate like piecewise training and also scales well due to the use of PL. Another intuitive approach is to compute approximate marginal distributions using a global propagation algorithm like LBP, and simply substitute the resulting beliefs into the exact maximum likelihood gradient [94], which will result in approximate partial derivatives. Similarly, the beliefs can also be used by a sampling method such as Markov chain Monte-Carlo (MCMC) where the true marginals are approximated by running an MCMC algorithm for a few iterations. Such an approach is called contrastive divergence [65] and is popular for training CRFs.

Statistical Relational Learning

All the above training methods were originally developed for propositional data while real-world data is inherently noisy and relational. Statistical Relational Learning [39, 53] deals with uncertainty and relations among objects. The advantage of relational models is that they can succinctly represent probabilistic dependencies among the attributes of different related objects leading to a compact representation of learned models. While relational models are very expressive, learning them is a computationally intensive task. Recently, there have been some advances in learning SRL models, especially in the case of Markov Logic Networks [80, 84, 85, 96]. Algorithms based on functional-gradient boosting [50] have been developed for learning SRL models such as Relational Dependency Networks [114] and Markov Logic Networks [80]. Piecewise learning has also been pursued already in SRL. For instance, the work by Richardson and Domingos [134] used pseudolikelihood to approximate the joint distribution of MLNs which is inspired from the local training methods mentioned above. Although all of these methods exhibit good empirical performance, they apply the closed-world assumption, i.e., whatever is unobserved in the world is considered to be false. They cannot easily deal with missing information. To do so, algorithms based on classical EM [46] have been developed for ProbLog, CP-logic, PRISM, probabilistic relational models, Bayesian logic programs [52, 60, 77, 140, 153], among others, as well as gradient-based approaches for relational models with complex combining rules [71, 115]. Poon and Domingos [128] extended the approach of Lowd and Domingos [96] which is using a scaled conjugate gradient with preconditioner to handle missing data. All these approaches, however, assume a *batch* learning setting; they do not update the parameters until the entire data has been scanned. In the presence of large amounts of data such as relational data, the above methods can be wasteful. Stochastic gradient methods as considered in the present paper, on the other hand, are online and scale sub-linearly with the amount of training data, making them very attractive for large data sets. Only Huynh and Mooney [68] have recently studied online training of MLNs.

Here, training was posed as an online max margin optimization problem and a gradient for the dual was derived and solved using incremental-dual-ascent algorithms. Huynh and Mooney's approach, however, is orthogonal to our approach in that they do discriminative learning as opposed to generative learning. Also, they do not employ lifted inference for training and make the closed-world assumption. It would be interesting to see where the approaches can complement each other, e.g., by employing parallel lifted max-product belief propagation for the max margin computation.

Distributed Inference and Training

To scale probabilistic inference, Gonzalez *et al.* [58, 59] present algorithms for parallel inference on large factor graphs using loopy belief propagation in shared memory as well as the distributed memory setting of computer clusters. Although, Gonzalez *et al.* report on map-reducing lifted inference within MLNs, they actually assume the lifted network to be given, in which case lifted LBP only consists of the modified LBP equations. Piatkowski *et al.* [125] introduced a GPU-parallel loopy belief propagation approach for conditional random fields. For pairwise factors, the message update of loopy belief propagation is equivalent to a matrix-vector product. There is a large body of work for efficient matrix-vector operations (see e.g. [63]), as they are essential in a lot of applications, one of which we will touch upon later in this thesis, namely the PageRank.

Another avenue to address the scaling of SRL algorithms is the combination of SRL with database technology. The relational database research community and the commercial industry have optimized the products and algorithms for decades since it was first introduced in 1970 by Codd [33]. Recently the need for uncertainty in databases has emerged to develop the notions of probabilistic databases and how to efficiently answer queries therein [10, 27, 35, 91, 132, 139].

Niu *et al.* [121] considered the problem of scaling up ground inference and learning over factor graphs that are multiple terabytes in size. They achieve this using database technology with two key observations:

(i) Grounding the entire SRL model into a factor graph is seen as a relational database management system (RDBMS) join that is realized using a distributed RDBMS.

(ii) They make learning I/O bound by using a storage manager to efficiently run inference over factor graphs that are larger than main memory.

It remains a very interesting and exciting future work to implement our algorithms using this database technology.

Beedkar *et al.* [11] propose an approach for parallel inference in Markov logic networks. The authors introduce a parallel grounding scheme for the MLN based on importance sampling and use this partitioning for parallel probabilistic inference.

All these approaches, however, do not make use of symmetries in the graphical models. The only other approach combining symmetries and parallelism in statistical relational models is the work by Noessner *et al.* [122] exploiting symmetries in the integer linear program formulation for MAP inference.

2

Background

2.1 Relational Graphical Models

Logical models and probabilistic models capture important aspects of the real-world. Each of them, however, are not able to represent complex domains including logic and uncertainty on their own. Statistical relational models provide powerful formalisms to compactly represent these complex real-world domains. There exist various different formalisms that combine logic and probability, sometimes also referred to as the alphabet soup of SRL [53]. These models can be directed or undirected. Some models build upon probabilistic models and add the logical aspect, others build upon logical models and add uncertainty. Examples are probabilistic relational models (PRM) [53], Bayesian logic programs (BLP) [77], relational Bayesian networks (RBN) [71], relational dependency networks (RDN) [118], and many more (see [53] for an overview). They enable us to effectively represent and tackle large problem instances for which inference and training is increasingly challenging.

What they generally have in common is that they employ (a subset of) first-order logic, introduced in Sec. 2.1.1, to define the factors or potential functions of the graphical models' (Sec. 2.1.2) probability distribution. One of the most prominent examples of statistical relational models are Markov logic networks [134]. We will have a closer look on Markov Logic networks in Sec. 2.1.3, since these are the relational models on which we will demonstrate the algorithms and techniques throughout this thesis. However, we would like to note that the algorithms apply to other relational models as well.

2.1.1 First-Order Logic

In the following we introduce a function-free first-order logic. The main building blocks of first-order logic are constants, logical variables, and predicates.

A *constant* represents an entity in the domain of interest, e.g., a particular cereal bowl or the kitchen table in Ex. 1.1.1. A *variable* is a place holder for objects in the domain. In this thesis called *logical variables* to make a distinction to *random variables* or simply *variables* where the context allows. Variables and constants may be typed, in which case the scope of a variable is limited to constants within the domain that are of the same type.

Objects within the domain may have relations. The relations are named by *predicates*. "InCupboard(CerealBowl$_1$)", for example, is a relation with the predicate "InCupboard". The arity of this relation is 1, as it involves only one object. In case there are multiple cupboards a relation with arity of at least 2 is necessary to appropriately represent this domain. "InCupboard(CerealBowl$_3$,Cupboard$_1$)" would state that cereal bowl number three is placed in the cupboard number one. Let n be the arity of a predicate. A predicate represents a mapping from n-tuples of constants to $\{$True, False$\}$.

By applying a predicate to a tuple of *terms*, i.e., variables or constants, we obtain an *atomic formula* or *atom*. Formulas are recursively composed from atomic formulas using logical connectives and quantifiers.

- negation: $\neg F_1$ is a formula $\Leftrightarrow$ F_1 is a formula
- disjunction: $F_1 \vee F_2$ is a formula $\Leftrightarrow$ F_1 and F_2 are formulas
- conjunction: $F_1 \wedge F_2$ is a formula $\Leftrightarrow$ F_1 and F_2 are formulas
- existential quantification: $\exists x F_1$ is a formula $\Leftrightarrow$ x is a variable and F_1 is a formula
- universal quantification: $\forall x F_1$ is a formula $\Leftrightarrow$ x is a variable and F_1 is a formula

Formulas and atomic formulas that contain no variables are called *ground formulas* and *ground atoms* respectively.

A *theory* consists of a finite set of implicitly conjoined formulas. A convenient form for theories is the conjunctive normal form (CNF). In a CNF each formula is a disjunction of positive and negative atomic formulas, and all variables are implicitly universally quantified. Any theory can be transformed into this form.

Assuming that,

(i) there are no other objects than those represented by the constants (*domain closure assumption*) and

(ii) two different constants designate different objects (*unique name assumption*),

a *Herbrand interpretation* or *possible world* is an assignment of a truth value to all ground atoms.

A *substitution* $F_i[x/t]$ maps all appearances of the variable x in formula F_i to the term t, also denoted as $F_i\theta$, where $\theta = \{x \rightarrow t\}$. Multiple substitutions can conveniently be written as

$$\begin{aligned} F\theta_1\theta_2 &= F\{x_1 \rightarrow t_1\}\{x_2 \rightarrow t_2\} \\ &= F\{x_1 \rightarrow t_1, x_2 \rightarrow t_2\} \\ &= F\theta_{1,2} \ . \end{aligned}$$

A substitution is ground when it maps each variable to a constant term. Applying a ground substitution to a formula is called *grounding*.

An interpretation *satisfies* a ground formula if it evaluates to *true* under the given assignment. A formula containing variables is satisfied if all of its groundings are satisfied.

Given a theory, there are two possible inference tasks: *SAT* (*model generation*) and *theorem proving*. In SAT one is interested in the question, whether a theory is *satisfiable*, that is, whether there exists an assignment such that the theory evaluates to *true*. Theorem proving answers the question whether a formula F holds in all models of a given theory T which can be reduced to SAT by showing that $F \cup T$ is unsatisfiable.

A basic SAT solving approach is to employ the *resolution* rule for inference [135]. Given two clauses, the resolution rule produces a single clause entailed by them. This is repeated until the resolution rule cannot be applied any more. If the resulting theory is *false*, the initial theory is also *false*, otherwise it is satisfiable. Resolution can be applied to ground or first-order theories. Let us demonstrate resolution on a small example.

Example 2.1.1. *Assume we want to check whether the following theory about the efficiency and liftability of algorithms is satisfiable:*

$$\begin{aligned} &\texttt{Efficient}(X) \\ \neg&\texttt{Efficient}(X) \vee \texttt{Liftable}(X) \ . \end{aligned}$$

Lifted resolution allows us to derive a new clause from two clauses with complementary literals. The above theory thus entails the following formula:

$$\texttt{Liftable}(X) \ .$$

This clause is not trivially false. *Thus the given theory is satisfiable. If we were to perform ground resolution on the above theory, the resolution rule would have to be applied to all ground instances of this theory, for example,*

$$\begin{array}{l} \ \texttt{Efficient}(ve) \\ \neg\texttt{Efficient}(ve) \vee \texttt{Liftable}(ve) \\ \ \texttt{Efficient}(bp) \\ \neg\texttt{Efficient}(bp) \vee \texttt{Liftable}(bp) \\ \ \texttt{Efficient}(lp) \\ \neg\texttt{Efficient}(lp) \vee \texttt{Liftable}(lp) \\ \ \texttt{Efficient}(mdp) \\ \neg\texttt{Efficient}(mdp) \vee \texttt{Liftable}(mdp) \\ \vdots \end{array}$$

In this sense, a lifted resolution step is equivalent to many ground resolutions. This example also demonstrates how much more efficient lifted resolution can be compared to ground resolution. Instead of applying the resolution rule for the first-order clauses once, we would have had to apply it once for every possible grounding.

In this thesis we will show how this can be carried over to inference and training of graphical models which are introduced in the following.

2.1.2 Probabilistic Graphical Models

Let X_i be a discrete-valued random variable and x_i represent a possible realization of the random variable X_i from the set of all possible values called the range of X_i, denoted as *range(X_i)*. Similarly, we define the range of a set of n discrete-valued random variables

$$\mathbf{X} = (X_1, X_2, \dots, X_n)$$

to be

$$range(\mathbf{X}) = range(X_1) \times range(X_2) \times \ldots \times range(X_n)$$

and let $\mathbf{x}$ represent a possible realization of the set of random variables $\mathbf{X}$.

A joint probability distribution P is a function that maps the joint states to a real-valued positive number and sums up to one, i.e.,

(i) $P : range(\mathbf{X}) \rightarrow \mathbb{R}_{\geq 0}$

(ii) $\sum_{\mathbf{x}} P(\mathbf{x}) = 1$.

An important inference task is to compute *marginal probabilities*, that is, the conditional probability of a set of variables given the values of some others. To do this, we need to *sum out* the joint probability distribution with respect to all other variables. We say that the other random variables are being *eliminated*.

Example 2.1.2. *Given the joint probability distribution $P(A, B, C)$ we can obtain the marginal probability of the random variable A by summing out the others. We thus compute*

$$P(A) = \sum_B \sum_C P(A, B, C)$$

where the sum is over all possible states of B and C, respectively.

Graphical models represent the joint probability distribution by an underlying graph and thus facilitate a clear mathematical formulation as well as the use of graph theoretical concepts for the understanding of the probabilistic methods.

Graphical models can be directed like, e.g., *Bayesian networks*, or undirected such as *Markov networks* or *Markov random fields (MRF)*. The difference between directed and undirected models generally lies in their parametrization and their Markov properties like, for example, the *conditional independence assumptions*.

The notion of conditional independence and the Markov property are fundamental for graphical models. We denote two random variables X and Y conditionally independent given a third variable Z denoted as $X \perp\!\!\!\perp Y|Z$ if

$$P(X, Y|Z) = P(Y|Z) \cdot P(X|Z)$$

Here the joint probability of X and Y can be written as the product of the two marginal probabilities, that is, it can be factorized.

In graphical models the *Markov property* holds: every variable is conditionally independent of the remaining, given its direct neighbors (*the Markov blanket*), that is, for a given variable X_k it holds that,

$$P(X_k|\mathbf{X}\backslash X_k) = P(X_k|MB(X_k)) \ ,$$

where $MB(X_k)$ is the Markov blanket of variable X_k.

Both directed and undirected graphical models consist of two components. The structure of the model is represented by a graph G and a set

$$F = \{f_k(\mathbf{X}_k)\}_{k=1,\ldots,n}$$

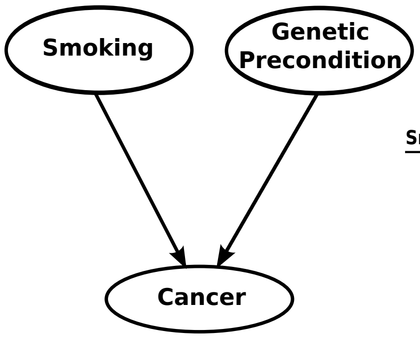
of *factor functions* which represent the *parametrization* of the model. Each factor f_k is a non-negative function of a subset of the variables $\mathbf{X}_k \subseteq \mathbf{X}$.

Probabilistic graphical models make use of the independencies in the joint distribution over $\mathbf{X}$ and compactly represent it as a product of factors [124], i.e.,

$$P(\mathbf{X} = \mathbf{x}) = \frac{1}{Z} \prod\nolimits_k f_k(\mathbf{x}_k) \ . \qquad (2.1)$$

Here, Z is a normalization constant ensuring that P is a probability distribution

$$\sum_{\mathbf{x}} P(\mathbf{x}) = 1 \ .$$



Smoking	P(Smoking)
yes	0.3
no	0.7

Precondition	P(Precondition)
risky	0.1
normal	0.9

Smoking	Precondition	Cancer	P(Cancer|Smoker,Precondition)
yes	risky	yes	0.30
yes	risky	no	0.70
yes	normal	yes	0.20
yes	normal	no	0.80
no	risky	yes	0.20
no	risky	no	0.80
no	normal	yes	0.05
no	normal	no	0.95

Figure 2.1: An example for a Bayesian network with associated CPTs.

The factorized representation of a probability distribution is generally much more efficient. Consider, for instance, a network of n binary variables with a maximum degree of d (maximum number of neighboring variables). A joint probability table would contain 2^n entries (or parameters) whereas the factorized distribution can be specified with $O(d \cdot n \cdot 2^d)$ entries, where usually $d \ll n$.

Bayesian networks

Bayesian networks [124] are one of the most popular and most widely used probabilistic graphical models. A Bayesian network $G(V, E)$ is a *directed acyclic graph (DAG)* where each node in V represents a random variable and an edge $e \in E \subseteq V \times V$ represents a conditional dependency between two nodes.

The subsets of random variables on which the *factors* are defined consist of the node X_i and its set of parents $Pa(X_i)$. This function is defined as the conditional probability of the nodes given its parents. Thus the joint probability distribution is defined as the product of the local conditional probabilities, that is,

$$P(\mathbf{X} = \mathbf{x}) = \prod_i P(X_i | Pa(X_i)) \ . \tag{2.2}$$

Example 2.1.3. *Fig. 2.1 shows an example for a Bayesian network. The Bayesian network shown on the left has three variables, namely* Smoking*, whether a certain* genetic precondition *is met and a variable* Cancer*. All three variables are binary and the CPTs are given on the right. Since* Cancer *depends on its parents, its CPTs has to include the two variables* Smoking *and* genetic precondition*. The joint distribution is now computed from the product of all the CPTs, that is,*

$$\begin{aligned}&P(Smoking, G.Precondition, Cancer)\\ &= P(Smoking)P(G.Precondition)P(Cancer|Smoking, G.Precondition) \ .\end{aligned}$$

Markov networks

Markov networks or *Markov random fields* [82, 124] are undirected graphical models. In a Markov network the structure $G(V, E)$ is an *undirected graph* where each node represents a random variable and an edge $e \in E \subseteq V \times V$ represents a conditional dependency between two nodes. The functions F are defined over the cliques of the graph.

$$P(\mathbf{X} = \mathbf{x}) = \frac{1}{Z} \prod_{i=1}^{|C|} f_i(\mathbf{x}_i) \ , \tag{2.3}$$

where C is the set of cliques associated with the graph and Z is a normalizing constant.

As long as $P(\mathbf{X} = \mathbf{x}) > 0$ for all joint configurations $\mathbf{x}$, the distribution can be equivalently represented as a log-linear model:

$$P(\mathbf{X} = \mathbf{x}) = \frac{1}{Z} \exp \left[\sum_i w_i \cdot \varphi_i(\mathbf{x}) \right] \ , \tag{2.4}$$

where the features $\varphi_i(x)$ are arbitrary functions of (a subset of) the configuration $\mathbf{x}$.

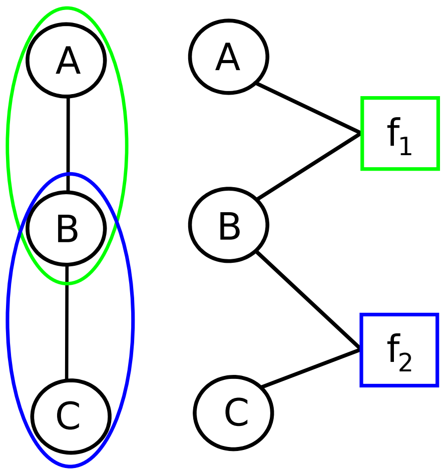
High-order MRFs can always be reduced to a pairwise one [51]. Without loss of generality, one often assumes a pairwise Markov random field. In a pairwise MRF the functions are defined over the edges of the graph, that is, the cliques of size two. The graphical representation alone, however, can be ambiguous. Assume we have a fully connected graph of size three, i.e., a Markov network with three nodes. In this graph the factors can be defined over the edges, three cliques of size two, or the whole network, i.e., one clique of size three. A factor graph is an equivalent representation of the Markov network that explicitly shows the factorization structure of the model.

Factor graphs

A factor graph $G(V_X, V_f, E)$ is a bipartite graph that expresses the factorization structure in Eq. (2.1). It explicitly shows the independencies of the set of variables $\mathbf{X}$ that lead to the factorization of the joint probability distribution. It has a variable node $v_{X_i} \in V_X$ (denoted as a circle) associated with each random variable X_i , $i = 1, \dots, n$, and a factor node $v_{f_k} \in V_f$ (denoted as a square) associated with each factor f_k , $k = 1, \dots, m$. An edge $e_{ij} \in E \subseteq V_X \times V_f$ connects variable node v_{X_i} to factor node v_{f_k} if and only if X_i is an argument of f_k. We will consider one factor $f_i(\mathbf{x}) = \exp[w_i \cdot \varphi_i(\mathbf{x})]$ per feature $\varphi_i(\mathbf{x})$, i.e., we will not aggregate factors over the same variables into a single factor.

Example 2.1.4. *Fig. 2.2 shows an example for Markov network (left) and its equivalent factor graph representation (center). The factor graph shown has three variable nodes and two factor nodes, i.e., $V_X = \{v_A, v_B, v_C\}$ and $V_f = \{v_{f_1}, v_{f_2}\}$. The joint distribution is factorized into the two factors f_1 and f_2, that is,*

$$P(A, B, C) = f_1(A, B) \cdot f_2(B, C)$$



A	B	f_1
True	True	1.2
True	False	1.4
False	True	2.0
False	False	0.4

C	B	f_2
True	True	1.2
True	False	1.4
False	True	2.0
False	False	0.4

Figure 2.2: An example for a Markov network **(left)** and the equivalent factor graph **(center)** with associated potentials **(right)**. Circles denote variables (binary in this case), squares denote factors.

The random variables are binary, thus the joint distribution can be fully specified by two tables of size four each, shown in Fig. 2.2 (right).

For more details on probabilistic graphical models we refer to [86].

2.1.3 Markov Logic Networks

A Markov logic network (MLN) F is defined by a set of first-order formulas (or clauses) F_i with associated weights w_i, $i \in \{1, \ldots, m\}$. The formulas are implicitly universally quantified. Together with a set of constants $C = \{C_1, C_2, \ldots, C_n\}$ F can be grounded, i.e., the free variables in the predicates of the formulas F_i are bound to be constants in C, to define a Markov network. This ground Markov network contains a binary node for each possible grounding of each predicate, and a feature for each grounding f_k of each formula. The joint probability distribution of an MLN is given by

$$P(\mathbf{X} = \mathbf{x}) = \frac{1}{Z} \exp\left(\sum_{i=1}^{|F|} \theta_i n_i(\mathbf{x})\right) \tag{2.5}$$

where for a given possible world $\mathbf{x}$, i.e., an assignment of all variables $\mathbf{X}$, $n_i(\mathbf{x})$ is the number of times the ith formula is evaluated $true$ and Z is a normalization constant.

Example 2.1.5. *An example of a Markov Logic network is depicted in Tab. 2.1* (**top**)*. It consist of two first order clauses, $F = \{F_1, F_2\}$ with associated weights. Together*

English	**First-Order Logic**	**Weight**
Smoking causes cancer	$\texttt{Smokes(x)} \Rightarrow \texttt{Cancer(x)}$	1.5
Friends have similar smoking habits	$\texttt{Friends(x, y)} \Rightarrow (\texttt{Smokes(x)} \Leftrightarrow \texttt{Smokes(y)})$	2.0

$1.5 : Smokes(Anna) \Rightarrow Cancer(Anna)$
$1.5 : Smokes(Bob) \Rightarrow Cancer(Bob)$
$2.0 : Friends(Anna, Bob) \Rightarrow (Smokes(Anna) \Leftrightarrow Smokes(Bob))$
$2.0 : Friends(Bob, Anna) \Rightarrow (Smokes(Bob) \Leftrightarrow Smokes(Anna))$

English	**First-Order Logic**	**Weight**
Buddies have similar smoking habits	$\texttt{Buddies(x, y)} \Rightarrow (\texttt{Smokes(x)} \Leftrightarrow \texttt{Smokes(y)})$	2.0

Table 2.1: (**Top**) Example of Markov logic network inspired by [146]. Free variables are implicitly universally quantified. (**Center**) Grounding of the MLN for constants *Anna* and *Bob*. (**Bottom**) Additional clause about similar smoking habits of buddies instead of friends.

with a set of constants, say Anna *and* Bob*, it can be grounded to the ground clauses depicted in Tab. 2.1 (***center***). This set of ground clauses now defines a Markov network and in turn a factor graph with binary variables and factors $f_1 - f_4$. Each ground atom, e.g.* Smokes(Anna)*, is represented by a variable node in the factor graph and each ground clause is represented by a factor node. There is an edge between a variable and a factor iff the ground atoms appears in the corresponding ground clause and*

$$f_i(x_{\{i\}}) = e^{w_i n_i(x)} ,$$

where $x_{\{i\}}$ are the truth values of the variables appearing in f_i, w_i the weight of the clause and $n_i(x)$ is the truth value of the clause, given the assignment x.
For example, if we know that Anna *smokes but does not have cancer, the implication in f_1 evaluates to false, thus*

$$f_1(Smokes(Anna) = True, Cancer(Anna) = False) = e^{1.5*0} = 1 \text{ and}$$
$$f_2(Smokes(Bob) = False, Cancer(Bob) = False) = e^{1.5*1} ,$$

assuming Bob *does not smoke nor has cancer and the implication in f_2 evaluates to true. Thus, given the current evidence E we have on* Anna *and* Bob*, for F_1 we have that*

$$F_1(E) = e^{1.5*1} .$$

Tab. 2.1 and Ex. 2.1.5 illustrate how, given a set of constants which form the *domain* we need to reason about, e.g., the persons in a social network or the words in a document, the free variables within the first-order rules can be bound to produce a propositional rule for every instance. These grounded networks are propositional and all propositional techniques may be applied.

Grounding the Friends-and-Smokers example from Tab. 2.1 (**top**) with two constants already produces a propositional factor graph with six variables and four factors (unary factors / priors neglected). As this small example already demonstrates, statistical relational models such as MLNs introduced here make it easy to define large and complicated models. Thus, statistical relational models enable us to tackle a broader class of applications and larger instances, however, they also pose major challenges for inference and training.

2.2 Graphical Model Inference and Training

2.2.1 Loopy Belief Propagation

As we have seen in Sec. 2.1.2, we can obtain the marginal probability of a variable by summing out the joint probability distribution with respect to all remaining variables, In many problems, however, we are interested in more than one or even all marginal probabilities. Although we could compute each separately, when we have a closer look at the computations we notice that in this case a lot of the intermediate results are shared.

Example 2.2.1. *Reconsider the example factor graph from 2.2. To obtain the marginal probabilities of the random variables we have to sum out all the remaining variables. We thus compute*

$$\begin{aligned} P(A) &= \sum_B \sum_C P(A,B,C) \\ &= \sum_B \sum_C f_1(A,B) f_2(B,C) \\ &= \sum_B f_1(A,B) \sum_C f_2(B,C) \end{aligned}$$

$$P(B) = \sum_A f_1(A,B) \sum_C f_2(B,C)$$

$$P(C) = \sum_B f_1(A,B) \sum_A f_2(B,C)$$

When computing *P(A)* and *P(B)* in Ex. 2.2.1 we have to sum out the variable C from factor f_2 in both cases. The intermediate terms that arise during the computation can also be viewed as messages sent along the edges of the factor graph. Instead of viewing the inference process as an elimination process this opens up a message-passing view to inference. Summing out C from f_2 then corresponds to a message sent to A. The operations for computing the marginals of the variables thus consist of sums and products of factor functions.

The belief propagation (BP) or sum-product algorithm is an efficient way to solve this problem. Belief propagation makes local computations only and makes use of the graphical structure such that the marginals can be computed much more efficiently. One should note that the problem of computing marginal probability functions is in general hard (#P-complete).

The computed marginal probability functions will be exact if the factor graph has no cycles, but the BP algorithm is still well-defined when the factor graph does have cycles. Although this loopy belief propagation (LBP) has no guarantees of convergence or of giving the correct result, in practice it often does, and can be much more efficient than

Algorithm 1: Loopy Belief Propagation (flooding)

```
Input: A factor graph $G = (V_X \cup V_f, E)$ and a convergence threshold.
Output: A set of approximate marginals $\widetilde{b}_{X_i}$, $i \in 1, \dots, n$.
1  $q \leftarrow 0$;
2  foreach edge $\{i, k\} \in E$ do
3      $\mu^q_{f_k \to X_i}(x) \leftarrow 1$;
4      $\mu^q_{X_i \to f_k}(x) \leftarrow 1$;
5  repeat
6      $q \leftarrow q + 1$;
7      foreach variable node $X_i \in V_X$ do
8          foreach factor node $f_k \in \mathrm{nb}(X_i)$ do
9              $\mu^q_{X_i \to f_k}(x_i) \leftarrow \prod_{f_j \in \mathrm{nb}(X_i) \setminus \{f_k\}} \mu^{q-1}_{f_j \to X_i}(x_i)$;
10     foreach factor node $f_k \in V_f$ do
11         foreach variable node $X_i \in \mathrm{nb}(f_k)$ do
12             $\mu^q_{f_k \to X_i}(x_i) \leftarrow \sum_{\mathbf{x}_k \setminus \{x_i\}} \left( f(\mathbf{x}_k) \prod_{X_j \in \mathrm{nb}(f_k) \setminus \{X_i\}} \mu^{q-1}_{X_j \to f_k}(x_j) \right)$;
13     $\epsilon \leftarrow \max_\mu |\mu^q - \mu^{q-1}|$;
14 until $\epsilon \leq$ convergence threshold;
15 return $\widetilde{b}_{X_i}(x) = \prod_{f_k \in \mathrm{nb}(X_i)} \mu^q_{f_k \to X_i}(x)$ ,  $i \in 1, \dots, n$;
```

other methods [112]. We will now describe the LBP algorithm in terms of operations on a factor graph.

To define the LBP algorithm, we first introduce messages between variable nodes and their neighboring factor nodes and vice versa. The message from a variable X to a factor f is

$$\mu_{X \to f}(x) = \prod_{h \in \mathrm{nb}(X) \setminus \{f\}} \mu_{h \to X}(x) \tag{2.6}$$

where $\mathrm{nb}(X)$ is the set of factors X appears in. The message from a factor to a variable is

$$\mu_{f \to X}(x) = \sum_{\neg\{X\}} \left(f(\mathbf{x}) \prod_{Y \in \mathrm{nb}(f) \setminus \{X\}} \mu_{Y \to f}(y) \right) \tag{2.7}$$

where $\mathrm{nb}(f)$ are the arguments of f, and the sum is over all the values of these except X, denoted as $\neg\{X\}$. The messages are usually initialized to 1.

Now, the unnormalized belief of each variable X_i can be computed from the equation

$$b_i(x_i) = \prod_{f \in \mathrm{nb}(X_i)} \mu_{f \to X_i}(x_i) \tag{2.8}$$

Evidence is incorporated by setting $f(\mathbf{x}) = 0$ for states $\mathbf{x}$ that are incompatible with it.

Different schedules may be used for message-passing. In the flooding protocol (synchronous) all messages are updated simultaneously, that is, to compute the message for the current iteration q only the messages from the previous iteration $q-1$ are used. When the messages are updated asynchronously we always use the latest message for the computation of new messages.

Alg. 1 summarizes the loopy belief propagation algorithm for the flooding schedule. Given a graph G all messages are initialized to 1 (**lines 2-4**). Then for each iteration q we send a message from each variable node to its neighboring factor nodes (**lines 7-9**) and into the other direction from the factor nodes to the variable nodes (**lines 11-12**). At convergence the belief of variable node X_i is obtained by the product of all incoming messages into the node (**line 15**). Convergence is reached if the belief or alternatively the message changes of two successive iterations falls below a specified threshold (**lines 13-14**).

Loopy belief propagation is an efficient algorithm for exact inference in trees and approximate inference for loopy graphs in cases where exact inference is not feasible. On loopy graphs it is not guaranteed to converge and one may use *damping* to facilitate convergence of some instances. In that case, the new message is taken to be a weighted average between the old message from the previous iteration and the new message according to the ordinary LBP update rule and is characterized by the damping parameter $\gamma \in [0,1]$.

Loopy belief propagation can be applied to propositional graphical models and statistical relational models that have been grounded.

2.2.2 Training of Graphical Models

The standard training task, also referred to as parameter learning or parameter estimation, for Markov networks can be formulated as follows. Given a set of training instances $D = \{D_1, D_2, \ldots D_M\}$ each consisting of an assignment to the variables in X, the goal is to output a parameter vector $\theta = (\theta_1, \ldots, \theta_m)$. To train the model, we can seek to maximize the log-likelihood function or equivalently minimize the negative log-likelihood function $-\log P(D \mid \theta)$ given by

$$\ell(\theta, D) = -\frac{1}{M} \sum\nolimits_D \log P_\theta(\mathbf{X} = \mathbf{x}_{D_i}) \; . \tag{2.9}$$

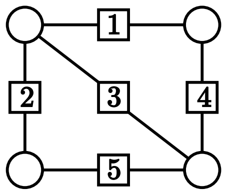
The likelihood, however, is computationally hard to obtain. A widely-used alternative is to maximize the pseudo-log-likelihood instead, i.e.,

$$\log P^*(\mathbf{X} = \mathbf{x} \mid \theta) = \sum\nolimits_{l=1}^{n} \log P_\theta(X_l = x_l | \mathit{MB}_x(X_l)) \tag{2.10}$$

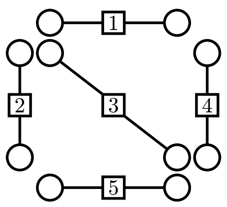
where $\mathit{MB}_x(X_l)$ is the state of the Markov blanket of X_l in the data, i.e., the assignment of all variables neighboring X_l. In this thesis, we minimize the negative log-likelihood (Eq.(2.9)).

An alternative approach is to decompose the factor graph into tractable subgraphs (or pieces) that are trained independently [149].

Example 2.2.2. *Consider the following example of a factor graph.*



It has four variable nodes with five pairwise factors connecting them. Inference in the joint models can become too complex and to make training tractable the global model can be approximated by many smaller local models. Intuitively, this cuts the dependencies between the factors. On the resulting model we can learn the parameters and apply them for our original model.



We thus decompose a factor graph into tractable but not necessarily disjoint subgraphs (or pieces) $\mathcal{P} = \{p_1, \ldots, p_k\}$. Training these pieces independently is a reasonable idea since in many applications, the local information in each factor alone is already enough to do well at predicting the outputs. The parameters learned locally are then used to perform global inference on the whole model.

More formally, at training time, each piece from $\mathcal{P} = \{p_1, \ldots, p_k\}$ has a local likelihood as if it were a separate graph, i.e., training example and the global likelihood is estimated by the sum of its pieces:

$$\hat{\ell}(\theta, D) = \sum\nolimits_{p_i \in \mathcal{P}} \ell(\theta|_{p_i}, D|_{p_i}) \; . \tag{2.11}$$

Here $\theta|_{p_i}$ denotes the parameter vector containing only the parameters appearing in piece p_i and $D|_{p_i}$ the evidence for variables appearing in the current piece p_i. The standard piecewise decomposition breaks the model into a separate piece for each factor.

No matter which objective function is used, one typically runs a gradient-descent to train the model. That is, we start with some initial parameters θ_0 — typically initialized to be zero or at random around zero — and update the parameter vector using

$$\theta_{t+1} = \theta_t - \eta_t \cdot g_t \; . \tag{2.12}$$

Here g_t denotes the gradient of the likelihood function and is given by:

$$\partial\ell(\theta, D)/\partial\theta_k = M\mathbf{E}_{\mathbf{x}\sim P_\theta}[n_k(\mathbf{x})] - n_k(D) \tag{2.13}$$

This gradient expression has a particularly intuitive form: the gradient attempts to make the feature counts in the empirical data equal to their expected counts relative to the learned model. Note that, to compute the expected feature counts, we must perform inference relative to the current model. This inference step must be performed at every step of the gradient process. In the case of partially observed data, we cannot simply read-off the feature counts in the empirical data and have to perform inference there as well. Consequently, there is a close interaction between the training approach and the inference method employed for training such as loopy belief propagation.

2.3 Algebra on Graphs

It is often useful to represent the graphs in terms of vectors and matrices. This enables us to derive and analyze the symmetries by using linear algebra.

Graphs can be encoded using the notion of an adjacency matrix showing for every pair of nodes whether they are adjacent, that is, whether they share an edge. More formally, the adjacency matrix of a graph $G = (V, E)$ with n vertices is an $n \times n$ matrix $\mathbf{A}(G)$ whose entries are

$$\mathbf{A}(G)_{uv} = \begin{cases} 1 & \text{if } \{u, v\} \in E \\ 0 & \text{otherwise.} \end{cases}$$

In this representation we do no make a distinction between the edges and thus we say that the edges have the same color. If we have additional information for our vertices and edges, however, we define a coloring (also called labeling) function

$$\textit{color} : V(G) \cup E(G) \to \mathbb{N} \ .$$

The entry $\mathbf{A}(G)_{uv}$ is then defined to be the color of the edge $\{u, v\}$ and $\mathbf{A}(G)_{uu}$ the color of the vertex u, represented as a natural number.

We consider two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ to be equivalent (*isomorphic*) iff there exists a bijection ϕ (called an *isomorphism*) mapping the vertices of G_1 to the vertices of G_2 such that

$$\{u, v\} \in E_1 \Leftrightarrow \{\phi(u), \phi(v)\} \in E_2 \ .$$

For colored graphs a further condition must hold, namely edge and vertex colors must be preserved, that is,

$$\textit{color}(u) = \textit{color}(\phi(u)) \quad \text{and} \quad \textit{color}(\{u, v\}) = \textit{color}(\{\phi(u), \phi(v)\}) \ ,$$

for all vertices u, v. Note that encoding colors as natural numbers is rather arbitrary and the same properties will hold over any field, thus, we might have chosen arbitrary rational or real numbers to represent colors, as we will do for continuous random variables such as in Sec. 3.3.

The problem of deciding if two graphs are isomorphic has not been shown to be NP-complete, yet no polynomial algorithm for the task is known [55].

We say that a graph G is *symmetric* whenever it is isomorphic to itself. More precisely, there exists an isomorphism $\sigma : V \to V$, which we call an *automorphism*, different from the identity.

The notion of graph automorphism also carries over to matrices. An $n \times n$ matrix $\mathbf{X}_\sigma$ represents an automorphism $\sigma : V \to V$ iff the following holds:

a-i) $\mathbf{X}_\sigma$ is a permutation matrix (i.e., $(\mathbf{X}_\sigma)_{ij} = 1 \Leftrightarrow \sigma(i) = j, 0$ otherwise) and

a-ii) $\mathbf{X}_\sigma$ commutes with $\mathbf{A}(G)$: $\mathbf{A}(G)\mathbf{X}_\sigma = \mathbf{X}_\sigma \mathbf{A}(G)$.

As discussed in [55], the set of matrices associated with the automorphisms of G, which we will call *Aut*(G), forms a group under matrix multiplication, that is, it satisfies four algebraic properties:

g-i) $\mathbf{X}_\sigma, \mathbf{X}_\pi \in \mathit{Aut}(G) \Rightarrow \mathbf{X}_\sigma \cdot \mathbf{X}_\pi \in \mathit{Aut}(G)$.

g-ii) $(\mathbf{X}_\sigma \cdot \mathbf{X}_\pi) \cdot \mathbf{X}_\tau = \mathbf{X}_\sigma \cdot (\mathbf{X}_\pi \cdot \mathbf{X}_\tau)$.

g-iii) The identity matrix $\mathbf{I}$ belongs to *Aut*(G) and it holds that for all other elements $\mathbf{X} \cdot \mathbf{I} = \mathbf{I} \cdot \mathbf{X} = \mathbf{X}$.

g-iv) For every element $\mathbf{X}$ of *Aut*(G), there exists an inverse $\mathbf{X}^{-1}$ such that $\mathbf{X} \cdot \mathbf{X}^{-1} = \mathbf{X}^{-1} \cdot \mathbf{X} = \mathbf{I}$.

These facts about automorphisms hold regardless of whether we talk about colored or uncolored graphs.

A division of the set of vertices V into k parts $\mathcal{U} = \{U_1, ..., U_k\}$ is called *partitioning*, if

(i) $\bigcup_{i=1}^{k} U_i = V$, the vertex set V is the union of the parts, and

(ii) $i \neq j \Rightarrow U_i \cap U_j = \emptyset$, the sets are distinct.

We say that u is equivalent to v iff u is mapped to v by some automorphism of G. This equivalence relation induces a partition of V, called the *orbit partition* of V [90]. The orbit partition of a graph can be encoded by characteristic matrices. Suppose a partition of a graph with n vertices has k parts. Then the entries of its characteristic matrix are given by $\mathbf{B} \in \{0,1\}^{n \times k}$

$$\mathbf{B}_{ij} = \begin{cases} 1 & \text{if vertex } i \text{ belongs to part } j \\ 0 & \text{otherwise.} \end{cases}$$

Knowing this partition can greatly help to efficiently reason in graphs, for example, since we know that the properties we might be interested in are identical within the equivalence classes. Unfortunately, enumerating the orbit partition, often referred to as graph stabilization, has the same computational complexity as deciding whether two graphs are isomorphic [98].

2.4 Linear Programming

As a very important example of how to benefit from indistinguishability, we will consider linear programs (LPs) and their symmetry in Ch. 4. A linear program in primal (equality-constrained) form $\mathcal{LP}_p = (\mathbf{A}, \mathbf{b}, \mathbf{c})$ is a mathematical program that can be specified in the following way:

$$\begin{aligned} \min_{\mathbf{y}\in\mathbb{R}^n} \quad & \mathbf{c}^T\mathbf{y} \\ \text{s.t.} \quad & \mathbf{A}\mathbf{y} = \mathbf{b} \\ & \mathbf{y} \geq 0, \end{aligned}$$

where the inequality and equality signs are to be understood as component-wise inequalities of real-valued vectors.

In other words, we search for a minimizer in $\mathbb{R}^n$ of the linear cost function $f(\mathbf{x}) = \mathbf{c}^T\mathbf{y}$, such that the system of m linear equalities over the n variables, given by $\mathbf{A}\mathbf{y} = \mathbf{b}$ is satisfied. The feasible set of a linear program is known from geometry as a polyhedron, or in case it is bounded, a polytope. Without loss of generality, we will assume from now on that the polyhedron is full-dimensional, i.e., the dimension of its affine hull is n. It is known that linear programs are solvable in polynomial time [21]. In fact, state-of-the-art interior point solvers have strong theoretical guarantees, while also showing fast empirical performance.

Linear programs may also be expressed by means of inequalities. A linear program in inequality (which we will also call *dual form*) form $\mathcal{LP}_d = (\mathbf{A}', \mathbf{b}', \mathbf{c}')$ is formulated as

$$\begin{aligned} \max_{\mathbf{x}\in\mathbb{R}^q} \quad & \mathbf{c}'^T\mathbf{x} \\ \text{s.t.} \quad & \mathbf{A}'\mathbf{x} \leq \mathbf{b}'. \end{aligned}$$

These two forms are equivalent, as one may express an equality-constrained in inequality-constrained form by introducing extra constraints, whereas the reverse can be achieved by the use of slack variables. There is a special relation between these two forms. Every primal LP, $\mathcal{LP} = (\mathbf{A}, \mathbf{b}, \mathbf{c})$ has a dual linear program $\overline{\mathcal{LP}} = (\mathbf{A}^T, \mathbf{c}, \mathbf{b})$ where strong duality holds, namely, if $\mathbf{y}^*$ and $\mathbf{x}^*$ are optima of both $\mathcal{LP}$ and $\overline{\mathcal{LP}}$, $\mathbf{c}^T\mathbf{y}^* = \mathbf{b}^T\mathbf{x}^*$.

2.5 MapReduce

The *MapReduce* [43] programming model allows parallel processing of massive data sets inspired by the functional programming primitives map and reduce and is basically divided into these two steps, the *Map*- and the *Reduce*-step.

1. In the *Map*-step the input is taken, divided into smaller sub-problems and then distributed to all worker nodes. These smaller sub problems are then solved by the nodes independently in parallel. Alternatively, the sub-problems can be further distributed to be solved in a hierarchical fashion.

2. The outputs of the Map phase have to be sorted and grouped before the subproblems can be passed on to the subsequent step.

3. In the *Reduce*-step all outputs of the sub-problems are collected and combined to form the output for the original problem.

Let us demonstrate the MapReduce principle on an example.

Example 2.5.1. *Assume we are given a weighted graph and would like to count how many edges have the same weight, that is, how many edges have the weight w_1, how many w_2, and so on. The graph is given as a colored adjacency matrix $\mathbf{A}$, where an edge between nodes n_i and n_j with the weight w is indicated by the entry $a_{ij} = w$.*

Let the graph be given by $\mathbf{A} = \begin{pmatrix} 0 & 0 & 3 \\ 1 & 0 & 2 \\ 3 & 0 & 0 \end{pmatrix}$

Now each row of the matrix $\mathbf{A}$ is an input for one mapper. All mappers can run in parallel. They process the input and output key-value pairs.

$map([0\ 0\ 3]) = 3 \rightarrow 1$
$map([1\ 0\ 2]) = 1 \rightarrow 1,\ 2 \rightarrow 1$
$map([3\ 0\ 0]) = 3 \rightarrow 1$

Here the first entry (the key) is the weight of the edge and the second entry (the value) is simply a 1. Before the key-value pairs are sent to the reducers, we group them by their keys:

$1 \rightarrow 1$
$2 \rightarrow 1$
$3 \rightarrow 1\ 1$

Now, each of the lines is passed to one of the reducers. The grouping (or sorting) was necessary to ensure that the pairs with the same keys are processed by the same reducer. The reducers typically perform some aggregation operation on the list of values, in this case the sum of the values. The result after reduction is:

$1 \rightarrow 1$
$2 \rightarrow 1$
$3 \rightarrow 2$

So there is one edge with the weight of 1, one has the weight of 2 and the weight of 3 appears twice in the graph.

Part I

Symmetries in Belief Propagation

In this part we will show how symmetries within graphical models can be used to speed up inference using loopy belief propagation (LBP). Given a probability distribution $P(\mathbf{X})$ defined by a graphical model there are numerous different practically relevant queries we might have to answer. Loopy belief propagation as a very efficient inference technique is well suited for inference in graphical models but it is not naïvely applicable to answer all different kinds of queries. Each query has specific challenges inherent in the inference problem it poses. Our focus in the subsequent chapters will be on efficiently solving the following inference tasks:

1. *Computing the marginal distributions $P(X_i|E)$ for a subset of nodes $X_i \in \mathbf{X_A}$, $\mathbf{X_A} \subseteq \mathbf{X}$ provided some evidence E (Ch. 3).*
2. *Finding a* MAP *assignment to a probability distribution P, i.e., computing $\operatorname{argmax}_{\mathbf{X}} P(\mathbf{X}|E)$ (Ch. 4).*
3. *Computing the joint marginal distribution $P(\mathbf{X}_A|E)$ of a subset $\mathbf{X}_A \subseteq \mathbf{X}$ of nodes provided some evidence E (Ch. 5).*
4. *Efficiently solving multiple inference tasks, i.e., computing the marginal distribution $P(X_i|E_j)$ for a subset of nodes $X_i \in \mathbf{X}_A$, $\mathbf{X}_A \subseteq \mathbf{X}$ and multiple evidence cases E_j , $j = 1, \ldots, m$ (Ch. 6).*

In each chapter we will name the specific challenges and address each of these in turn.

3 Marginal Inference

In this chapter we will tackle the task of marginal inference. More specifically, we

compute the marginal distributions $P(X_i|E)$ for a subset of nodes $X_i \in \mathbf{X_A}$, $\mathbf{X_A} \subseteq \mathbf{X}$ provided some evidence E.

LBP, as introduced in Sec. 2.2.1, quite efficiently approximates the marginals in general loopy graphs. Exploiting the symmetries, however, has so far been neglected as a dimension of scaling in the statistical machine learning approaches.

As the large body of work we have reviewed in Sec. 1.3 for lifted inference shows, this is very promising avenue for efficient probabilistic inference and has been mostly tackled for exact inference in relational models.

Exact inference approaches, however, do not scale to real-world problems and, more importantly, symmetries exist in abundance in propositional models as well. In the following we show how lifting can be achieved for approximate inference by message-passing. In particular, we show how to exploit symmetries for scaling inference in relational and propositional discrete models using *lifted LBP (LLBP)* (Sec. 3.1) and *informed LLBP (iLLBP)* (Sec. 3.2), and extend these results to inference in Gaussian models (Sec. 3.3).

3.1 Lifted Loopy Belief Propagation

Although already quite efficient, many graphical models produce inference problems with a lot of additional regularities in the graphical structure that are not exploited by LBP. Probabilistic graphical models such as MLNs are prominent examples. As an illustrative example, reconsider the factor graph in Fig. 2.2. The associated potentials are identical. In other words, although the factors involved are different on the surface, they actually share quite a lot of information. Standard LBP cannot make use of this information. In contrast, *lifted* LBP – which we will introduce in the following – can make use of it and can speed up inference by orders of magnitude.

The closest work to our approach for exploiting symmetries within message passing is the work by Singla and Domingos [146]. Singla and Domingos' *lifted first-order belief propagation* (LFOBP) builds upon the work of Jaimovich *et al.* [72] and also groups random variables, i.e., nodes that send and receive identical messages. Lifted LBP differs from LFOBP in that it does not require the specification of the probabilistic model in first-order logical format as input. For LFOBP only nodes over the same predicate can be grouped together to form so-called supernodes and it thus coincides with standard LBP for general factor graphs and propositional MLNs, i.e., MLNs involving propositional variables only. The reason is that propositions are predicates with arity 0 so that the supernodes are singletons. Hence, no nodes and no features are grouped together. In contrast, our lifting can directly be applied to any factor graph over finite random variables. In contrast to LFOBP that essentially groups together nodes that communicate via the same clauses, LLBP groups together nodes that communicate via identical potentials. In turn, LLBP can still yield significant efficiency gains when applied to Markov networks. In this sense, lifted LBP is a generalization of LFOBP, as will also be discussed shortly later.

Lifted LBP performs two steps: Given a factor graph G, it first computes a compressed factor graph $\mathfrak{G}$ and then runs a modified LBP on $\mathfrak{G}$. We will now discuss each step in turn using fraktur letters such as $\mathfrak{G}$, $\mathfrak{X}$, and $\mathfrak{f}$ to denote compressed graphs, nodes, and factors.

Step 1 – Compressing the Factor Graph:

To be able to compress the factor graph into a smaller network we need to determine the equivalent nodes and factors in the original graph. Thus, we essentially simulate LBP keeping track of which nodes and factors send the same messages, and group nodes and factors together correspondingly.

Let G be a given factor graph with Boolean variable and factor nodes. Initially, all variable nodes fall into three groups (one or two of these may be empty), namely known true, known false, and unknown. For ease of explanation, we will represent the groups by colored circles, say, magenta, green, and red. All factor nodes with the same associated potentials also fall into one group represented by colored squares. For the factor graph in Fig. 2.2 the situation is depicted in Fig. 3.1. As shown on the left-hand side, assuming no evidence, all variable nodes are unknown, i.e., red. Now, each variable node sends a message to its neighboring factor nodes saying "I am of color red". A factor

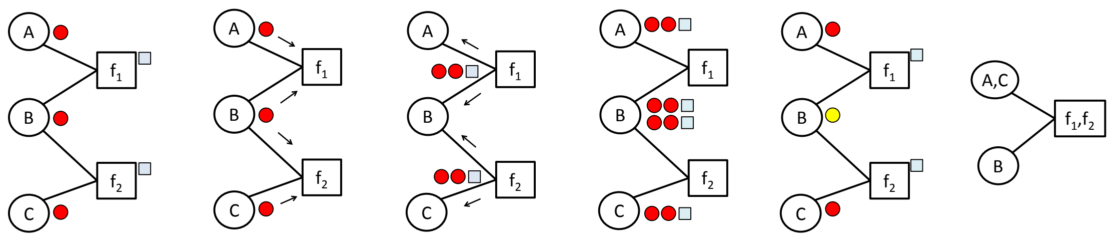


Figure 3.1: From left to right, the steps of CFG compressing the factor graph in Fig. 2.2 assuming no evidence. The shaded small circles and squares denote the groups and signatures produced running CFG. On the right-hand side, the resulting compressed factor graph is shown.

node sorts the incoming colors into a vector according to the order the variables appear in its arguments. The last entry of the vector is the factor node's own color, represented as light blue square in Fig. 3.1. This color signature is sent back to the neighboring variables nodes, essentially saying "I have communicated with these nodes". The variable nodes stack the incoming signatures together and, hence, form unique signatures of their one-step message history. Variable nodes with the same stacked signatures, i.e., message history can be grouped together. To indicate this, we assign a new color to each group. In our running example, only variable node *B* changes its color from red to yellow. The factors are grouped in a similar fashion based on the incoming color signatures of neighboring nodes. Finally, we iterate the process. As the effect of the evidence propagates through the factor graph, more groups are created. The process stops when no new colors are created anymore.

The final compressed factor graph $\mathfrak{G}$ is constructed by grouping all nodes with the same color into so-called *supernodes* and all factors with the same color signatures into so-called *superfactors*. In our case, variable nodes `A`, `C` and factor nodes f_1, f_2 are grouped together, see the right hand side of Fig. 3.1. Supernodes (resp. superfactors) are sets of nodes (resp. factors) that send and receive the same messages at each step of carrying out LBP on G. It is clear that they form a partition of the nodes in G.

Note that, we have to keep track of the position a variable appeared in a factor. Since factors in general are not commutative, it matters where the node appeared in the factor. Reconsider our example from Fig. 2.2, where

$$f_1(A = True, B = False) \neq f_1(A = False, B = True)$$

However, in cases where the position of the variables within the factor does not matter, one can gain additional lifting by sorting the colors within the factors' signatures. The ordering of the factors in the node signatures, on the other hand, should always be neglected, thus we perform a sort of the node color signatures.

Alg. 2 summarizes our approach for computing the compressed factor graph $\mathfrak{G}$ from a propositional factor graph G and given evidence E as input. Initially, the nodes have to be colored based on the evidence we have **(line 1)** and the factors based on their potentials **(line 2)**. Since nodes as well as factors have to be colored, their coloring

Algorithm 2: CFG – CompressFactorGraph

```
Data: A factor Graph $G$ with variable nodes $X$ and factors $f$, Evidence $E$
Result: Compressed Graph $\mathfrak{G}$ with supervariable nodes $\mathfrak{X}$ and superfactor nodes $\mathfrak{f}$
1  Compute initial clusters of the $X_i$s w.r.t. $E$;
2  Compute initial clusters of the $f_k$s w.r.t. their potential;
3  repeat
       // Form color signature for each factor
4      foreach factor $f_k$ do
5          $signature_{f_k} = [\,]$;
6          foreach node $X_i \in nb(f_k)$ do in order of appearance in $f_k$
7              $signature_{f_k}$.append($X_i.color$);
8          $signature_{f_k}$.append($f_k.color$);
9      Group together all $f_k$s having the same signature;
10     Assign each such cluster a unique color;
11     Set $f_k.color$ correspondingly for all $f_k$s;
       // Form color signature for each variable
12     foreach node $X_i \in X, i = 1, \dots, n$ do
13         $signature_{X_i} = [\,]$;
14         foreach factor $f_k \in nb(X_i)$ do
15             $signature_{X_i}$.append($f_k.color$);
16         sort $signature_{X_i}$ according to ordering given by $color$;
17         $signature_{X_i}$.append($X_i.color$);
18     Group together all $X_i$s having the same signature;
19     Assign each such cluster a unique color;
20     Set $X_i.color$ correspondingly for all $X_i$s;
21 until grouping does not change;
```

has to be carried out alternatingly until convergence. For each factor **(lines 4-11)** we first collect the colors of their neighboring nodes **(lines 6-7)** and form a signature by appending the factors own color **(line 8)**. All factors are now grouped based on the signatures **(line 9)** and assigned a new unique color **(line 10-11)**. The nodes are colored analogously **(lines 12-20)**. The only difference is that in the case of the nodes there is an additional sorting step **(line 16)** since the factors' position in the node signatures can be neglected. The coloring process is stopped when the grouping does not change **(line 21)**.

The clustering we do here, groups together nodes and factors that are indistinguishable given the loopy belief propagation computations. To better understand the color-passing and the resulting grouping of the nodes, it is useful to think of LBP and its operations in terms of its *computation tree* (CT), see e.g. [70]. The CT is the unrolling of the (loopy) graph structure where each level i corresponds to the i-th iteration of message passing. Similarly we can view the color-passing step (CP) within the lifting procedure as building a *colored computation tree* (CCT). More precisely, one considers for every node X the computation tree rooted in X but now each node in the tree is colored according to the nodes' initial colors, cf. Fig. 3.2**(left)**. For simplicity edge colors

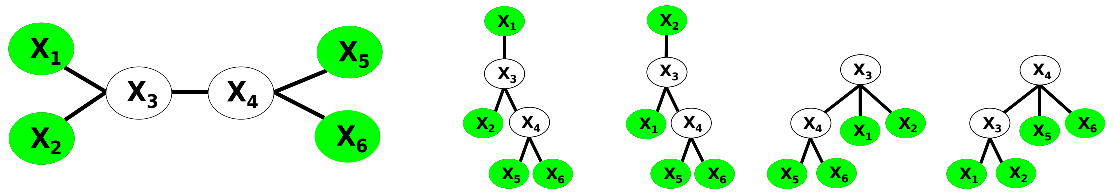


Figure 3.2: Original factor graph with colored nodes **(left)** and the corresponding colored computation trees for nodes X_1 to X_4 **(right)**. As one can see, nodes X_1 and X_2, respectively X_3 and X_4, have the same colored computation tree, thus are grouped together during color-passing.

are omitted and we assume that the potentials are the same on all edges. Each CCT encodes the root nodes' local communication patterns that show all the colored paths along which node X communicates in the network. Consequently, CP groups nodes with respect to their CCTs: nodes having the same set of rooted paths of colors (node and factor names neglected) are clustered together. For instance, Fig. 3.2**(right)** shows the CCTs for the nodes X_1 to X_4. Because their set of paths are the same, X_1 and X_2 are grouped into one supernode, X_3 and X_4 into another (together with X_5 and X_6 which are omitted here).[1]

Now we can run LBP with minor modifications on the compressed factor graph $\mathfrak{G}$.

Step 2 – LBP on the Compressed Factor Graph:

Recall that the basic idea is to simulate LBP carried out on G on $\mathfrak{G}$. An edge from a superfactor $\mathfrak{f}$ to a supernode $\mathfrak{X}$ in $\mathfrak{G}$ essentially represents multiple edges in G. Let $c(\mathfrak{f},\mathfrak{X},p)$ be the number of identical messages that would be sent from the factors in the superfactor $\mathfrak{f}$ to each node in the supernode $\mathfrak{X}$ that appears at position p in $\mathfrak{f}$ if LBP was carried out on G. The message from a supervariable $\mathfrak{X}$ to a superfactor $\mathfrak{f}$ at position p is

$$\mu_{\mathfrak{X}\to\mathfrak{f},p}(x) = \mu_{\mathfrak{f},p\to\mathfrak{X}}(x)^{c(\mathfrak{f},\mathfrak{X},p)-1} \prod_{\mathfrak{h}\in\mathrm{nb}(\mathfrak{X})} \prod_{\substack{q\in P(\mathfrak{h},\mathfrak{X})\\ (h,q)\neq(f,p)}} \mu_{\mathfrak{h},q\to\mathfrak{X}}(x)^{c(\mathfrak{h},\mathfrak{X},q)}\,, \tag{3.1}$$

where $\mathrm{nb}(\mathfrak{X})$ now denotes the neighbor relation in the compressed factor graph $\mathfrak{G}$ and $P(\mathfrak{h},\mathfrak{X})$ denotes the positions nodes from $\mathfrak{X}$ appear in $\mathfrak{f}$. The $c(\mathfrak{f},\mathfrak{X},p)-1$ exponent reflects the fact that a supervariable's message to a superfactor excludes the corresponding factor's message to the variable if LBP was carried out on G. The message from the factors to neighboring variables essentially remains unchanged. The difference is that we now only send one message per supernode and position, given by

$$\mu_{\mathfrak{f},p\to\mathfrak{X}}(x) = \sum_{\neg\{\mathfrak{X}\}} \left(\mathfrak{f}(\mathbf{x}) \prod_{\mathfrak{Y}\in\mathrm{nb}(\mathfrak{f})} \prod_{q\in P(\mathfrak{f},\mathfrak{Y})} \mu_{\mathfrak{Y}\to\mathfrak{f},q}(y)^{c(\mathfrak{f},\mathfrak{Y},q)-\delta_{\mathfrak{X}\mathfrak{Y}}\delta_{pq}} \right)\,, \tag{3.2}$$

[1]The partitioning of the nodes obtained by color-passing corresponds to the so-called coarsest equitable partition of the graph. We will briefly come back to this issue in Ch. 4.

where $\delta_{\mathfrak{X}\mathfrak{Y}}$ and δpq are one iff $\mathfrak{X} = \mathfrak{Y}$ and $p = q$ respectively. The unnormalized belief of $\mathfrak{X}_i$, i.e., of any node X in $\mathfrak{X}_i$ can be computed from the equation

$$b_i(x_i) = \prod_{\mathfrak{f} \in \mathrm{nb}(\mathfrak{X}_i)} \prod_{p \in P(\mathfrak{f}, \mathfrak{X}_i)} \mu_{\mathfrak{f},p \to \mathfrak{X}_i}(x_i)^{c(\mathfrak{f},\mathfrak{X},p)} . \tag{3.3}$$

Evidence is incorporated either on the ground level by setting $f(\mathbf{x}) = 0$ or on the lifted level by setting $\mathfrak{f}(\mathbf{x}) = 0$ for states $\mathbf{x}$ that are incompatible with it. [1] Again, different schedules may be used for message-passing. If there is no compression possible in the factor graph, i.e., there are no symmetries to exploit, there will be only a single position for a variable $\mathfrak{X}$ in factor $\mathfrak{f}$ and the counts $c(\mathfrak{f}, \mathfrak{X}, 1)$ will be 1. In this case the equations simplify to Eqs.(2.6)-(2.8).

To conclude the section, the following theorem states the correctness of *lifted* LBP.

Theorem 3.1.1. *Given a factor graph G, there exists a unique minimal compressed factor graph $\mathfrak{G}$, and algorithm* CFG(G) *returns it. Running LLBP on $\mathfrak{G}$ using Eqs.* (3.1) *and* (3.3) *produces the same results as LBP applied to G.*

The theorem generalizes the theorem of Singla and Domingos [146] but can essentially be proven along the same line. Although very similar in spirit, lifted LBP has one important advantage: not only can it be applied to first-order and relational probabilistic models, but also directly to traditional, i.e., propositional models such as Markov networks.

Proof. We prove the uniqueness of $\mathfrak{G}$ by contradiction. Suppose there are two minimal lifted networks $\mathfrak{G}_1$ and $\mathfrak{G}_2$. Then there exists a variable node X that is in supernode $\mathfrak{X}_1$ in $\mathfrak{G}_1$ and in supernode $\mathfrak{X}_2$ in $\mathfrak{G}_2$, $\mathfrak{X}_1 \neq \mathfrak{X}_2$; or similarly for some superfactor $\mathfrak{f}$. Since all nodes in $\mathfrak{X}_1$, and $\mathfrak{X}_2$ respectively, send and receive the same messages $\mathfrak{X}_1 = \mathfrak{X}_2$. Following the definition of supernodes, any pair of nodes $\mathfrak{X}$ and $\mathfrak{Y}$ in $\mathfrak{G}$ send and receive different messages, therefore no further grouping is possible. Hence, $\mathfrak{G}$ is a unique minimal compressed network.

Now we show that algorithm CFG(G) returns this minimal compressed network. The following arguments are made for the variable nodes in the graph, but can analogously be applied to factor nodes. Reconsider the colored computation trees (CCT) which resemble the paths along which each node communicates in the network. Variable nodes are being grouped if they send and receive the same messages. Thus nodes X_1 and X_2 are in they same supernode iff they have the same colored computation tree. Unfolding the computation tree to depth k gives the exact messages that the root node receives after k LBP iterations. CFG(G) finds exactly the similar CCTs. Initially all nodes are colored by the evidence we have, thus for iteration $k = 0$ we group all nodes that are similarly colored at the level k in the CCT. The signatures at iteration $k + 1$ consist of the signatures at depth k (the nodes own color in the previous iteration) and the colors of all direct neighbors. That is, at iteration $k + 1$ all nodes that have a similar CCT

[1]Note that, the variables have been grouped according to evidence and their local structure. Thus all factors within a superfactor are indistinguishable and we can set the states of the whole superfactor $\mathfrak{f}$ at once.

up to the $(k+1)$-th level are grouped. CFG(G) is iterated until the grouping does not change. The number of iterations is bounded by the longest path connecting two nodes in the graph. The proof that modified LBP applied to $\mathfrak{G}$ gives the same results as LBP applied to G also follows from CCTs, Eq.(3.1) and Eq.(3.2), and the count resembling the number of identical messages sent from the nodes in G. $\square$

In contrast to the color-passing procedure, Singla and Domingos [146] work on the relational representation and lift the Markov logic network in a top-down fashion. Color-passing on the other hand starts from the ground network and groups together nodes and factor bottom-up. While a top-down construction of the lifted network has the advantage of being more efficient for liftable relational models since the model does not need to be grounded, a bottom-up construction has the advantage that we do not rely on a relational model such as Markov logic networks. Color-passing can group ground instances of similar clauses and atoms even if they are named differently. Reconsider the two clause example from Tab.2.1**(top)**. Now, we add the clause from Tab.2.1**(bottom)** which has the same weight as the second clause and is similar in the structure, i.e., it has the same neighbors. Starting top-down, these two clauses would never be grouped by Singla and Domingos [146] whereas color-passing would initially give these clauses the same color. More importantly, as long as we can initially color the nodes and factors, based on the evidence and the potentials respectively, color-passing is applicable to all inference tasks for loopy belief propagation, for relational as well as propositional data such as Boolean formulae shown in the following.

Evaluation

Our intention here is to investigate the following questions:

(Q3.1) Can we scale inference in graphical models by exploiting symmetries?

(Q3.2) Can lifted inference help in propositional models at all?

To this aim, we implemented lifted loopy belief propagation (LLBP)in C++ based on libDAI[1] with bindings to Python. We will evaluate the gain for inference by presenting significant showcases for the application of lifted LBP, namely approximate inference for dynamic relational models and model counting of Boolean formulae.

Showing highly impressive lifting ratios for inference is commonly done by restricting to the classical symmetrical relational models without evidence. The two showcases for which we demonstrate lifted inference in the following, on the other hand, are particularly suited to not only show the benefits but also the shortcomings of lifted inference. Our first showcase, dynamic relational domains consists of long chains that destroy the indistinguishability of variables which might exist in a single time-step. Due to long chains, within and across time-steps, variables become correlated by virtue of sharing some common influence. The second showcase, the problem of model counting of Boolean formulae, as we will see later, is an iterative procedure that repeatedly

[1]http://cs.ru.nl/ jorism/libDAI/

English	**First-Order Logic**	**Weight**
Most people do not smoke	$\neg\texttt{Smokes}(\texttt{x})$	1.4
Most people do not have cancer	$\neg\texttt{Cancer}(\texttt{x})$	2.3
Most people are not friends	$\neg\texttt{Friends}(\texttt{x},\texttt{y})$	4.6
Smoking causes cancer	$\texttt{Smokes}(\texttt{x}) \Rightarrow \texttt{Cancer}(\texttt{x})$	2.0
Friends have similar smoking habits	$\texttt{Friends}(\texttt{x},\texttt{y}) \Rightarrow (\texttt{Smokes}(\texttt{x}) <=> \texttt{Smokes}(\texttt{y}))$	2.0
Apriori most people do not smoke	$\neg\texttt{Smokes}(\texttt{x},0)$	1.4
Apriori most people do not have cancer	$\neg\texttt{Cancer}(\texttt{x},0)$	2.3
Apriori most people are not friends	$\neg\texttt{Friends}(\texttt{x},\texttt{y},0)$	4.6
Smoking causes cancer	$\texttt{Smokes}(\texttt{x},\texttt{t}) \Rightarrow \texttt{Cancer}(\texttt{x},\texttt{t})$	2.0
Friends have similar smoking habits	$\texttt{Friends}(\texttt{x},\texttt{y},\texttt{t}) \Rightarrow (\texttt{Smokes}(\texttt{x},\texttt{t}) <=> \texttt{Smokes}(\texttt{y},\texttt{t}))$	2.0
Most friends stay friends	$\texttt{Friends}(\texttt{x},\texttt{y},\texttt{t}) \Leftrightarrow \texttt{Friends}(\texttt{x},\texttt{y},\texttt{succ}(\texttt{t}))$	5.0
Most smokers stay smokers	$\texttt{Smokes}(\texttt{x},\texttt{t}) \Leftrightarrow \texttt{Smokes}(\texttt{x},\texttt{succ}(\texttt{t}))$	5.0

Table 3.1: (**Top**) Example of a social network Markov logic network inspired by [146]. Free variables are implicitly universally quantified. (**Bottom**) Dynamic extension of the static social network model.

runs inference. In each iteration new asymmetrical evidence is introduced and lifted inference is run on the modified model. Both are very challenging tasks for inference and in particular for lifting. Thus, in this section, we already address random evidence and randomness in the graphical structure that are major issues for lifted inference and training, as we will learn in the following chapters.

Lifted Inference in Dynamic Relational Domains

Stochastic processes evolving over time are widespread. The truth values of relations depend on the time step t. For instance, a smoker may quit smoking tomorrow. Therefore, we extend MLNs by allowing the modeling of time. The resulting framework is called dynamic MLNs (DMLNs). Specifically, we introduce *fluents*, a special form of predicates whose last argument is time.

Here, we focus on discrete time processes, i.e., the time argument is non-negative integer valued. Furthermore, we assume a successor function $\texttt{succ}(\texttt{t})$, which maps the integer $\texttt{t}$ to $\texttt{t}+1$. There are two kinds of formulas: *intra-time* and *inter-time* ones. Intra-time formulas specify dependencies within a time slice and, hence, do not involve the succ function. To enforce the Markov assumption, each term in the formula is restricted to at most one application of the succ function, i.e., terms such as $\texttt{succ}(\texttt{succ}(\texttt{t}))$ are disallowed. A dynamic MLN is now a set of weighted intra- and inter-time formulas. Given the domain constants, in particular the time range $0,\ldots,T_{\max}$ of interest, a DMLN induces an MLN and in turn a Markov network over time.

As an example consider the social network DMLN shown in Tab. 3.1 (**Bottom**). The first three clauses encode the initial distribution at $\texttt{t}=0$. The next two clauses are intra-time clauses that talk about the relationships that exist within a single time-step. They say that smoking causes cancer and that friends have similar smoking habits. Of course, these are not hard clauses as with the case of first-order logic. The weights presented in the right column serve as soft-constraints for the clauses. The last two clauses are the inter-time clauses and talk about friends and smoking habits persisting over time.

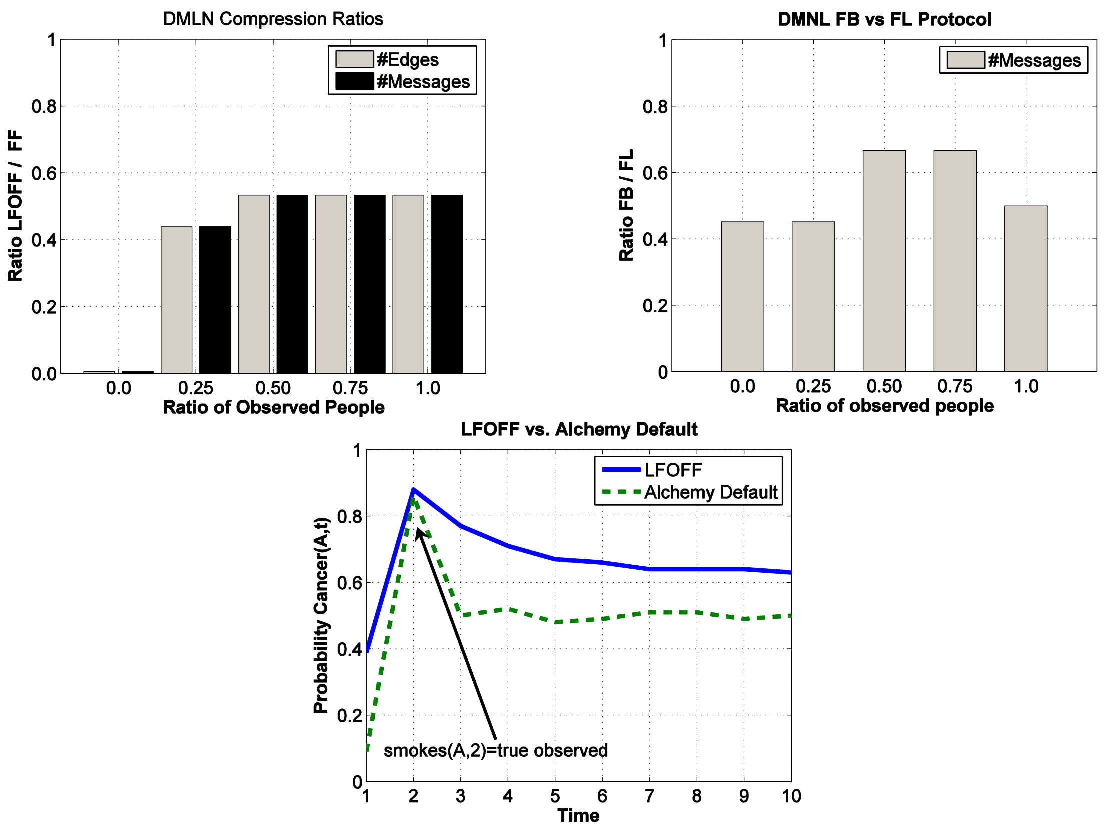


Figure 3.3: (**Left**) Ratios (LFOFF / FF) of number of edges and messages computed. The lower the value, the greater the speed-up when using LFOFF in place of FF. (**Right**) Ratios (Forwards-Backwards / Flooding protocol) of number of messages computed. The lower the value, the greater the speed-up when using the FB protocol in place of the FL protocol. (**Bottom**) Probability estimates for $\mathtt{cancer(A, t)}$ over time.

Assume that there are two constants Anna and Bob. Let us say that Bob smokes at time 0 and he is friends with Anna. Then the ground Markov network will have a clique corresponding to the first two clauses for every time-step starting from 0. There will also be edges between $\mathtt{Smokes(Bob)}$ (correspondingly Anna) an between the $\mathtt{Friends(Bob, Anna)}$ for consecutive time-steps.

To perform inference, we could employ any known MLN inference algorithm. Unlike the case for static MLNs, however, we need approximation even for sparse models: Random variables easily become correlated over time by virtue of sharing common influences in the past.

Classical approaches to perform approximate inference in dynamic Bayesian networks (DBN) are the Boyen-Koller (BK) algorithm [22] and Murphy and Weiss's factored frontier (FF) algorithm [111]. Both approaches have been shown to be equivalent to one iteration of LBP but on different graphs [111]. BK, however, involves exact inference, which for probabilistic logic models is extremely complex, so far does not scale to realistic domains, and hence has only been applied to rather small artificial problems. In contrast, FF is a more aggressive approximation. It is equivalent to LBP on the regular factor graph with a *forwards-backwards* message protocol: each node first sends message from “left” to “right” and then sends messages from ”right” to ”left”. Therefore a *frontier set* is maintained. Starting from time-step $t = 0$ we first send local messages then messages to the next time-step. A node is included in the *frontier set* iff all of its parents, that is, all neighbors from time-step $t-1$, and its neighbors from the same time-step are included. Only then it receives a message from its neighbors. The basic idea of lifted first-order factored frontier (LFOFF) is to *plug in lifted LBP in place of LBP* in FF. The local structure is replicated for each time-step in the dynamic network. Thus, the initial coloring of the nodes is the same for all time-steps. However, the communication patterns of the instantiation from different time-steps are different. Therefore, when we compress such a dynamic network nodes and factors from different time-steps end up being in different supernodes and superfactors respectively. To see this, suppose we have network consisting of a single node over three time-steps X^t, $t \in \{0, 1, 2\}$, i.e., a chain of length three. Initially all nodes get the same color. However, the signatures are different. Node X^0 only has a right neighbor, node X^1 has two neighbors (left and right) and node X^2 has a right neighbor. The *frontier set* for the lifted network is still well defined and we can run the lifted factored frontier algorithm.

We used the social network DMLN in Tab. 3.1 (**Bottom**). There were 20 people in the domain. For fractions $r \in \{0.0, 0.25, 0.5, 0.75, 1.0\}$ of people we randomly choose whether they smoke or not and who 5 of their friends are, and randomly assigned a time step to the information. Other friendship relations are still assumed to be unknown. $\texttt{Cancer}(\texttt{x}, \texttt{t})$ is unknown for all persons $\texttt{x}$ and all time steps. The ”observed” people were randomly chosen. The query predicate was `Cancer`.

In the first experiment, we investigated the compression ratio between standard FF and LFOFF for 10 time steps as shown in Fig. 3.3 (**Left**). As one can see, the size of the factor graph as well as the number of messages sent is much smaller for LFOFF. When there is no evidence, i.e., the ratio of observed people is 0.0, all the people are indistinguishable and we can achieve a great amount of lifting. Furthermore, the more evidence we have the smaller the amount of lifting. However, we can still almost half the number of edges and in turn the number of messages sent.

In the second experiment, we compared the “forwards-backwards” message protocol with the “flooding” protocol, the most widely used and generally best-performing method for static networks. Here, messages are passed from each variable to each corresponding factor and back at each step. Again, we considered 10 time steps. The results shown in Fig. 3.3 (**Right**) clearly favor the FB protocol.

For a qualitative comparison, we finally computed the probability estimates for $\texttt{cancer}(\texttt{A}, \texttt{t})$ using LFOFF and MC-SAT, the default inference of the ALCHEMY system[1]. For MC-SAT, we used default parameters. There were four persons (`A`, `B`, `C`, and `D`) and we observed that `A` smokes at time step 2. All other relations where unobserved for all time steps. We expect that the probability of `A` having cancer has a peak at $t = 2$ smoothly fading out over time. Fig. 3.3 (**Bottom**) shows the results. In contrast to LFOFF, MC-SAT does not show the expected behavior. The probabilities drop irrespective of the distance to the observation.

[1]http://alchemy.cs.washington.edu/

So far **Q1** is affirmatively answered. The results clearly show that by lifting we can exploit symmetries for inference in the graphical model. A large amount of compression and thereby a speed-up, however, is not guaranteed, especially in the presence of evidence. Another challenge for lifting is randomness in the graph structure as we will see in the following experiments.

Model Counting using Lifted Loopy Belief Propagation

Model counting is the classical problem of computing the number of solutions of a given propositional formula. It vastly generalizes the NP-complete problem of propositional satisfiability, and hence is both highly useful and extremely expensive to solve in practice. Interesting applications include multi-agent reasoning, adversarial reasoning, and graph coloring, among others.

Our approach, called LIFTED BPCOUNT, is based on BPCOUNT for computing a probabilistic lower bound on the model count of a Boolean formula F, which was introduced by Kroc *et al.* [89]. The basic idea is to efficiently obtain a rough estimate of the "marginals" of propositional variables using loopy belief propagation with damping. The marginal of variable u in a set of satisfying assignments of a formula is the fraction of such assignments with $u = \texttt{true}$ and $u = \texttt{false}$ respectively. If this information is computed accurately enough, it is sufficient to recursively count the number of solutions of only one of "F with $u = \texttt{true}$" and "F with $u = \texttt{false}$", and scale the count up accordingly. Kroc *et al.* have empirically shown that BPCOUNT can provide good quality bounds in a fraction of the time compared to previous, sample-based methods.

The basic idea of LIFTED BPCOUNT now is to *plug in lifted LBP in place of LBP*. However, we have to be a little bit more cautious: propositional variables can appear at any position in the clauses. This makes high compression rates unlikely because, for each supernode (set of propositional variables) and superfeature (set of clauses) combination, we carry a count for each position the supernode appears in the superfeature. Fortunately, however, we deal with disjunctions only (assuming the formula f is in CNF). Propositional variables may appear negated or unnegated in the clauses which is the only distinction we have to make. Therefore, we can safely assume two positions (negated, unnegated) and besides sorting the node color signatures we can now also sort the factor color signatures by position. Reconsider the example from Fig. 3.1 and assume that the potentials associated with f_1, f_2 encode disjunctions. Indeed, the order of the arguments of f_1, for instance, does not change the semantics of f_1. As our experimental results will show this can result in huge compression rates and large efficiency gains.

We have implemented (LIFTED) BPCOUNT based on SAMPLECOUNT [1] using our (L)LBP implementation. We ran BPCOUNT and LIFTED BPCOUNT on the circuit synthesis problem `2bitmax_6` with damping factor 0.5 and convergence threshold 10^{-8}. The formula has 192 variables, 766 clauses and a true count of 2.1×10^{29}. The resulting factor graph has 192 variable nodes, 766 factor nodes, and 1800 edges. The statistics of running (lifted) BPCount are shown in Fig. 3.4 (**Left**). It shows the ratio

[1] www.cs.cornell.edu/~sabhar/#software/

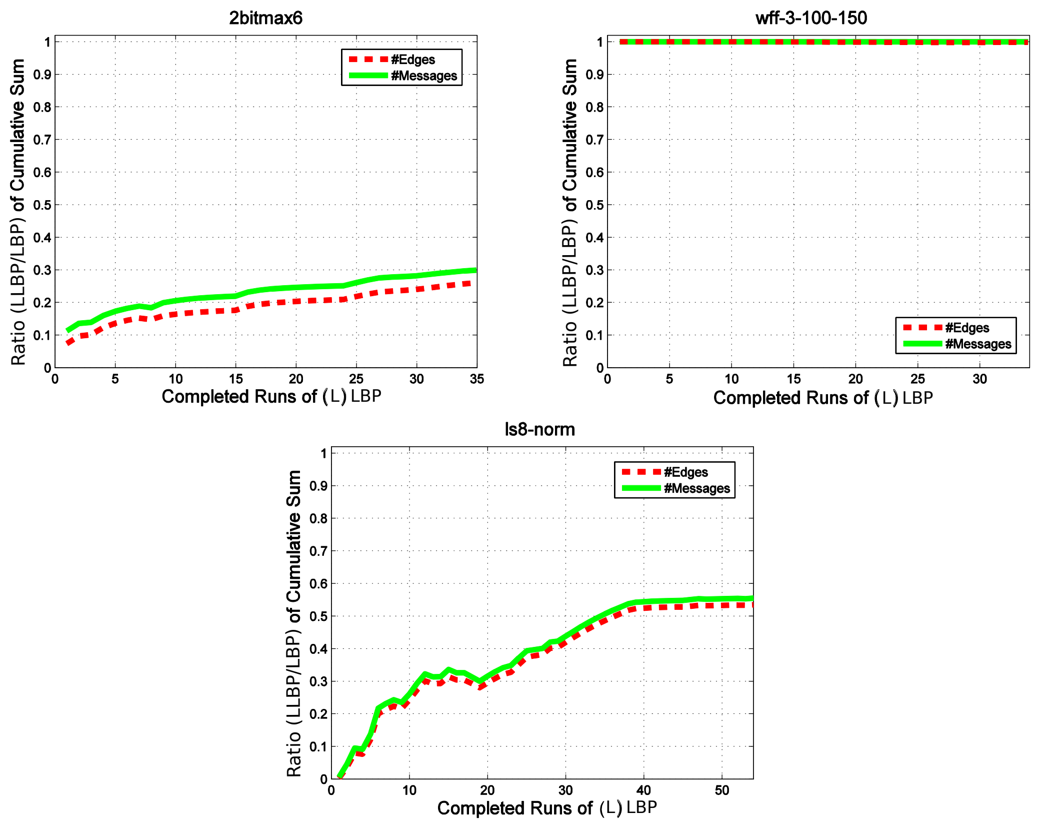


Figure 3.4: Ratios LIFTED BPCOUNT/BPCOUNT between 0.0 and 1.0 of the cumulative sum of edges computed respectively messages sent. A ratio of 1.0 means that LIFTED LBP sends exactly as many messages as LBP; a ratio of 0.5 that it sends half as many messages. (**Left**) `2bitmax_6`: Using LIFTED LBP saved 88.7% of the messages LBP sent in the first iteration of BPCOUNT; in total, it saved 70.2% of the messages. (**Right**) Random 3-CNF `wff-3-100-150`: No efficiency gain. The small difference in number of edges is due to a differently selected proposition due to tie breaking. (**Bottom**) `ls8-norm`: In the first iteration of LIFTED BPCOUNT, using LIFTED LBP saved 99.4% of the messages LBP sent. In total, this value dropped to 44.6%.

(LLBP/LBP) of messages sent (color messages included) and the number of edges. As one can see, a significant improvement in efficiency is achieved when the marginal estimates are computed using LIFTED LBP instead of LBP: LIFTED LBP reduces the messages sent by 88.7% when identifying the first, most balanced variable; in total, it reduces the number of messages sent by 70.2%. Both approaches yield the same lower bound of 5.8×10^{28}, which is in the same range as Kroc *et al.* report. Getting exactly the same lower bound was not possible because of the randomization inherent to BPCOUNT. Constructing the compressed graph took 9% of the total time of LIFTED LBP. Overall, LIFTED BPCOUNT was about twice as fast as BPCOUNT, although our LIFTED LBP implementation was not optimized.

Unfortunately, such a significant efficiency gain is not always obtainable. We ran BPCOUNT and LBPCOUNT on the random 3-CNF `wff-3-100-150`. The formula has 100 variables, 150 clauses and a true count of 1.8×10^{21}. Both approaches yield again the same lower bound, which is in the same range as Kroc *et al.* report. The statistics of running (lifted) BPCount are shown in Fig. 3.4 (**Right**). LIFTED LBP is not able to compress the factor graph at all. If there are no symmetries – such as in this random 3-CNF – lifted LBP essentially coincides with LBP. In turn, it does not gain any efficiency but actually produces a small overhead due to trying to compress the factor graph and by computing the counts.

In real-world domains, however, there is often a lot of redundancy. As a final experiment, we ran BPCOUNT and LIFTED BPCOUNT on the Latin square construction problem `ls8-norm`. The formula has 301 variables, 1601 clauses and a true count of 5.4×10^{11}. Again, we got similar estimates as Kroc *et al.*. The statistics of running (lifted) BPCount are shown in Fig. 3.4 (**Bottom**). In the first iteration, Lifted LBP sent only 0.6% of the number of messages LBP sent. This corresponds to 162 times less many messages sent than LBP.

The results on model counting and lifted inference in dynamic relational domains clearly affirm **(Q3.1)** and show that lifted loopy belief propagation can exploit symmetries and thus scale inference in relational as well as propositional domains **(Q3.2)**.

3.2 Informed Lifting for Loopy Belief Propagation

YouTube like media portals have changed the way users access media content in the Internet. Every day, millions of people visit social media sites such as Flickr and YouTube, among others, to share their photos and videos, while others enjoy themselves by searching, watching, commenting, and rating the photos and videos; what your friends like will bear great significance for you. The vision of social search underlies the great success of all the recent social platforms. However, while many of these services are centralized, i.e., rely on server replication, the full flexibility of social information-sharing can often be better realized through direct sharing between peers.

Let's consider the problem of distributing data to a large network more closely. A naïve solution to this problem involves unicasting the data to individual nodes from the source node. This approach, however, does not scale well as packages are sent again and again. Another simple solution is server replication, a solution that might not be cost effective for all users. In contrast, peer-to-peer solutions divide the file into parts and the nodes exchange these parts until they re-assemble back to the complete file.

Recently, Bickson *et al.* [14] have shown how to use LBP to solve this problem efficiently at realistic scale. More precisely, they have shown how to formalize the next-step problem — compute the next action that each peer in the content distribution network should take, where an action is the transfer of one file part from a neighboring node — as an undirected graphical model and ran LBP on this model in order to find the next best action. So, the question naturally arises: "Does Bickson *et al.*'s graphical model for solving the next-step problem also produce inference problems with symmetries in the graphical structure that are not being exploitet?" An investigation of this question was the seed that grew into our main contribution of this section: *informed* lifted LBP (iLLBP).

In fact, simply applying LLBP to Bickson *et al.*'s graphical model did not succeed. The reason is rather simple. Although there are symmetries exploitable for lifting, standard LBP runs simply too quickly: it often converges within few iterations. In contrast, LLBP's preprocessing that lifts the network in a first step takes more iterations, does not produce any belief estimates, and may yield lifted networks that are too pessimistic. The situation is depicted in Fig. 3.5. It shows the error curves of running (L)LBP on a local snapshot of a real-world file sharing network. As one can see, LBP only takes 5 iterations, whereas lifted LBP first takes 7 iterations of a color-message passing scheme to compute the lifted network and only then runs 5 highly efficient iterations simulating LBP on the lifted network. This is slower than LBP because:

- Computing the color-messages often takes essentially as much time as computing the LBP messages, in particular for discrete nodes with few states and low degree.
- The lifting step is purely syntactic. Colors only denote the nodes and factors producing LBP messages. Neither LBP messages nor any belief estimates are actually computed.
- In turn, LLBP cannot make use of the fact that LBP message errors decay along paths [70] already at lifting time. It may spuriously assume some nodes send and

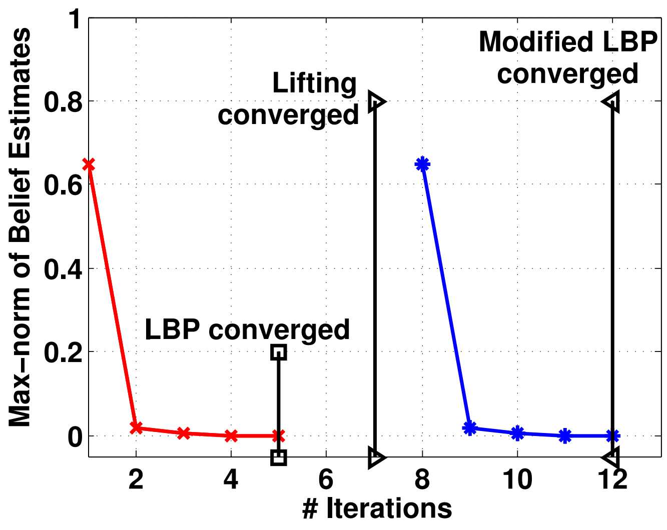


Figure 3.5: Max-norm of belief estimates vs. number of iterations of (L)LBP on a factor graph representing the collaboration graph of a local snapshot of the Gnutella network (1374 nodes, 4546 edges). LBP converges within 5 iterations. LLBP first lifts the network using color-passing (7 iterations) without estimating any beliefs at all; then it runs the highly efficient modified LBP on the lifted network. (Best viewed in color)

receive different messages and, hence, produce pessimistic lifted network.

Consequently it runs two additional rounds of color-message passing. Intuitively, for a long chain graph with identical, weak edge potentials, distant nodes will send and receive identical messages yet their computation trees are very different.

Informed lifted LBP (iLLBP) overcomes these problems. It is a novel, easy-to-implement, lifted LBP approach that tightly interleaves lifting and modified LBP iterations. This allows one to efficiently monitor the true LBP messages sent and received in each iteration and to group nodes accordingly. As our experiments show, significant efficiency gains are obtainable: iLLBP can yield lifted networks significantly faster. These networks are “lifted more”, i.e. they have more compression than LBP, while not degrading in performance. Above all, we show that iLLBP is faster than LBP when solving the problem of distributing data to a large network of fixed, known topology, an important real-world application where LBP is faster than (uninformed) LLBP.

Let us first illustrate the pessimistic nature of the (uninformed) LLBP. To do so, we analyze how LLBP behaves on the chain model assuming weak (but still identical) edge potentials. The lifted graph produced by LLBP is shown in Fig. 3.6 **(top)**. Due to the purely syntactic ”lifting” criterion used by CP — *group nodes according to identical (colored) computation trees*, the tree-structured unrolling of the underlying graph rooted at the nodes — nodes with the same distance to one of the ends of the chain are grouped together. Consequently, $n/2+1$ many supernodes are produced. Can we do any better?

It was shown by Ihler *et al.* [70] that message errors decay along paths. Intuitively, for long chain graphs — they are also a subproblem in both acyclic and cyclic graphical

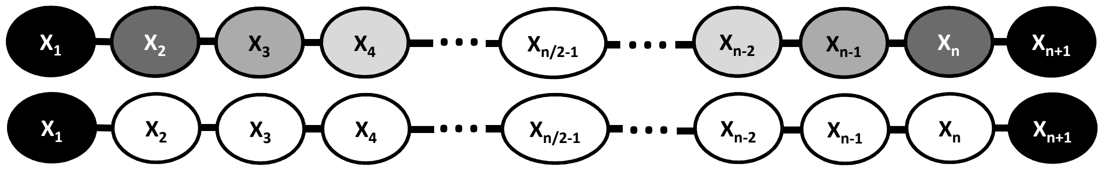


Figure 3.6: Supernodes — indicated by the colors of the original nodes — produced by uninformed **(top)** and informed **(bottom)** lifting on a chain graph model with $n+1$ nodes and identical, weak edge potentials. Factors have been omitted. Uninformed lifting produces $n/2-1$ many supernodes whereas informed lifting takes advantage of decaying message errors and produces only 2 supernodes: one for the end nodes of the chain and one for the other nodes.

models — with weak edge potentials, distant nodes are approximately independent. In general, when running LBP on arbitrary models, one often sees that some parts of the model form stable message clusters very quickly, whereas others take much longer time. So, the question naturally arises: "How can we make use of decaying errors with the goal to produce higher lifted graphs?"

Intuitively, making use of it would be easy, if we knew the true LBP messages sent and received in each iteration of LBP. We run one iteration of LBP, color nodes and factors according to the true LBP messages they sent and received, and iterate until the colors are not changing anymore. Because CP takes only finitely many iterations as proven in [146] and in Sec. 3.1, it follows that this informed coloring indeed converges after a finite number of iterations. The question is when? In the worst case, we may run LBP until convergence on the original graph before switching to the highly efficient modified LBP on the lifted graph. Hence, this naïve solution cancels the benefits of lifted inference

Consequently, we propose an adaptive approach, called *informed* LLBP, that interleaves CP and modified LBP iterations as summarized in Algorithm 3. First, we simulate LBP's first iteration (**lines 1-4**). We cluster the nodes with respect to the evidence and run one iteration of CP. On the resulting lifted factor graph $\mathfrak{G}$, we run one iteration of modified LBP. This is simulating the initial LBP iteration on the original factor graph G. Hence, we can make use of LBP's 1-iteration messages $m_i(x_i)$ and belief estimates $b_i(x_i)$. Consequently, we can safely group together nodes according to the LBP messages (**line 5**). In the while-loop (**line 6**), we repeat this process until the belief estimates converge. That is, we compute the (possibly refined) lifted network and run another CP-iteration (**line 7**). By induction, running modified LBP on the resulting lifted graph $\mathfrak{G}'$ (**line 9**) simulates the second LBP iteration on the original graph G. Hence, we can re-colorize nodes according to the true LBP messages (**lines 10-12**), and so on. It follows that iLLBP produces the same results as LBP. As we have stated in Sec. 2.3, encoding colors as natural numbers is rather arbitrary. Here we are using the real-valued messages to represent the colors and iLLBP produces the same results as running LLBP and in turn, it produces the same results as running LBP.

Note that, to reduce the $O(n^2)$ effort for naïvely re-coloring in each iteration, line 10

Algorithm 3: iLLBP – informed Lifted LBP. We use $b_i(x_i)$ resp. $m_i(x_i)$ to denote the unnormalized beliefs resp. messages of both variable node X_i and variable nodes covered by supernodes $\mathfrak{X}_i$.

```
   Data: A factor graph $G$ with variable nodes $X$ and factors $f$, Evidence $E$
   Result: Unnormalized marginals $b_i(x_i)$ for all supernodes and, hence, for all
           variable nodes
1  Colorize $X$ and $f$ w.r.t. $E$;
2  $\mathfrak{G} \leftarrow$ one iteration CP;
3  Initialize messages for $\mathfrak{G}$;
4  $(b_i(x_i), m_i(x_i)) \leftarrow$ one iteration modified LBP on $\mathfrak{G}$;
5  Colorize all $X_i$s according to $m_i(x_i)$;
6  while $b_i(x_i)$s have not converged do
7      $\mathfrak{G}' \leftarrow$ one iteration CP (based on new colors);
8      Initialize novel supernodes using current coloring based on $b_i(x_i)$ and $m_i(x_i)$;
9      $(b_i(x_i), m(x_i)) \leftarrow$ one iteration of modified LBP on $\mathfrak{G}'$;
10     foreach supernode $\mathfrak{X}$ in $\mathfrak{G}$ do
11         if the $m(x_i)$s of the $X_i$s in $\mathfrak{X}$ differ then
12             Colorize all $X_i$ in $\mathfrak{X}$ according to $m(x_i)$
13 Return $b_i(x_i)$ for all supernodes
```

iterates over all supernodes and checks whether any re-colorizing is required at all. This takes $O(n)$ time. If a change has been detected, only the variable nodes corresponding to the supernode at hand are re-colorized. We also note that in models over discrete variables with few states and low degree as often produced by Markov logic networks, iLLBP is very efficient: computing color messages is essentially as expensive as computing LBP messages.

The lifted graph produced by informed LLBP on the chain model example we started from is shown in Fig. 3.6 **(bottom)**. Independently of the domain size $n + 1$, only 2 supernodes are produced, an order of magnitude less than LLBP produces.

Evaluation

Our intention here is to investigate the following questions:

(Q3.1) Can iLLBP be faster than LLBP, also in cases where LBP is faster than LLBP?

(Q3.2) Can it yield more compressed lifted networks than LLBP?

To this aim, we implemented ((i)L)LBP in Python using the LIBDAI library [109] and evaluated their performances on the dynamic ”*Friends & Smokers*” Markov logic network (MLN) as well as for solving the content distribution problem on snapshots of real-world file-sharing networks. To assess performance, we report the number of nodes and respectively supernodes, as well as the number of edges of the original and the lifted

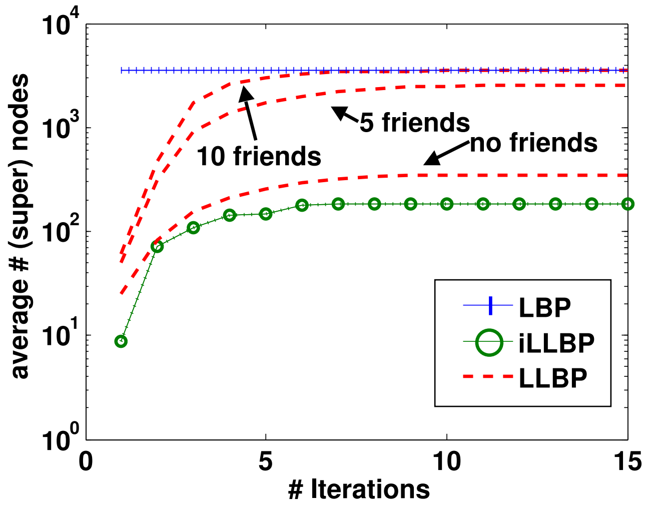


Figure 3.7: Number of super(nodes) (in log-scale) vs. iterations (including CP iterations for LLBP) averaged over 5 reruns for the social network, dynamic MLN. The number of (super)nodes roughly corresponds to the efficiency, i.e., the number of messages sent. As one can see, LLBP's performance degrades with more evidence. In contrast, iLLBP remains unaffected. In particular, iLLBP produces less many supernodes.

factor graphs. Running time is measured by the number of messages sent. For the typical message sizes, e.g., for binary random variables with low degree, computing color messages is at most as expensive as computing the actual LBP messages. Therefore, we view the running time of LLBP in terms of the number of both color and (modified) LBP messages computed, treating individual message updates as atomic unit time operations. In all experiments, we used the "flooding" message protocol. Here, messages are passed from each variable to each corresponding factor and back at each step. Messages were not damped, and the convergence threshold was 10^{-8}.

Social Networks

In our first experiment, we considered the dynamic ”*Friends & Smokers*” MLN as introduced in Sec. 3.1 for 20 people and 10 time steps. This created a ground factor graph consisting of 4400 nodes and 8420 factors. For 5 randomly selected people, we randomly choose whether they smoke or not and who $0, 5$ resp. 10 of their friends are, and randomly assigned a time step to the information. Other friendship relations are still assumed to be unknown. Whether a person has cancer or not is unknown for all persons and all time points.

The goal is to see whether iLLBP can produce more lifted networks than LBP. The number of (super)nodes vs iterations (averaged over 5 reruns) are summarized in Fig. 3.7. As one case see, iLLBP can indeed produce less many supernodes and, hence, be faster than LLBP. As more and more evidence is introduced, LLBP can not handle the additional evidence and the size of its compressed networks is closer to the size of the propositional network indicated by the horizontal line. Informed LLBP, on the other hand, is rather unaffected. The max-norm of iLLBP was always below 10^{-8}, the same threshold to determine whether loopy belief propagation is converged or not.

Content Distribution Problem (CDP)

Finally, we investigated the CDP. Bickson *et al.* assume that at the beginning of the download there is a source peer that contains all the file parts and that the other peers have no parts of the file. Starting from an initial graphical model, they run a decimation approach for solving the CDP:

1. Solve the next step problem running the LBP max-product inference algorithm.
2. Use the MAP assignment result as the solution.
3. Simplify the graphical model according to it.
4. Repeat.

Here, we use ((i)L)LBP in step (1).

Bickson *et al.* construct the graphical model essentially as follows (for more details, we refer to [14]). The peers participating in the download form the nodes of the underlying graphical model. Each peer/node has only local knowledge: the parts, which it presently holds, the list of its direct neighbors and the parts they hold. From this, all actions a peer can take follow such as "*download part i from neighbor j*". The initial probability of taking an action is encoded in the node potentials. There are several heuristics for initializing them. Here, we focused on the *uniform* policy that assigns equal probability to all parts. Similar results have been achieved using the *rarest part first* policy that assigns the highest probability to the part that has the lowest frequency in the network. Additionally, there is an edge potential between two peers/nodes that can take a part from a common neighbor. The edge potentials coordinate the actions among the peers. That is, we assign a very small value to pairs of actions that involves taking a part from the same third node: *at any time, only one peer can download a part from another peer*. The other values are set according to the node potentials. More details can be found in [14].

We simulated to distribute a single file(part) through a snapshot of the Gnutella network[1]. The max number of ((i)L)LBP iterations per decimation round was set to 40. The results are summarized in Fig. 3.8. As one can see, iLLBP can indeed be faster than LBP in cases where LLBP is slower than LBP. In total, LBP sent $5.761.952$ messages, iLLBP only $4.272.164$ messages, and LLBP $6.381.516$ messages (including color messages). On a similar run on a different Gnutella network[2] consisting of 6.301 nodes and 20.777 the approaches behaved similar. In total, LBP sent $5.761.952$ messages iLLBP only $1.972.662$ messages, and LLBP $2.962.311$ messages (including color messages).

[1]http://snap.stanford.edu/data/p2p-Gnutella04.html

[2]http://snap.stanford.edu/data/p2p-Gnutella08.html

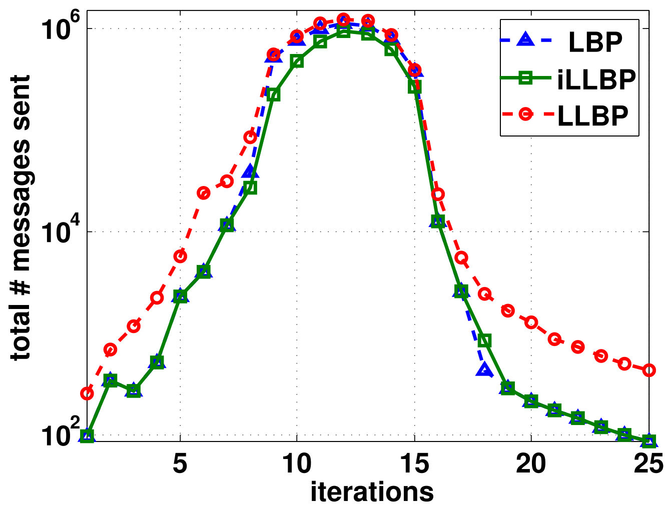


Figure 3.8: Total sum of messages (in log-scale) vs. content distribution decimation iterations on a Gnutella snapshot (10.876 nodes, 39.994 edges). The number of messages essentially corresponds to the efficiency. As one can see iLLBP is the most efficient one.

Again, the max-norm of iLLBP was always below 10^{-8}. Finally, quantitatively similar results have been achieved when distributing multiple file parts.

As the experimental results suggest, iLLBP produces highly lifted graphs extremely quickly. Moreover, in contrast to LLBP, belief estimates are computed (i) from the very first iteration on (ii) using modified LBP, i.e., really lifted inference.

Thus, to summarize, our questions can be answered affirmatively: iLLBP can produce much higher lifted networks in less time than LLBP while not degrading in accuracy. Beliefs are already estimated at lifting time and they converge to the same estimates as those produced by (L)LBP.

3.3 Lifted Gaussian Belief Propagation

Many real world applications such as environmental sensor networks, information diffusion in social networks, and localization in robotics involve systems of continuous variables. All previous LLBP approaches, however, have been developed for discrete domains only. While in principle they can be applied to continuous domains through discretization, Choi and Amir [30] have noted the precision of discretization deteriorates exponentially in the number of random variables. Thus, discretization and application of LLBP would be highly imprecise for large networks.

In this section, we present Gaussian belief propagation (GaBP) and show how the lifting that we have shown earlier (Sec. 3.1) can be applied to the Gaussian case leading to lifted GaBP (LGaBP). We develop LGaBP in the context of solving linear systems that are key to our lifted PageRank and Kalman filtering applications presented later in Ch. 6.

Gaussian Belief Propagation

One of the most fundamental problems encountered in real world applications is solving linear systems of the form

$$\mathbf{A}\mathbf{x} = \mathbf{b}\ ,$$

where $\mathbf{A} \in \mathbb{R}^{n \times n}$ is a real-valued square matrix, and $\mathbf{b} \in \mathbb{R}^n$ is real-valued column vector, and we seek the column vector $\mathbf{x}$ such that equality holds. As a running example, consider $\mathbf{b} = (0\ 0\ 1)^t$ (where ^{t} denotes transpose) and

$$\mathbf{A} = \begin{pmatrix} 10 & 4 & 3 \\ 4 & 10 & 3 \\ 5 & 5 & 11 \end{pmatrix}\ . \tag{3.4}$$

We could, of course, do that using traditional methods like Gaussian elimination or the Cholesky decomposition, however, we are going to take a rather different route, following Shental *et al.* [144]. The authors have shown how this problem can be cast into a probabilistic inference problem, i.e., to solve a linear system of equations of size n we compute the marginals of the Gaussian variables $x_1, \ldots, x_n$ in an appropriately defined graphical model.

Consider the quadratic form

$$Q(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} - \mathbf{b}^T\mathbf{x}\ .$$

Since $\nabla_{\mathbf{x}} Q(\mathbf{x}) = \mathbf{A}\mathbf{x} - \mathbf{b}$, and $Q(\mathbf{x})$ is convex ($\mathbf{A}$ is positive definite), there is a unique minimizer $\mathbf{x}^*$ of Q which satisfies $\nabla_{\mathbf{x}} Q(\mathbf{x}^*) = 0$. However, $\nabla_{\mathbf{x}} Q(\mathbf{x}^*) = 0$ implies that $\mathbf{A}\mathbf{x}^* - \mathbf{b} = 0$, i.e., $\mathbf{x}^*$ is a solution of the original problem. Using $Q(\mathbf{x})$ we can define the Gaussian distribution

$$p(\mathbf{x}) = \frac{1}{Z}\exp(-Q(\mathbf{x})) = \frac{1}{Z}\exp\left(-\frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} + \mathbf{b}^T\mathbf{x}\right)\ .$$

As $p(\mathbf{x})$ is monotonically decreasing with $Q(\mathbf{x})$, the maximum of p corresponds to the minimum of Q. We can now define $\mu = \mathbf{A}^{-1}\mathbf{b}$ and rewrite $p(\mathbf{x})$ as

$$\begin{aligned} p(\mathbf{x}) = Z^{-1} &\exp\left(\mu^T \mathbf{A}\mu/2\right) && (3.5)\\ &\times \exp\left(-\mathbf{x}^T \mathbf{A}\mathbf{x}/2 + \mu^T \mathbf{A}\mathbf{x} - \mu^T \mathbf{A}\mu/2\right) && (3.6)\\ &= \zeta^{-1} \exp\left(-(\mathbf{X} - \mu)^T \mathbf{A}(\mathbf{x} - \mu)/2\right) && (3.7)\\ &= \mathcal{N}(\mu, \mathbf{A}^{-1}), && (3.8) \end{aligned}$$

where $\zeta = Z \exp\left(-\mu^T \mathbf{A}\mu/2\right)$ is the new normalization factor. The sought after maximizer $\mathbf{x}^* = \mathbf{A}^{-1}\mathbf{b}$ is now identical to the mean vector μ of the distribution p. If we now compute the marginal distributions p_{x_i}, which are also Gaussian

$$p_{x_i} \sim \mathcal{N}(\mu_i = \left(\mathbf{A}^{-1}\mathbf{b}\right)_i, P_i^{-1} = (\mathbf{A}^{-1})_{ii}\right) \ ,$$

we can read off the means μ_i which correspond to individual components of the solution $\mathbf{x}^*$.

So, if we could obtain the marginals of p by means other than explicitly computing $\mathbf{A}^{-1}\mathbf{b}$, we would have an $\mathbf{x}^*$ without resorting to traditional linear equation solvers. This is where the graphical model view comes into play. As shown in [13, 144], p can be factorized according to the graph consisting of edge potentials ψ_{ij} and self potentials ϕ_i as follows:

$$p(\mathbf{x}) \propto \prod\nolimits_{i=1}^{n} \phi_i(x_i) \prod\nolimits_{i,j} \psi_{ij}(x_i, x_j) \ , \tag{3.9}$$

where the potentials are

$$\begin{aligned} \psi_{ij}(x_i, x_j) &:= \exp(-\frac{1}{2} x_i A_{ij} x_j)\\ \phi_i(x_i) &:= \exp(-\frac{1}{2} A_{ii} x_i^2 + b_i x_i). \end{aligned}$$

The edge potentials ψ_{ij} are defined for all (i, j) s.t. $\mathbf{A}_{ij} > 0$.

To solve the inference task, Shental *et al.* proposed to use Weiss *et al.*'s [162] Gaussian BP (GaBP) which is a special case of continuous LBP, where the underlying distribution is Gaussian. LBP in Gaussian models sends real-valued messages along the edges of the graph and gives simpler update formulas than the general continuous case and the message updates can directly be written in terms of the mean and precision. Since $p(\mathbf{x})$ is jointly Gaussian, the messages are proportional to Gaussian distributions $\mathcal{N}(\mu_{ij}, P_{ij}^{-1})$ with precision and mean defined as follows:

$$\begin{aligned} P_{ij} &= -A_{ij}^2 P_{i\backslash j}^{-1}\\ \mu_{ij} &= -P_{ij}^{-1} A_{ij} \mu_{i\backslash j} \end{aligned}$$

where

$$\begin{aligned} P_{i\backslash j} &= \tilde{P}_{ii} + \sum\nolimits_{k \in N(i)\backslash j} P_{ki}\\ \mu_{i\backslash j} &= P_{i\backslash j}^{-1} \big(\tilde{P}_{ii}\tilde{\mu}_{ii} + \sum\nolimits_{k \in N(i)\backslash j} P_{ki}\mu_{ki}\big) \end{aligned}$$

for $i \neq j$ and $\tilde{P}_{ii} = A_{ii}$ and $\tilde{\mu}_{ii} = b_i/A_{ii}$. Here, $N(i)$ denotes the set of all the nodes neighboring the ith node and $N(i) \setminus j$ excludes the node j from $N(i)$. All messages parameters P_{ij} and μ_{ij} are initially set to zero. The marginals are Gaussian probability density functions $\mathcal{N}(\mu_i, P_i^{-1})$ with precision and mean

$$P_i = \tilde{P}_{ii} + \sum\nolimits_{k \in N(i)} P_{ki}$$
$$\mu_i = P_{i \setminus j}^{-1} \big(\tilde{P}_{ii}\tilde{\mu}_{ii} + \sum\nolimits_{k \in N(i)} P_{ki}\mu_{ki}\big).$$

If the spectral radius of the matrix $\mathbf{A}$ is smaller than 1 then GaBP converges to the true marginal means ($\mathbf{x} = \mu$). We refer to [144] for details.

Lifting by Color Passing

As we have seen in Sec. 3.1, LLBP can exploit the graphical structure by automatically grouping nodes (potentials) of the graphical model G into supernodes (superpotentials) if they have identical *computation trees*. This compressed graph $\mathcal{G}$ is computed by passing around color signatures in the graph that encode the message history of each node. The signatures in the discrete case that we have been looking at are initialized with the evidence we have on the corresponding nodes. The key point to observe is that this very same process also applies to GaBP (viewing "identical" for potentials only up to a finite precision), thus leading to a novel LGaBP algorithm. The initial colors of the signatures with the evidence is equivalent to initializing with the color of the self potentials, i.e.,

$$cs_i^0 = \phi_i$$

As in the discrete case the color signature of a node i in the pairwise Gaussian model is then iteratively updated by

$$cs_i^k = \{cs_i^{k-1}\} \cup \{[\psi_{ij}, cs(\psi_j^{k-1})] | \, j \in N(i)\}.$$

Example 3.3.1.

$$\mathbf{A} = \begin{pmatrix} 10 & 4 & 3 \\ 4 & 10 & 3 \\ 5 & 5 & 11 \end{pmatrix} \, , \; \mathbf{b} = (0\;0\;1)^t$$

To continue our running example, let us examine computing the inverse of matrix $\mathbf{A}$ *in using LGaBP. We note that* $\mathbf{A}^{-1}$ *can be computed by solving* n *systems of linear equations one for each basis vector* $\mathbf{e}_i$*, i.e., we compute*

$$\mathbf{A}^{-1} = [\mathbf{x}_1, \dots, \mathbf{x}_n]$$
$$\mathbf{A}\mathbf{x}_i = \mathbf{e}_i \; , i = 1 \dots n$$
$$\mathbf{I} = [\mathbf{e}_1, \dots, \mathbf{e}_n]$$

for the $n \times n$ *identity matrix* $\mathbf{I}$*. In our running example, this solution of* $\mathbf{A}\mathbf{x}_i = \mathbf{e}_i$ *for* $i = \mathbf{e}_1, \dots, \mathbf{e}_n$ *yields the respective lifted networks in Figs. 3.9(a–c). As one can*

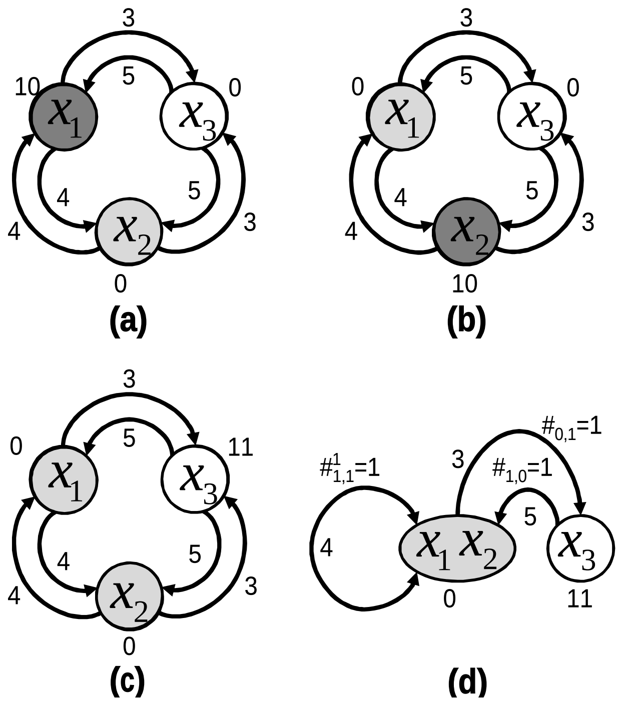


Figure 3.9: Lifted graphical models produced when inverting $\mathbf{A}$ in Eq. (3.4) using LGaBP. An edge from i to j encodes potential ψ_{ij}. The ϕ potentials are associated with the nodes. **(a)** Colored network when computing $\mathbf{A}\mathbf{x}_1 = \mathbf{e}_1$. All nodes get different colors; no compression and lifted inference is essentially ground. **(b)** Colored network for $\mathbf{A}\mathbf{x}_2 = \mathbf{e}_2$. Again no compression. **(c)** Colored network for $\mathbf{A}\mathbf{x}_3 = \mathbf{e}_3$. Nodes x_1 and x_2 get the same color and are grouped together in the lifted network **(d)**.

see, for the evidence cases e_1 and e_2 there is no compression and lifted inference is essentially ground. All nodes get different colors for these cases (Figs. 3.9(a) and (b)). For the evidence case $\mathbf{e}_3$, however, variables x_1 and x_2 are assigned the same color by LGaBP (Fig. 3.9(c)). The final lifted graph $\mathcal{G}$ is constructed by grouping all nodes (potentials) with the same color (signatures) into supernodes (superpotentials), which are sets of nodes (potentials) that behave identical at each step of carrying out GaBP on G (Fig. 3.9(d)).

On the lifted graph $\mathcal{G}$, LGaBP then runs a modified GaBP. The modified messages simulate running GaBP on the original graph G. Following LLBP, we have to pay special attention to the self-loops introduced by lifting that correspond to messages between different nodes of the same supernode. As shown in Fig. 3.9(d), there is a self-loop for the supernode $\{x_1, x_2\}$. In general, there might be several of them for each supernode and we assume that they are indexed by k. To account for the self-loops and in contrast to GaBP, we introduce “self-messages” $P^k_{ii} = -A^2_{ii}(P^k_{i\setminus i})^{-1}$ and $\mu^k_{ii} = -P^{-1}_{ij}A_{ii}\mu^k_{i\setminus i}$

with

$$P^k_{i\backslash i} = \tilde{P}_{ii} + (\sum_{l\in S(i)\backslash k} \sharp^l_{ii} P^l_{ii}) + (\sum_{l\in N(i)\backslash i} \sharp_{li} P_{li})$$

$$\mu^k_{i\backslash i} = P^{-1}_{i\backslash j}\Big[\tilde{P}_{ii}\tilde{\mu}_{ii} + \sum_{l\in S(i)\backslash k} \sharp^l_{ii} P^l_{ii}\mu^l_{ii} + \sum_{l\in N(i)\backslash i} \sharp_{li} P_{li}\mu_{li}\Big] \; .$$

As i is now a neighbor of itself, the term $N(i) \setminus i$ is required. Furthermore, $\sharp^l_{ij}$ — also given in Fig. 3.9(d) — are counts that encode how often the message (potential) would have been used by GaBP on the original network G. Using these counts we can exactly simulate the messages that would have been sent in the ground network. Messages between supernode i and j, $i \neq j$, are modified correspondingly:

$$P_{i\backslash j} = \tilde{P}_{ii} + \sum_{k\in S(i)} \sharp^k_{ii} P^k_{ii} + (\sum_{k\in N(i)\backslash i,j} \sharp_{ki} P_{ki})$$

$$\mu_{i\backslash j} = P^{-1}_{i\backslash j}\Big[\tilde{P}_{ii}\tilde{\mu}_{ii} + \sum_{k\in S(i)} \sharp^k_{ii} P^k_{ii}\mu^k_{ii} + \sum_{k\in N(i)\backslash i,j} \sharp_{ki} P_{ki}\mu_{ki}\Big]$$

where $N(i) \setminus i, j$ denotes all neighbors of node i without i and j.

Adapting the arguments for Th. 3.1.1, the following LGaBP correctness theorem on the correctness of LGaBP can be proven:

Theorem 3.3.2. *Given a Gaussian model G, LGaBP computes the minimal compressed lifted model, and running modified GaBP on $\mathcal{G}$ produces the same marginals as GaBP on G.*

4 MAP Inference

In this chapter we investigate the question of how to

> *find a* maximum a posteriori (MAP) *probability estimate or a mode of the posterior distribution P, i.e., computing* $\operatorname{argmax}_{\mathbf{X}} P(\mathbf{X}|E)$.

After reviewing MAP inference in Sec. 4.1, we show how we can answer a MAP inference problem by relaxing it into a *linear program*. In Sec. 4.2 we illustrate the symmetries that can arise in LPs and how we can employ lifted Gaussian belief propagation to solve the systems of linear equations arising when running an interior-point method to solve the LP (Sec. 4.3). However, this naïve solution cannot make use of standard solvers for linear equations and is doomed to construct lifted networks in each iteration of the interior-point method again which itself can be quite costly. To address both issues, we then show in Sec. 4.4 how to read off an equivalent lifted LP from the lifted GaBP computations that can be solved using any off-the-shelf LP solver.

The lifted linear programs introduced here lay the ground for lifted message-passing approaches to MAP inference. MPLP introduced by Globerson and Jaakola [54], for example, is a convergent message-passing algorithm for MAP inference and can be shown to be liftable using the results of the current chapter.

4.1 Maximum a posteriori (MAP) Estimation

The second type of query we investigate in the current chapter is the maximum a posteriori (MAP) query. This query is quite different to the marginal probability query we have looked at in Ch. 3. Here we ask for the most probable joint assignment to all variables given some evidence, instead of the individual states with maximum probability. These two can in fact be quite different.

Example 4.1.1. *Consider we have a small undirected model with two binary variables A and B and two factors f_1 and f_2, defined as follows:*

f_1	
A=0	*0.8*
A=1	*0.2*

f_2	*B=0*	*B=1*
A=0	*1*	*1*
A=1	*1*	*5*

Recall that, unlike Bayesian models, the factors do not necessarily have to represent probabilities, and f_2 is a function of both random variables and not a conditional probability $P(B|A)$. Computing the marginal probabilities one obtains $P(A=0) > P(A=1)$, whereas the state with the largest joint probability is $P(A=1, B=1) = 0.64$.

We can compute the max-marginals, i.e., the MAP probabilities, exactly by using a *variable elimination algorithm for MAP*. Since we are looking for the joint configuration with a maximum joint probability, the sums in the equations are replaced by *max*. As in the case for marginal probabilities variable elimination for MAP makes use of the fact that we can sometimes change the order of product- and max-computations an push in an operation if the scope of the variables admit that.

Example 4.1.2. *Now suppose we want to compute the max-marginals from Ex. 4.1.1. As already mentioned we need to maximize over all variables of interest. Thus we need to compute the maximum over all states a and b of variables A and B, respectively:*

$$\begin{aligned}\max_{a,b} f_1(a) \cdot f_2(a,b) &= \max_a \max_b f_1(a) \cdot f_2(a,b) \\ &= \max_a f_1(a) \max_b f_2(a,b)\end{aligned}$$

Since B is not in the scope of f_1 we can push in the maximization and compute the max-marginals more efficiently. In analogy we can sometimes change the order of summation and max-computations.

The final assignment is then obtained by traversing the variables in the reverse order and choosing the variables so as to maximize the corresponding factors corresponding to the previous choices (Traceback). All complexity results for variable elimination apply to variable elimination for MAP as well. Thus, in many cases an exact solution to the MAP problem via a variable elimination procedure is intractable. Note that, although MAP might seem easier than computing the marginal beliefs, it is NP-complete [86]. In this case we have to resort to approximate techniques such as message-passing procedures to compute approximate max-marginals. These can then be used for selecting an assignment.

The max-product algorithm computes the max marginals more efficiently. When each max-marginal is uniquely achieved, that is, $x_i^* \in \operatorname{argmax}_{x_i} \ \mu_i(x_i)$ is unique for all variables $X_i \in \mathbf{X}$, then $\mathbf{x}^* = (x_1^*, \ldots, x_n^*)$ is the global MAP configuration. When there is a tie, however, we have to keep track of the maximizing values of the neighbors to recover a global MAP configuration. The max-product algorithm is similar to the sum-product belief propagation algorithm with one exception: In Alg. 1 (**line 13**), the sum is replaced by a max, that is,

$$\mu^q_{f_k \to X_i}(x_i) \leftarrow \max_{\mathbf{x}_k \setminus \{x_i\}} \left(f(\mathbf{x}_k) \prod_{X_j \in \mathrm{nb}(f_k) \setminus \{X_i\}} \mu^{q-1}_{X_j \to f_k}(x_j) \right) . \tag{4.1}$$

Lifted loopy belief propagation clusters the nodes and factors with a purely syntactic criterion. Thus, the lifting naturally also applies to the max-product case and we can compute maximum joint assignments using lifted max-product belief propagation.

Max-product belief propagation often converges in practice. It is, however, not guaranteed to do so in loopy graphs. The global MAP optimization problem of finding a single joint assignment is essentially converted into an optimization problem in which the cliques are locally consistent. Furthermore, we can not give any optimality guarantees or tightly bound the error of the obtained solution. Despite the simplicity of the max-product belief propagation algorithm we thus seek for alternatives. The problem of solving the MAP can be expressed as an instance of many well-studied optimization problems such as linear programs, graph cuts or even re-factorization in the style of LBP [86]. For the purposes of this argument, we take special interest in the linear programming approach which has numerous advantages over the max-product approach:

(i) Linear optimization problems are widespread and we can rely on the huge body of work for solving linear programs.

(ii) LPs can easily be extended with additional constraints in case this should be necessary for a particular inference task.

(iii) The main advantage of using LPs is that optimality certificates for the obtained solutions can be given.

The linear programming view is developed as follows. Recall that a Markov random field or Markov network is a joint distribution over n random variables. Assume – without loss of generality – we have a binary pairwise MRF. The joint distribution is defined as follows:

$$P(\mathbf{X} = \mathbf{x}) = \frac{1}{Z} \prod_{i=1}^{n} \phi_i(x_i) \prod_{\{i,j\}} \psi_{ij}(x_i, x_j), \tag{4.2}$$

where the ϕ_i denote the unary factors and ψ_{ij} the pairwise factors. Assuming no configuration of $\mathbf{x}$ has probability zero, we may consider the logarithm of p since the maximum is preserved due to monotonicity. (4.2) then, $\log p$ denoted as the *energy*, becomes

$$\log P(\mathbf{X}=\mathbf{x}) = \sum_{i=1}^{n} \underbrace{\log \phi_i(x_i)}_{\theta_i(x_i)} + \sum_{\{i,j\}} \underbrace{\log \psi_{ij}(x_i,x_j)}_{\theta_{ij}(x_i,x_j)} + \theta_{\text{const}}, \tag{4.3}$$

with θ_{const} being a constant that can be discarded in the optimization. A common way to phrase the problem of maximizing the energy is via the following linear programming formulation [54]:

$$\boldsymbol{\mu}^{L*} = \operatorname*{argmax}_{\boldsymbol{\mu}\in\mathcal{M}_L(G)} \sum_{i=1}^{n} \theta_i(x_i)\mu_i(x_i) + \sum_{\{i,j\}} \theta_{ij}(x_i,x_j)\mu_{ij}(x_i,x_j) \ , \tag{4.4}$$

where the set $\mathcal{M}_L(G)$ is the local polytope,

$$\mathcal{M}_L(G) = \left\{ \boldsymbol{\mu} \geq 0 \;\middle|\; \begin{array}{l} \forall \{x_i,x_j\} \in \operatorname{dom}(\psi_{ij}) : \\ \sum_{\hat{x}_i} \mu_{ij}(\hat{x}_i,x_j) = \mu_j(x_j) \\ \sum_{\hat{x}_j} \mu_{ij}(x_i,\hat{x}_j) = \mu_i(x_i) \\ \forall x_i \in \mathbf{x} : \\ \sum_{x_i} \mu_i(x_i) = 1 \end{array} \right\} . \tag{4.5}$$

The local polytope as introduced above consists of two kinds of constraints [54]:

- Normalization constraints δ_i defined as $\sum_{x_i} \mu_i(x_i) = 1$ and
- Marginalization constraints $\lambda_{ij}(x_i)$ defined as $\sum_{\hat{x}_j} \mu_{ij}(x_i,\hat{x}_j) = \mu_i(x_i)$.

Note that, as mentioned before, solving the MAP problem exactly, that is, the integer linear program that computes an integral solution to the vector $\boldsymbol{\mu}$ is known to be an NP-complete problem. Thus, we relax the problem by allowing the μ's to be real numbers. We will call this particular linear relaxation over the local polytope MAP-LP.

Triggered by the success lifted LBP, it is reasonable to ask whether linear programming, another important AI technique, is liftable too. Indeed, at the propositional level, considerable attention has been already paid to the link between LBP and linear programming. This relation is natural since the MAP inference problem can be relaxed into linear programming, see e.g. [166].

At the lifted level, however, the link has not been established nor explored yet. Doing so significantly extends the scope of lifted inference since it paves the way to lifted solvers for linear assignment, allocation and flow problems as well as novel lifted (relaxed) solvers for SAT problems, Markov decision processes and maximum aposteriori (MAP) inference within probabilistic models, among others.

4.2 Symmetries in Linear Programming

To illustrate the symmetries that may arise in linear programs, consider an extension of the friends-and-smokers MLN [134] to targeted advertisement [28].

Example 4.2.1. *Suppose we want to serve smoking-related advertisements selectively to the smoking users of a website and advertisements not related to smoking, e.g., sport ads, to the non-smoking users, as to maximize the expected number of advertisements that will be clicked on. Companies make considerable revenue through advertising, and consequently attracting advertisers has become an important and competitive endeavor. Assume that a particular delivery schedule for advertisements is defined by the matrix* $\mathtt{show}(\mathtt{AType}, \mathtt{U}) \geq 0$ *denoting the number of times that advertisement of type* $\mathtt{AType}$ *is to be shown on a web site to a particular user* $\mathtt{U}$*, who may be a smoker with a certain probability, in a given period of time (e.g. a day). Assume further that we know the probability* $\mathtt{click}(\mathtt{AType}, \mathtt{UType})$ *that advertisement of* $\mathtt{AType}$ *will be clicked on if shown to a person type* $\mathtt{UType} \in \{\mathtt{Sm}, \mathtt{NonSm}\}$*. We model the overall probability of an advertisement of certain type to be clicked on by a given user, as the expectation*

$$\mathtt{click}(\mathtt{AType}, \mathtt{U}) := \sum\nolimits_{\mathtt{UType}} \mathtt{click}(\mathtt{AType}, \mathtt{UType}) \cdot \mathit{prob}(\mathtt{UType}, \mathtt{U}) \ ,$$

where $\mathit{prob}(\mathtt{UType}, \mathtt{U})$ *is the probability that a user is a smoker obtained by running inference in the friends-and-smokers MLN. We can express the expected number of clicks for any schedule* $\mathbf{X}$ *as*

$$\sum\nolimits_{\mathtt{AType}} \sum\nolimits_{\mathtt{U}} \mathit{prob}(\mathtt{AType}, \mathtt{U}) \cdot \mathtt{show}(\mathtt{AType}, \mathtt{U}).$$

Our goal now is to find the schedule that maximizes this expectation. However, companies typically enter into contracts with advertisers and promise to deliver a certain number $\mathtt{quota}(\mathtt{AType})$ *of advertisements of any type,*

$$\sum\nolimits_{\mathtt{U}} \mathtt{show}(\mathtt{AType}, \mathtt{U}) \geq \mathtt{quota}(\mathtt{AType}).$$

Moreover, if a certain user visits the site only $\mathtt{visits}(\mathtt{U})$ *times per day, our daily delivery schedule should not expect to serve more than* $\mathtt{visits}(\mathtt{U})$ *advertisements to them,*

$$\sum\nolimits_{\mathtt{AType}} \mathtt{show}(\mathtt{AType}, \mathtt{U}) \leq \mathtt{visits}(\mathtt{U}).$$

Thus, we would like to find the schedule that maximizes the expected number of clicks with respect to these constraints. This is a linear program and, since we can exploit symmetries within the friends-smokers MLN, it is intuitive to expect that we can also do so for solving this linear program. As a sneak preview, we illustrate that this is indeed the case. Instantiating the problem for two people Alice and Bob, for example, gives us the following LP:

$$\begin{aligned}
\max_{\mathbf{x}} \textstyle\sum_{U\in\{a,b\}} & prob(\mathtt{Sm},\mathtt{U})\cdot\mathtt{click}(\mathtt{SmAd},\mathtt{U}) \\
& + prob(\mathtt{NonSm},\mathtt{U})\cdot\mathtt{click}(\mathtt{SpAd},\mathtt{U}) \qquad (4.6)\\
s.t.\ & \mathtt{show}(\mathtt{SmAd},\mathtt{a})+\mathtt{show}(\mathtt{SmAd},\mathtt{b})\geq\mathtt{q}(\mathtt{SmAd}),\\
& \mathtt{show}(\mathtt{SpAd},\mathtt{a})+\mathtt{show}(\mathtt{SpAd},\mathtt{b})\geq\mathtt{q}(\mathtt{SpAd}),\\
& \mathtt{show}(\mathtt{SmAd},\mathtt{a})+\mathtt{show}(\mathtt{SpAd},\mathtt{a})\leq\mathtt{visits}(\mathtt{a}),\\
& \mathtt{show}(\mathtt{SmAd},\mathtt{b})+\mathtt{show}(\mathtt{SpAd},\mathtt{b})\leq\mathtt{visits}(\mathtt{b}).
\end{aligned}$$

Now consider the following variant introducing symmetries by adding one more person into the domain:

$$\begin{aligned}
\max_{\mathbf{x}} \textstyle\sum_{U\in\{a,b,c\}} & prob(\mathtt{Sm},\mathtt{U})\cdot\mathtt{click}(\mathtt{SmAd},\mathtt{U}) \\
& + prob(\mathtt{NonSm},\mathtt{U})\cdot\mathtt{click}(\mathtt{SpAd},\mathtt{U}) \qquad (4.7)\\
s.t.\ & \textstyle\sum_{\mathtt{p}\in\{\mathtt{a},\mathtt{b},\mathtt{c}\}}\mathtt{show}(\mathtt{SmAd},\mathtt{p})\geq\mathtt{q}(\mathtt{SmAd}),\\
& \textstyle\sum_{\mathtt{p}\in\{\mathtt{a},\mathtt{b},\mathtt{c}\}}\mathtt{show}(\mathtt{SpAd},\mathtt{p})\geq\mathtt{q}(\mathtt{SpAd}),\\
& \mathtt{show}(\mathtt{SmAd},\mathtt{a})+\mathtt{show}(\mathtt{SpAd},\mathtt{a})\leq\mathtt{visits}(\mathtt{a}),\\
& \mathtt{show}(\mathtt{SmAd},\mathtt{b})+\mathtt{show}(\mathtt{SpAd},\mathtt{b})\leq\mathtt{visits}(\mathtt{b}),\\
& \mathtt{show}(\mathtt{SmAd},\mathtt{c})+\mathtt{show}(\mathtt{SpAd},\mathtt{c})\leq\mathtt{visits}(\mathtt{c}).
\end{aligned}$$

Suppose we know as in the previous example that Alice is a smoker. For the others, however, we have no evidence. Thus they have identical cost in the objective and, due to the symmetric constraints, they are equal at the optimum. This is exactly what can be exploited by LGaBP.

Fig.4.1 shows the compression and efficiency gains that are achieved when processing the linear program with our method.

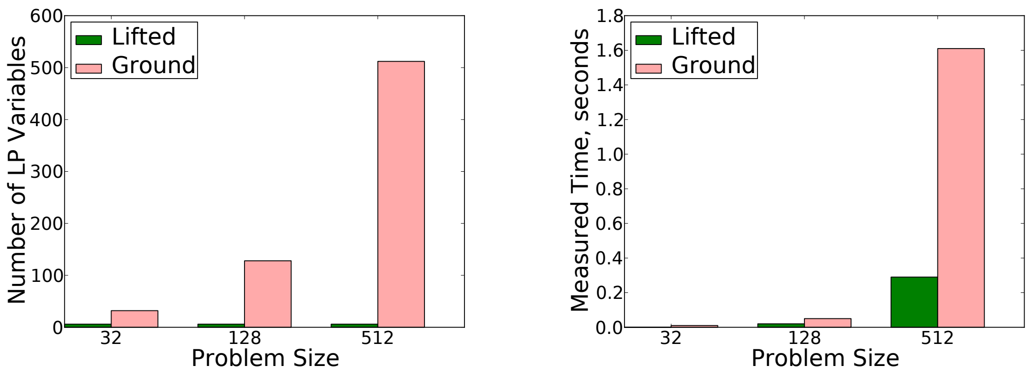


Figure 4.1: Computing an advertisement delivery schedule: Number of variables in the lifted and ground LPs **(left)** and measured time for solving the ground LP versus time for lifting and solving **(right)**.

To show why and how this can be done is exactly the focus of the present chapter. Specifically, our contribution is the first application of lifted inference techniques to linear programming.

To start, we note that the core computation of Bickson *et al.*'s [16] interior-point solver for LPs, namely solving systems of linear equations using Gaussian belief propagation (GaBP), can be naïvely lifted: replacing GaBP by lifted GaBP (Sec. 3.3). In fact, this naïve approach may already results in considerable efficiency gains. However, we can do considerably better. The naïve solution cannot make use of standard solvers for linear equations and is doomed to construct lifted networks in each iteration of the interior-point method again, an operation that can itself be quite costly. To address both issues, we show how to read off an equivalent LP from the lifted GaBP computations. This LP can be solved using any off-the-shelf LP solver. We prove the correctness of this compilation approach, including a lifted duality theorem, and experimentally demonstrate that it can greatly reduce the cost of inference.

4.3 Solving Linear Programs by Lifted Gaussian Belief Propagation

To recap, a primal linear program LP is a mathematical program of the following form:

$$\begin{aligned} \max\nolimits_{\mathbf{x}} \quad & \mathbf{c}^T\mathbf{x} \\ \text{s.t.} \;\; & \mathbf{A}\mathbf{x} = \mathbf{b}, \;\; \mathbf{x} \geq 0 \;, \end{aligned}$$

where $\mathbf{x} \in \mathbb{R}^n, \mathbf{c} \in \mathbb{R}^n, \mathbf{b} \in \mathbb{R}^m$ and $\mathbf{A} \in \mathbb{R}^{m \times n}$, $m \leq n$. We will also denote a LP in primal form as the tuple $LP = (A, b, c)$. Every primal LP has a dual linear program $\overline{LP}$ of the form

$$\min\nolimits_{\mathbf{y}} \quad \mathbf{b}^T\mathbf{y} \;\; \text{s.t.} \;\; \mathbf{A}^T\mathbf{y} \leq \mathbf{c} \;,$$

where strong duality holds, namely, if $\mathbf{x}^*$ and $\mathbf{y}^*$ are optima of both LP and $\overline{LP}$, $\mathbf{c}^T\mathbf{x}^* = \mathbf{b}^T\mathbf{y}^*$. A well-known approach for solving equality-constrained LPs, i.e., LPs in primal form is the primal barrier method, see e.g. [130], that is sketched in Alg. 4. It employs the Newton method to solve the following approximation to the original LP:

$$\begin{aligned} \max\nolimits_{x,\mu} \quad & \mathbf{c}^T\mathbf{x} - \mu \sum\nolimits_{k=1}^{n} \log x_k \\ \text{s.t.} \;\; & \mathbf{A}\mathbf{x} = \mathbf{b} \;. \end{aligned}$$

At the heart of the Newton method lies the problem of finding an optimal search direction. This direction is the solution of the following set of linear equations:

$$\underbrace{\begin{bmatrix} -\mu\mathbf{X}^{-2} & \mathbf{A}^T \\ \mathbf{A} & 0 \end{bmatrix}}_{=:\mathbf{N}} \underbrace{\begin{bmatrix} \Delta\mathbf{x} \\ \lambda^+ \end{bmatrix}}_{=:\mathbf{d}} = \underbrace{\begin{bmatrix} \mathbf{c} + \mu\mathbf{X}^{-1}e \\ 0 \end{bmatrix}}_{=:\mathbf{f}}, \tag{4.8}$$

Algorithm 4: Primal Barrier Algorithm

```
   Input: $\mathbf{A}$, $\mathbf{b}$, $\mathbf{c}$, $\mathbf{x}^0$, $\mu^0$, $\gamma$, stopping criterion
   Output: $\mathbf{x}^* = \arg\max_{\{\mathbf{x}|\mathbf{A}\mathbf{x}=\mathbf{b},\mathbf{x}\geq 0\}} \mathbf{c}^T x$
1  $k \leftarrow 0$;
2  while stopping criterion not fulfilled do
3      Compute Newton direction $\Delta\mathbf{x}$ by solving (4.8);
4      Set step size $t$ by backtracking line search;
5      Update $\mathbf{x}^{k+1} = \mathbf{x}^k + t \cdot \Delta\mathbf{x}$;
6      Choose $\mu^{k+1} \in (0, \mu^k)$;
7      $k \leftarrow k + 1$
8  return $\mathbf{x}^k$;
```

where $\Delta\mathbf{x}$ is the Newton search direction, $\mathbf{X}$ is the diagonal matrix $\mathrm{diag}(\mathbf{x})$ and λ^+ is a vector of Lagrangian multipliers that are discarded after solving the system.

Recently, Bickson *et al.* [16] have identified an important connection between barrier methods and probabilistic inference: solving the system of linear equations (4.8) can be seen as MAP inference within a pairwise Markov random field (MRF) over Gaussians, solved efficiently using Gaussian Belief Propagation (GaBP). Specifically, suppose we want to solve a linear system of the form $\mathbf{N}\mathbf{d} = \mathbf{f}$ where we seek the column vector $\mathbf{d}$ such that the equality holds. As we have seen in Sec. 3.3, this problem can be translated into a probabilistic inference problem. Given the matrix $\mathbf{N}$ and the observation matrix $\mathbf{f}$, the Gaussian density function

$$p(\mathbf{d}) \sim \exp(-\frac{1}{2}\mathbf{d}^t\mathbf{N}\mathbf{d} + \mathbf{f}^t\mathbf{d})$$

can be factorized according to the graph consisting of edge potentials ψ_{ij} and self potentials ϕ_i as follows:

$$p(\mathbf{d}) \propto \prod\nolimits_{i=1}^{n} \phi_i(d_i) \prod\nolimits_{i,j} \psi_{ij}(d_i, d_j) \ , \tag{4.9}$$

where the potentials are

$$\psi_{ij}(d_i, d_j) := \exp(-\frac{1}{2}d_i N_{ij} d_j) \tag{4.10}$$

$$\phi_i(d_i) := \exp(-\frac{1}{2}N_{ii}d_i^2 + b_i d_i) \ . \tag{4.11}$$

Computing the marginals for d_i using lifted GaBP (Sec. 3.3) gives us the solution of $\mathbf{N}\mathbf{d} = \mathbf{f}$.

Fig. 4.2 shows two ways to compute the solution of (4.8) in a single step of the log barrier method for the linear program given above. The first way (shown with dotted lines) constitutes of converting the matrix N and the vector f of our linear system (shown upper-left) into the corresponding MRF (shown below it). This MRF is lifted by color passing and by computing the marginals using LGaBP we obtain the solution of the linear system. From the picture it can be seen that lifting would group together the

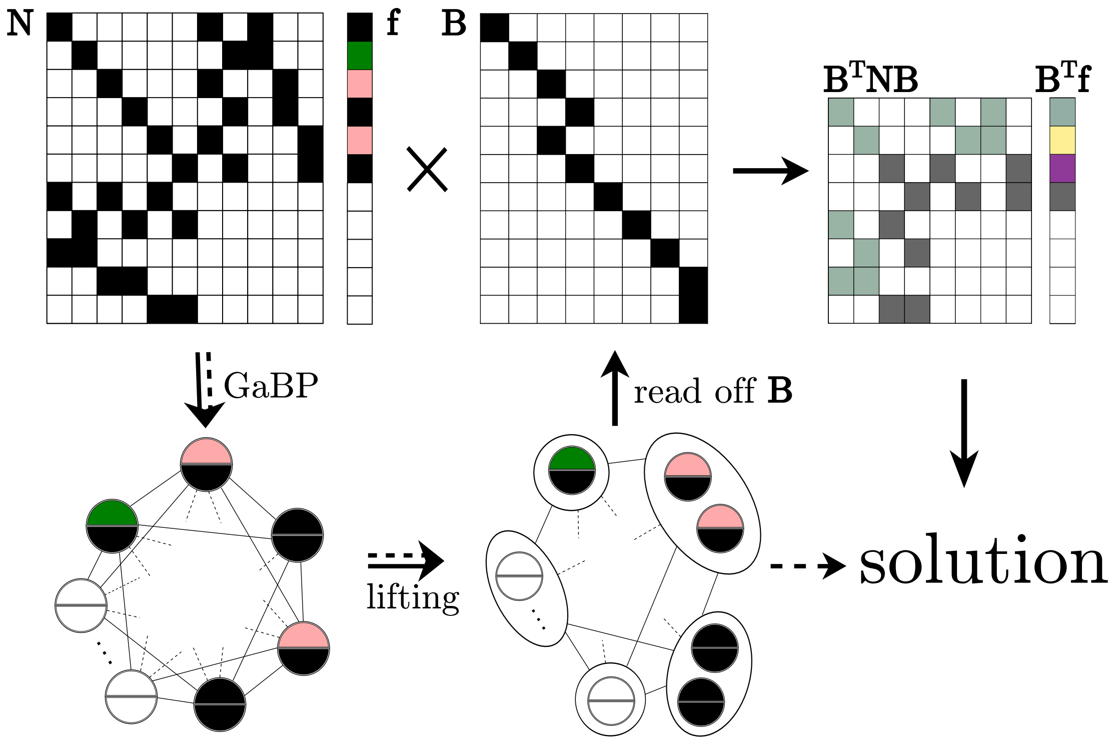


Figure 4.2: Lifted solving of the targeted advertisement LP with three persons. The dotted lines trace the LGaBP solution, whereas the solid lines compile the LP into a smaller set of lifted linear equations.

nodes corresponding to those who are indistinguishable in the MLN. The second way, outlined with solid lines, and its benefits are explained in the following.

4.4 Lifted Linear Programs

First, we show that we can read off an equivalent system of linear equations, called lifted linear equations, from the computations of lifted GaBP, thus avoiding to run a modified (Ga)BP. Then, we show that this result can be extended to reading off a single LP only, the lifted LP, thus avoiding the time-consuming re-lifting in each iteration.

Lifted Linear Equations

Recall that the modified GaBP sends messages involving counts. This suggests that we are actually running inference on a misspecified probabilistic model. We now argue that this is actually the case. Specifically, we show that the lifted problem can be compiled into an equivalent propositional problem of the same size. The resulting set of lifted linear equations can be solved using any standard solver, including GaBP. To see this, we start by noting that LGaBP computes a lifted, i.e., compiled version $\mathbf{r} \in \mathbb{R}^k$ of the solution vector $\mathbf{x}$ with $k \leq n$. The ground solution can be recovered as $\mathbf{x} = \mathbf{Br}$ where $\mathbf{B}$ encodes the partition induced by the $\sim$ relation due to the color passing. It can be read off from the lifted graph $\mathcal{G}$ as follows: $B_{ij} = 1$ if x_i belongs to the j-th supernode of $\mathcal{G}$, and $B_{ij} = 0$ otherwise. The matrix $\mathbf{B}$ has full column rank. Thus, when solving $\mathbf{Ax} = \mathbf{b}$ using LGaBP, we find $\mathbf{r}$ such that $\mathbf{ABr} = \mathbf{b}$. Multiplying by the left pseudoinverse of $\mathbf{B}$, i.e., $(\mathbf{B}^T\mathbf{B})^{-1}\mathbf{B}^T$ on both sides, one obtains

$$(\mathbf{B}^T\mathbf{B})^{-1}\mathbf{B}^T\mathbf{ABr} = (\mathbf{B}^T\mathbf{B})^{-1}\mathbf{B}^T\mathbf{b}\ . \tag{4.12}$$

The matrix $\mathbf{Q} := (\mathbf{B}^T\mathbf{B})^{-1}\mathbf{B}^T\mathbf{AB}$ is the well-known quotient matrix [67] of the partition $\mathbf{B}$ of $\mathbf{A}$. Interestingly, a similar idea has been used to optimize Pagerank computations [9]. For the case of GaBP models, the partitioning of the nodes by color-passing corresponds to the so-called coarsest equitable partition of the graph. A defining characteristic of equitable partitions is that the following holds [62]:

$$\mathbf{AB} = \mathbf{BQ}\ . \tag{4.13}$$

Thus, $\mathbf{Q}$ is invertible if $\mathbf{A}$ is invertible. To see this, let $\mathbf{u}$ be an eigenvector of $\mathbf{Q}$, i.e., $\mathbf{Qu} = \lambda\mathbf{u}$. Multiplying from the left by $\mathbf{B}$ gives $\mathbf{BQu} = \lambda\mathbf{Bu}$. Plugging in (4.13), this can be rewritten to $\mathbf{ABu} = \lambda\mathbf{Bu}$. Now, if $\mathbf{A}$ is invertible, all its eigenvalues are non-zero. By the above the eigenvalues λ of $\mathbf{Q}$ are also non-zero; $\mathbf{Q}$ is invertible.

Since we only deal with invertible matrices, both $\mathbf{Ax} = \mathbf{b}$ and (4.12) have a unique solution, and when solving (4.12) to obtain $\mathbf{r}$, then $\mathbf{x} = \mathbf{Br}$ is the solution of $\mathbf{Ax} = \mathbf{b}$. Since $\mathbf{Q} \in \mathbb{R}^{k\times k}$, one obtains a problem of size equal to the size of the lifted LGaBP graph. Finally we note that $\mathbf{B}^T\mathbf{B}$ is a diagonal matrix where each entry on the diagonal is the (strictly positive) count of the respective supernode. Thus, we can directly compute $(\mathbf{B}^T\mathbf{B})^{-1}$ and may also solve

$$\mathbf{B}^T\mathbf{ABr} = \mathbf{B}^T\mathbf{b}. \tag{4.14}$$

instead of solving (4.12). In other words, instead of lifting a solver for sets of linear equations, we lift the equations themselves and employ any standard solver.

Reconsider the targeted advertisement of the previous section and the solution of the linear system using LGaBP. Applying (4.14) to the problem of computing the Newton step for this program yields the second path of Fig. 4.2, lifting the linear equations, thus avoiding to run a modified (Ga)BP.

Lifted Linear Programs

However, we have to compute a set of lifted equations in each iteration of the barrier method. Can we avoid this and instead automatically compile the original LP into an equivalent LP that can be significantly smaller? That is, are there also lifted linear programs?

In the following, we show that this is the case. For instance, (4.7) can automatically be compiled into (4.6).

Consider the linear system

$$\mathbf{A}_0\mathbf{v} = \mathbf{c}_0 \tag{4.15}$$

with $\mathbf{A}_0 = \begin{bmatrix} \mathbf{I}_n & \mathbf{A}^T \\ \mathbf{A} & \mathbf{I}_m \end{bmatrix}$ and $\mathbf{c}_0 = \begin{bmatrix} \mathbf{c} \\ \mathbf{b} \end{bmatrix}$, where $\mathbf{I}_n$ and $\mathbf{I}_m$ are identity matrices of size n and m. We call this system the skeleton of the Newton search direction equation (4.8). The following Lemma says that the $\sim$ relation induced on $\mathbf{v}$ when solving the skeleton using LGaBP, denoted as $\sim_s$, is valid for solving (4.8) in all iterations of the barrier method applied: if $v_i \sim_s v_j$ then $\mathbf{x}_i^k = \mathbf{x}_j^k$ for $k = 0, 1, 2, 3, \ldots$.

Lemma 4.4.1. *Let* $\mathbf{x}^0$ *be an interior point of a linear program LP such that* $x_i^0 = x_j^0$ *if* $v_i \sim_s v_j$. *Then for all iterations* k *of the barrier method it holds that* $x_i^k = x_j^k$.

Proof. We are proving this in two steps. First, we prove that if $i \sim_s j$ for two variables i and j then $i \sim_b j$ for any $\sim$ relation produced by running the barrier method using LGaBP. Assume we are running the barrier method solving (4.8) using LGaBP. The MRFs constructed for all iterations k differ only in the self potentials ϕ, which in turn are completely specified by the $\mathbf{X} = \operatorname{diag}(\mathbf{x}) =: \mathbf{n}$ and $\mathbf{c} + \mu\mathbf{X}^{-1}e =: \mathbf{m}$ vectors. The edge potentials are not changing over the iterations of the barrier method. Consequently, the vectors $\mathbf{n}$ and $\mathbf{m}$ also determine the lifting produced in each iteration k since they encode the color signatures of the nodes after one iteration of the color passing. So, as long as $n_i = n_j$ respectively $m_i = m_j$ for all i and j with $i \sim_b j$, color-passing will result in a $\sim$ relation that respects $i \sim_b j$, i.e., if $i \sim j$ then $i \sim_b j$. By design, however, this holds for the $\sim_s$.

Now, in the second part, we prove that using $\sim_s$ produces the same solution vector. We prove this by induction on k. For $k = 0$, this holds by choice of the initial feasible point. For $k \mapsto k+1$, consider two nodes i and j with $i \sim_s j$. First, $c_i = c_j$ by construction of the skeleton. Second, $x_i^k = x_j^k$ is true due to the induction hypothesis. Thus, $\hat{c}_i = c_i + \mu\frac{1}{x_i^k} = c_j + \mu\frac{1}{x_j^k} = \hat{c}_j$. This proves the induction step for the $\mathbf{c} + \mu\mathbf{X}^{-1}e$ values. In a similar way we can prove this for the $-\mu\mathbf{X}^{-2}$ values. Thus, the search direction vector Δx respects the partition induced by $\sim_s$. $\square$

Due to the lemma, we have identified a partitioning that is loop invariant of Alg. 4. We now show that a subset of it can be directly applied to the matrix $\mathbf{A}$ and the vector $\mathbf{b}$ of a primal LP.

We have already established that instead of running LGaB we may also solve (4.14) using any standard solver for systems of linear equations. When doing so, the solution of the original problem is given as $\mathbf{x} = \mathbf{B}\mathbf{r}$. We now have to impose the following restriction: when building the lifted graph of LGaBP, we leave the first n variables ungrouped, even if the lifting tells us some of them should be grouped together. Clearly, this might result in loss of efficiency, although it does not affect the correctness of the LGaBP result. However, as we show below, it helps to derive a "relaxed" version of (4.14), which is of particular interest to us since it reveals how a lifted linear program can be constructed. The restriction can be translated to the equations by writing the block matrix $\mathbf{B}$ as

$$\mathbf{B} = \begin{bmatrix} \mathbf{I} & 0 \\ 0 & \mathbf{B}_M \end{bmatrix}, \tag{4.16}$$

$$\begin{tikzcd}
\mathbf{LP} \arrow[r, "\text{Theorem 4.4.2}"] \arrow[d, leftrightarrow, "\text{duality}"] & \mathbf{LP}^* \arrow[d, leftrightarrow, "\text{duality}"] \\
\overline{\mathbf{LP}} & \overline{\mathbf{LP}^*} \arrow[l, "\text{Theorem 4.4.3}"']
\end{tikzcd}$$

Figure 4.3: Links between LPs and lifted LPs.

so that the result of the LGaBP computation is expressed as

$$\begin{bmatrix} \Delta \mathbf{x} \\ \lambda^+ \end{bmatrix} = \begin{bmatrix} \mathbf{I} & 0 \\ 0 & \mathbf{B}_M \end{bmatrix} \mathbf{r}. \tag{4.17}$$

Now we argue in a similar way as in the previous section: if $\mathbf{r}$ is the lifted solution to (4.8), then $\mathbf{B}^T\mathbf{N}\mathbf{B}\mathbf{r} = \mathbf{B}^T\mathbf{f}$, or

$$\left(\mathbf{B}^T \begin{bmatrix} -\mu\mathbf{X}^{-2} & \mathbf{A}^T \\ \mathbf{A} & 0 \end{bmatrix} \mathbf{B} \right) \mathbf{r} = \mathbf{B}^T \begin{bmatrix} \mathbf{c} + \mu\mathbf{X}^{-1}e \\ 0 \end{bmatrix}.$$

With (4.16) we can compute this system explicitly as

$$\begin{bmatrix} -\mu\mathbf{X}^{-2} & \mathbf{A}^T\mathbf{B}_M \\ \mathbf{B}_M^T\mathbf{A} & 0 \end{bmatrix} \mathbf{r} = \begin{bmatrix} \mathbf{c} + \mu\mathbf{X}^{-1}\mathbf{e} \\ 0 \end{bmatrix}. \tag{4.18}$$

Note that $\mathbf{B}^T\mathbf{N}\mathbf{B}$ is invertible since $\mathbf{A}$ has full row-rank and the rows of $\mathbf{B}_M^T\mathbf{A}$ are sums of disjoint sets of the columns of $\mathbf{A}$, thus $\mathbf{B_M}^T\mathbf{A}$ also has full row rank. This means that the unique solution of (4.18) is a solution of (4.8), even though $\mathbf{B}$ does not necessarily correspond to an equitable partition of $\mathbf{N}$.

Now, consider the $LP^* = (\mathbf{B}_M^T\mathbf{A}, \mathbf{c}, \mathbf{B}_M^T\mathbf{b})$ (note, the fact that we compile the $\mathbf{b}$-vector reveals why it is present in the skeleton equation). We prove the following lifting theorem for primal LPs.

Theorem 4.4.2. *For every linear program $LP = (\mathbf{A}, \mathbf{b}, \mathbf{c})$, there exists a linear program $LP^* = (\mathbf{B}_M^T\mathbf{A}, \mathbf{B}_M^T\mathbf{b}, \mathbf{c})$ of smaller or equal size such that (1) the feasible region of LP^* is a superset of the feasible region of LP, (2) LP^* has at least one solution in common with LP, and (3) a common optimum to both will be found by Alg. 4 given a suitable initial point.*

Proof. Let $\mathbf{x_0}$ be a feasible solution of LP, i.e., $\mathbf{A}\mathbf{x}_0 = \mathbf{b}$. Multiplying $\mathbf{B}_M^T$ on both sides preserves equality. Thus, $\mathbf{B}_M^T\mathbf{A}\mathbf{x}_0 = \mathbf{B}_M^T\mathbf{b}$. Therefore $\mathbf{x}_0$ is feasible for LP^*. This proves (1). Now, given an initial interior point that preserves the lifting of the skeleton, Eq. (4.8) for LP^* is equivalent to (4.18) for LP in every step of the primal barrier method due to Lemma 4.4.1. That is, solving one step for LP^* is equivalent to solving one step of LP using LGaBP since it obtains the same search direction $\Delta\mathbf{x}$ (Eq. (4.17)). This proves (2). Equality of both objective values also holds since the objective functions of the two LPs are the same. This proves (3). □

However, we have to be a little bit more careful. So far, we have assumed the existence of an initial points that preserves symmetries, i.e., the $\sim$ relation. We now justify this assumption.

Following Dantzig [36], we can always construct a modified version of LP, called by LP_a, by adding an extra variable x_a associated with a very high cost R:

$$\begin{array}{ll} \max_{\mathbf{x},x_a} & \mathbf{c}^T\mathbf{x} - Rx_a \\ \text{s.t.} & \mathbf{A}\mathbf{x} + (\mathbf{b} - \mathbf{A}\mathbf{x}^+)x_a = \mathbf{b}, \\ & \mathbf{x} \geq 0, x_a \geq 0, \end{array}$$

where x^+ is any vector with positive components. This LP has the following properties, see also [36]:

(i) if R is sufficiently high, then the set of optimal solutions of LP_a is the same as for LP

(ii) $\mathbf{x} = \mathbf{x}^+, x_a = 1$ is a valid feasible solution.

Thus, one can choose $x^+ = \mathbf{1}$ which respects the symmetries of (4.15) for the original LP. Moreover, it can be shown that (4.15) for LP_a has the same symmetries as for LP except that the extra variable x_a will not be grouped with any other variable.

Theorem 4.4.2 shows that for every linear program, we can construct a possibly smaller linear program which has the property that when solved by the primal barrier method, or for that matter any other that maintains symmetries of this kind, will "simulate", at least partially, the use of lifted inference for the linear system solution step, so that relifting in every iteration can be avoided. The drawback of this method is that since the feasible region of the new LP may be larger than that of the original, it cannot be guaranteed that if a solution is found under different conditions (e.g., different solver or a non-symmetric interior point), it will still be valid for the original. As we show now, this situation is remedied when working with inequality constrained LPs. So, how do the lifted versions of LPs with inequality constraints look like? An elegant way to see this is to consider the dual linear program $\overline{LP^*}$ of the lifted program LP^*:

$$\min_{\mathbf{w}} \;\; (\mathbf{B}_M^T\mathbf{b})^T\mathbf{w} \;\; \text{s.t.} \;\; (\mathbf{B}_M^T\mathbf{A})^T\mathbf{w} \leq \mathbf{c} \, .$$

We show now that $\overline{LP^*}$ is the lifted version of $\overline{LP}$ and if $\mathbf{w}$ is a solution of $\overline{LP^*}$ then $\mathbf{y} = \mathbf{B}_M\mathbf{w}$ is a solution to $\overline{LP}$. In other words, we show a lifting theorem of duality:

Theorem 4.4.3 (Lifted Duality). *For every dual linear program $\overline{LP} = (\mathbf{A}^T, \mathbf{c}, \mathbf{b})$ there exists an equivalent dual linear program $\overline{LP^*} = (\mathbf{A}^T\mathbf{B}_M, \mathbf{c}, \mathbf{B}_M^T\mathbf{b})$ of smaller or equal size, whose feasible region and optima can be mapped to the feasible region and optima of $\overline{LP}$.*

```
Algorithm 5: Lifted Linear Programming
   Input: An inequality-constrained LP $(\mathbf{A}, \mathbf{b}, \mathbf{c})$
   Output: $\mathbf{x}^* = \text{argmin}_{\{\mathbf{x}|\mathbf{A}\mathbf{x}\leq\mathbf{b}\}} \mathbf{c}^T\mathbf{x}$
 1 Construct the equality-constrained LP $(\mathbf{A}^T, \mathbf{c}, \mathbf{b})$;
 2 Lift the corresponding skeleton equation (4.15) using color-passing;
 3 Read off the block matrix $\mathbf{B}_M$;
 4 Obtain the solution $\mathbf{r}$ of the LP $(\mathbf{A}\mathbf{B}_M, \mathbf{b}, \mathbf{B}^T\mathbf{c})$ using any standard LP solver;
 5 return $\mathbf{x}^* = \mathbf{B}\mathbf{r}$;
```

Proof. If $\mathbf{w}$ is a feasible solution of $\overline{LP^*}$ then $\mathbf{c} \geq (\mathbf{B}_M^T\mathbf{A})^T\mathbf{w} = \mathbf{A}^T(\mathbf{B}_M\mathbf{w}) = \mathbf{A}^T\mathbf{y}$. Thus, $\mathbf{y} = \mathbf{B}_M\mathbf{w}$ is a feasible solution of $\overline{LP}$. Moreover, any optimum of $\overline{LP^*}$ is an optimum of $\overline{LP}$. To see this, let $\mathbf{x}, \mathbf{x}^*, \mathbf{y}, \mathbf{w}$ be any optima of $LP, LP^*, \overline{LP}$ and $\overline{LP^*}$ respectively. By strong duality it holds that $\mathbf{c}^T\mathbf{x} = \mathbf{b}^T\mathbf{y}$ and $\mathbf{c}^T\mathbf{x}^* = (\mathbf{B}_M^T\mathbf{b})^T\mathbf{w}$. By Theorem 4.4.2 we have $\mathbf{c}^T\mathbf{x} = \mathbf{c}^T\mathbf{x}^*$. Therefore, $\mathbf{b}^T\mathbf{y} = (\mathbf{B}_M^T\mathbf{b})^T w$. □

Thus, if we want to lift an inequality constrained linear program, we can simply view it as the dual of some primal program and apply Theorem 4.4.3. More importantly, however, the theorem tells us that the feasible region of the lifted dual is a subset of the feasible region of the dual such that it contains at least one optimum of the original problem. Thus, inequality-constrained LPs can be solved by any LP solver as we do not have to worry about initial points. This is summarized in Alg. 5, which we call lifted linear programming since using Theorems 4.4.2 and 4.4.3 the algorithm can be applied to any form of LPs, cf. Fig. 4.3.

Evaluation

Our intention here is to investigate the following questions:

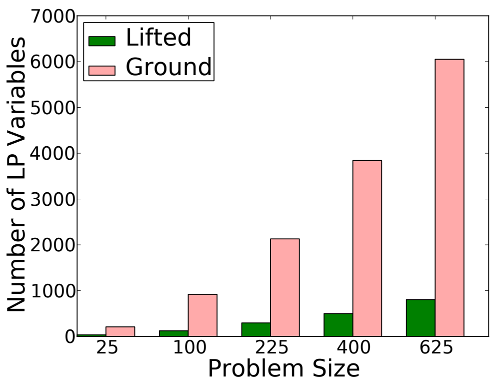
(Q4.1) Are there LPs that can be solved more efficiently by lifting?

(Q4.2) How much can we gain, given that we sacrifice the coarsest lifting for the construction of the lifted program.

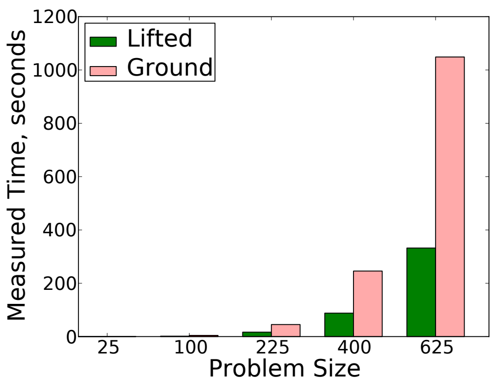
(Q4.3) How does lifting relate to the sparse vs. dense paradigm. Is it only making use of the sparsity in the LPs?

To this aim, we implemented lifted linear programming within Python[1] calling CVXOPT[2] as LP solver. All experiments were conducted on a standard Linux desktop computer.

[1]The implementation can be found at [104].
[2]`http://abel.ee.ucla.edu/cvxopt/`



(a) Number of variables in the lifted and ground LPs.



(b) Time for solving the ground LP vs. time for lifting and solving.

Figure 4.4: Experimental results for MAP inference on grids (best viewed in color).

Lifted MAP inference

As shown in previous chapters, inference in graphical models can be dramatically sped-up using lifted inference. Furthermore, a relaxed version of MAP inference can be solved by linear programs using the well-known LP relaxation, see e.g. [54] for details.

$$\begin{aligned} \max_\mu \quad & \sum\nolimits_{ij\in E}\sum\nolimits_{x_i,x_j} \theta_{i,j}(x_i,x_j)\mu_{ij}(x_i,x_j) \\ & +\sum\nolimits_i \sum\nolimits_{x_i} \mu_i(x_i)\theta_i(x_i) \\ \text{s.t.} & \sum\nolimits_{x_j} \mu_{ij}(x_i,x_j) = \mu_j(x_i), \;\; \sum\nolimits_{x_i} \mu_i(x_i) = 1. \end{aligned}$$

Thus, it is natural to expect that the symmetries in graphical models which can be exploited by standard lifted inference techniques will also be reflected in the corresponding linear program. To verify whether this is indeed the case we constructed pairwise MRFs of varying size. We scaled the number of random variables from 25 to 625 arranged in a grid with pairwise and singleton factors with identical potentials.

The results of the experiments can be seen in Figs. 4.4(a) and (b). As Fig. 4.4(a) shows, the number of LP variables is significantly reduced. The lifted linear programs are superior for all problem instances and show in all cases a reduction of the problem size to about ten percent compared to the ground version.

Furthermore, not only is the linear program reduced, but due to the fact that the lifting is carried out only once, we also measure a considerable decrease in running time as depicted in Fig. 4.4(b). Note that the time for the lifted experiment includes the time needed to compile the LP.

This experiment affirmatively answers **(Q4.1)** and shows that LPs can indeed be solved more efficiently by lifting.

-1	-1	-1	-1	100
-1	-1	-1	-1	-1
-1	-1	-1	-1	-1
-1	-1	-1	-1	-1
-1	-1	-1	-1	-1

(a) Gridworld example with 1 goal in the upper right corner

100	-1	-1	-1	100
-1	-1	-1	-1	-1
-1	-1	-1	-1	-1
-1	-1	-1	-1	-1
100	-1	-1	-1	100

(b) Gridworld example with a goal in every corner

Figure 4.5: Gridworld with 1 respectively 4 goal states.

Lifted MDPs

Another application of linear programs that we considered is the computation of the value function in a Markov decision process (MDP). MDPs are useful when an agent needs to reason about an ongoing process, for example, to *plan* its actions. Formally, an MDP is a discrete time stochastic control process here defined as a 5-tuple (S, A, P, C, γ). At each time step t, the process is in some state from the set of all possible states $s_t \in S$, and the agent may choose any action $a_t \in A$ that is available in state s_t. At the next time step the system moves into a new state s_{t+1} with a probability given by the state transition function, defined as

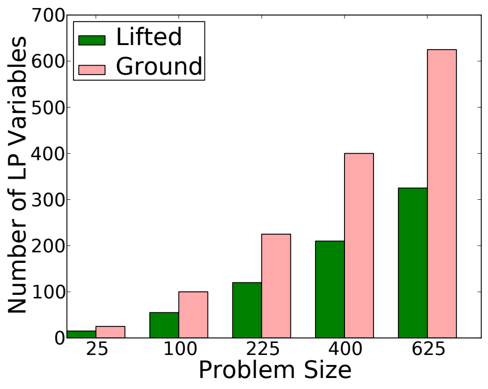
$$p_{ij}^k = P(s_{t+1} = j | s_t = i, a_t = k) \tag{4.19}$$

The cost of taking the action k in state i is given by c_i^k. A policy π is a mapping from states to actions, an optimal policy is one that maximizes the expected rewards. We can compute the value function using an LP formulation of this task as follows [95]:

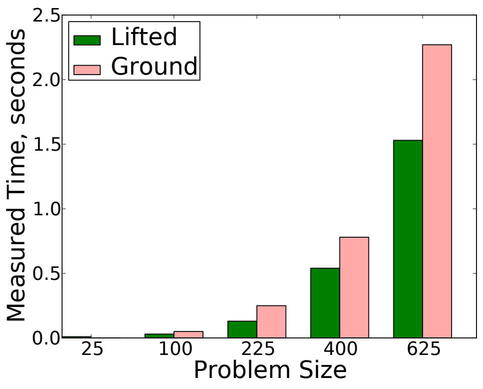
$$\max_{\mathbf{v}} \ \mathbf{1}^T \mathbf{v}, \ \text{ s.t. } v_i \leq c_i^k + \gamma \sum\nolimits_{j \in S} p_{ij}^k v_j,$$

where v_i is the value of state i, c_i^k is the reward that the agent receives when carrying out action k and p_{ij}^k is the probability of transferring from state i to state j by the action k. Later rewards are discounted by the factor γ.

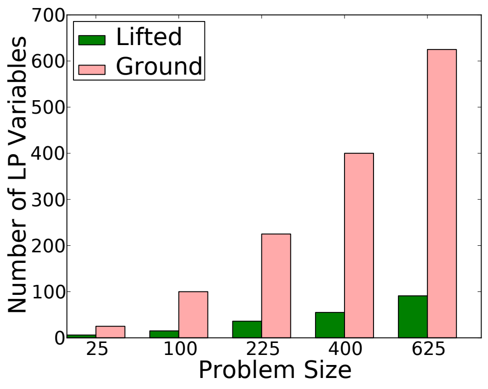
The MDP instance that we used is the well-known Gridworld (see e.g. [150]). The gridworld problem consists of an agent navigating within a grid of $n \times n$ states. Every state has an associated reward. Typically there is one or several states with high rewards, considered the goals, whereas the other states have zero or negative associated rewards. Further details on MDPs can be found in [74, 150].



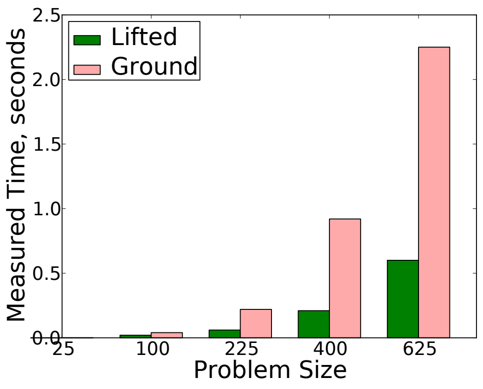
(a) Ground vs. lifted variables on a basic gridworld MDP.



(b) Measured times on a basic gridworld MDP.



(c) Variables on a gridworld with additional symmetry.

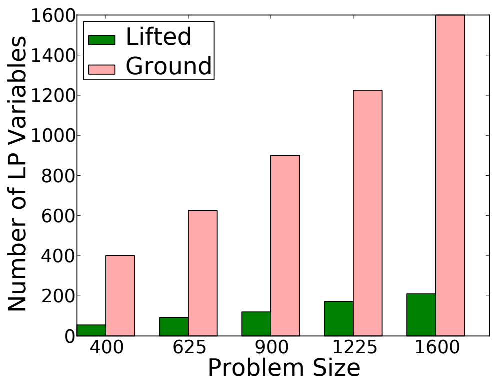


(d) Measured times on a gridworld with additional symmetry.

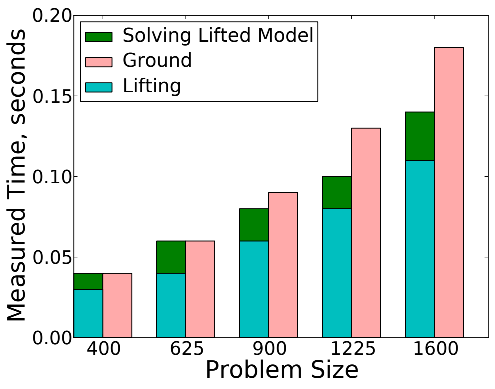
Figure 4.6: Experimental results for the gridworld MDP.

At first we considered an instance of gridworld with a single goal state in the upper-right corner with a reward of 100. Fig. 4.5 (a) shows the grid for the smallest instance of 5×5. The reward of all other states was set to -1. As can be seen in Fig. 4.6(a), this example can be compiled to about half the original size. Fig. 4.6(b) shows that already this compression leads to improved running time. We now introduce additional symmetries by putting a goal in every corner of the grid, as seen in the example in Fig. 4.5 (b). As one might expect this additional symmetry gives more room for compression which further improves efficiency as reflected in Figs. 4.6(c) and 4.6(d).

The two experiments presented so far affirmatively answer question **(Q4.1)**. However, the examples that we have considered so far are quite sparse in their structure. Thus, one might wonder whether the demonstrated benefit is achieved only because we are solving sparse problem in dense form. To address this we convert the MDP problem to a sparse representation for our further experiments. We scaled the number of states up to 1600 and as one can see in Fig. 4.7(a) and (b) lifting still results in an improvement of size as well as running time. Therefore, we can conclude that lifting an LP is beneficial



(a) Variables on a gridworld with additional symmetry in sparse form.



(b) Measured times on a gridworld with additional symmetry in sparse form.

Figure 4.7: Experimental results gridworld sparse form

regardless of whether the problem is sparse or dense, thus one might view symmetry as a dimension orthogonal to sparsity which answers question **(Q4.3)**. Furthermore, in Fig. 4.7(b) we break down the measured total time for solving the LP into the time spent on lifting and solving respectively. This presentation exposes the fact that the time for lifting dominates the overall computation time. Clearly, if lifting was carried out in every iteration (CVXOPT took on average around 10 iterations on these problems) the approach would not have been competitive to simply solving on the ground level.

This justifies that the loss of potential lifting we had to accept in order to not carry out the lifting in every iteration indeed pays off **(Q4.2)**. Remarkably, these results follow closely what has been achieved with MDP-specific symmetry-finding and model minimization approaches [44, 113, 131].

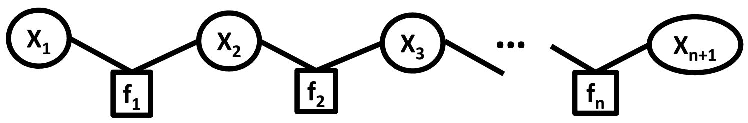
5 Higher Order Marginals

In Ch. 3 we have shown that lifted message passing approaches can be extremely fast at computing approximate marginal probability distributions over single variables and neighboring ones in the underlying graphical model. They do, however, not prescribe a way to solve more complex inference tasks such as the one we will investigate in this chapter, namely

> *computing the joint marginal distribution* $P(\mathbf{X_A}|E)$ *of a subset* $\mathbf{X_A} \leq \mathbf{X}$ *of nodes provided some evidence* E.

In the case of single nodes and neighboring random variables the marginals can be computed by the product of all incoming messages into the node (or factor respectively) after (lifted) loopy belief propagation has converged. For joint distributions over pairs, triples or k-tuples of distant random variables, however, this is not possible and lifted message passing approaches leave space for improvement.

BP-guided decimation for satisfiability and sampling [99, 108], for instance, are two tasks that are crucial to AI and have important applications. For example, sampling is often used in parameter learning and the idea of ”reduction to SAT” is a powerful paradigm for solving problems in different areas. In these tasks one is essentially interested in probabilities of the same network but with changing evidence. A popular solution in these cases is the idea of turning the complex inference task into a sequence of simpler ones as shown in Sec. 5.1. This can be done efficiently in a lifted fashion (Sec. 5.2). However, naïvely applying lifting to the conditioning relifts the network from scratch in every iteration. By reusing the previously lifted models we can avoid



X_1	X_2	f_1
True	True	1.2
True	False	1.4
False	True	2.0
False	False	0.4

X_2	X_3	f_2
True	True	1.2
True	False	1.4
False	True	2.0
False	False	0.4

...

X_n	X_{n+1}	f_n
True	True	1.2
True	False	1.4
False	True	2.0
False	False	0.4

Figure 5.1: A factor graph — a chain graph model with $n + 1$ nodes — with associated potentials. Circles denote variables, squares denote factors.

this relifting as we show in Sec. 5.3. Finally we briefly touch upon the conditioning order in Sec. 5.4 which, unlike the ground case, is crucial in the lifted case as it determines how much lifting is possible,

5.1 Joint Marginals by Conditioning

We decompose the complex inference task of computing arbitrary joint marginals into a series of inference tasks for single node marginals on the same graphical model, each time with changing evidence by selecting and clamping variables one at a time. Naïvely running lifted message passing again after each selection, however, recomputes the lifted network in each step from scratch, therefore often canceling the benefits of lifted inference. To avoid this, we need to find a way to efficiently compute the lifted network for each conditioning directly from the one already known for the single node marginals.

Example 5.1.1. *Let us illustrate the problem on a small example. Consider we want to compute joint marginals for the network in Fig. 5.1. It shows a factor graph which is a chain graph model with $n+1$ nodes. Computing the joint probability distribution for neighboring nodes given some evidence E, e.g. $P(X_1, X_2|E)$, can efficiently be done by one inference task using lifted loopy belief propagation, as we have shown in Ch. 3. The joint probability distribution for distant nodes such as $P(X_1, X_{n+1}|E)$, however, decomposes into the following steps:*

- *Compute $P(X_1|E)$ as in Ch. 3*
- *Condition the network on X_1*
- *Compute $P(X_{n+1}|E \cup \{X_1\})$ as in Ch. 3*
- *Compute $P(X_1, X_{n+1}|E) = P(X_{n+1}|E \cup X_1)P(X_1|E)$*

Essentially, we need to select variables one at a time for conditioning and run inference after each selection.

Naïvely applying lifted loopy belief propagation is wasteful as it recomputes the lifted network in each step from scratch. Recall that the lifting procedure is initialized with the evidence on the random variables. In our example we thus first have to lift the network for evidence E and then $E \cup \{X_1\}$.

Our goal is to avoid this by efficiently computing the lifted network for each conditioning directly from the one already known for the single node marginals.

There has been some prior work for related problems. Delcher *et al.* [45] propose a data structure that allows efficient queries when new evidence is incorporated in singly connected Bayesian networks and Acar *et al.* [1] present an algorithm to adapt the model to structural changes using an extension of Rake-and-Compress Trees. Van den Broeck and Davis [156] tackle the problem of conditioning in the context of first-order compilation for exact lifted probabilistic inference and show how it can efficiently be done for propositions and unary relations. The only other lifted message-passing approach we are aware of is the work by Nath and Domingos [116]. The authors essentially memorize the intermediate results of previous liftings. For new evidence they make a warm start from the first valid intermediate lifting. The algorithm of Nath and Domingos is only defined for Markov Logic Networks, whereas our approach does not rely on a relational representation and applies to any factor graph. *Conditioned* LLBP (CLLBP) is applicable for propositional models as well, thereby opening up a wide range of applications like satisfiability of boolean formulae and sampling among others. Furthermore, the approach presented in this chapter uses a clear characterization of the core information required for sequential clamping for lifted message passing: the shortest-paths connecting the variables in the network. We also consider the problem of determining the best variable to condition on in each iteration to stay maximally lifted over all iterations and propose a simple heuristic.

These contributions are significant in the lifted inference domain as they have applications to computing MAP assignments, sequential forward sampling, sensitivity analysis, among other tasks. For example, consider approximately solving the problem of maximum a posteriori (MAP) hypothesis: find the most probable instantiation e of some variables E. One class of approximate methods for MAP, cf. [123] and references in there, starts with some instantiation e and then tries to improve on it using local search, by examining all instantiations that result from changing the value of a single variable in e (called the neighbors of e). CLLBP provides an efficient mean to score the neighbors of e during local search: we first compute the lifted network given no evidence; now we can efficiently compute any lifted network during local search. As another application consider solving the SAT problem. It consists of a formula F representing a set of constraints over n Boolean variables, which must be set so as to satisfy all constraints. Survey propagation [100] solves the SAT problem using the decimation process, which is again essentially conditioning: assign a truth value to one variable (or a few variables) of F and simplify F, obtaining a smaller formula on $n-1$ variables. Now, survey propagation repeatedly decimates the formula in this manner, until a simplified instance is obtained that is easily solved , e.g., by existing SAT solvers.

CLLBP may also have future applications in more advanced relational learning tasks such as active learning. Clearly, information about dependencies between random variables such as mutual information is relevant for learning the structure of probabilistic relational models. Ground atoms which have a big impact on the system are good candidates.

In the following we will introduce CLLBP, then prove its soundness, and touch upon the problem of determining the best variable to condition on at each level of recursion.

5.2 Lifted Conditioning

When we are faced with the problem of repeatedly answering slightly modified queries on the same network, e.g., computing a joint distribution $P(X_1, X_2, \ldots, X_k)$ using LLBP, we use the *conditioning* procedure that we call *conditioned* LLBP (CLLBP). Let π define a *conditioning order* on the nodes, i.e., a permutation on the set $\{1, 2, \ldots, k\}$ and its i-th element be denoted as $\pi(i)$. The simplest permutation is $\pi(i) = i$, $i \in \{1, 2, \ldots, k\}$. Now, we select variables one at a time for conditioning, running LLBP after each selection, and combine the resulting marginals. More precisely,

1. Run LLBP to compute the prior distribution $P(X_{\pi(1)}|E)$.
2. Clamp $X_{\pi(1)}$ to a specific state $x_{\pi(1)}$. Run LLBP to compute the conditional distribution $P(X_{\pi(2)}|E, x_{\pi(1)})$.
3. Do this for all states of $X_{\pi(1)}$ to obtain the conditional distribution $P(X_{\pi(2)}|E, X_{\pi(1)})$. The joint distribution is now computed as

$$P(X_{\pi(2)}, X_{\pi(1)}) = P(X_{\pi(2)}|X_{\pi(1)}) \cdot P(X_{\pi(1)}) .$$

By iterating steps 2) and 3) and employing the chain rule we have

$$P(X_1, \ldots, X_k) = P(X_{\pi(1)}, \ldots, X_{\pi(k)}) = \prod\nolimits_{i=1}^{k} P(X_{\pi(i)}| \bigcap_{j=1}^{k-1} X_{\pi(j)}) .$$

CLLBP is simple and even exact for tree-structured models. Although in graphs with cycles the beliefs will in general not equal the true marginals, they often provide good approximations in practice. Welling and Teh [163] report that conditioning performs surprisingly well in terms of accuracy for estimating the covariance[1].

Because LLBP generally lacks the opportunity of adaptively changing the lifted graph and using the updated lifted graph for efficient inference, it is doomed to lift the original model in each iteration again from scratch. But how can we protect LLBP from performing poorly in terms of running time as we are repeatedly answering slightly modified queries on the same graph.

[1]The symmetrized estimate of the covariance matrix is typically not positive semi-definite and marginals computed from the joint distributions are often inconsistent with each other.

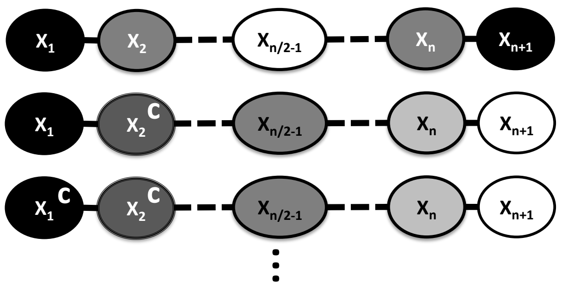


Figure 5.2: Colored chain graph model: Supernodes, indicated by the shades of the original nodes. The chain graph model with $n + 1$ nodes in the top shows the lifting without evidence, similar to 3.6, the other networks are produced by repeatedly clamping nodes, indicated by "c". Factors have been omitted. The conditioning order is $\pi = \{2, 1, 3, 4, \dots, n-2, n-1, n+1, n\}$. After clamping X_2 all subsequent LLBP runs work on the fully grounded network.

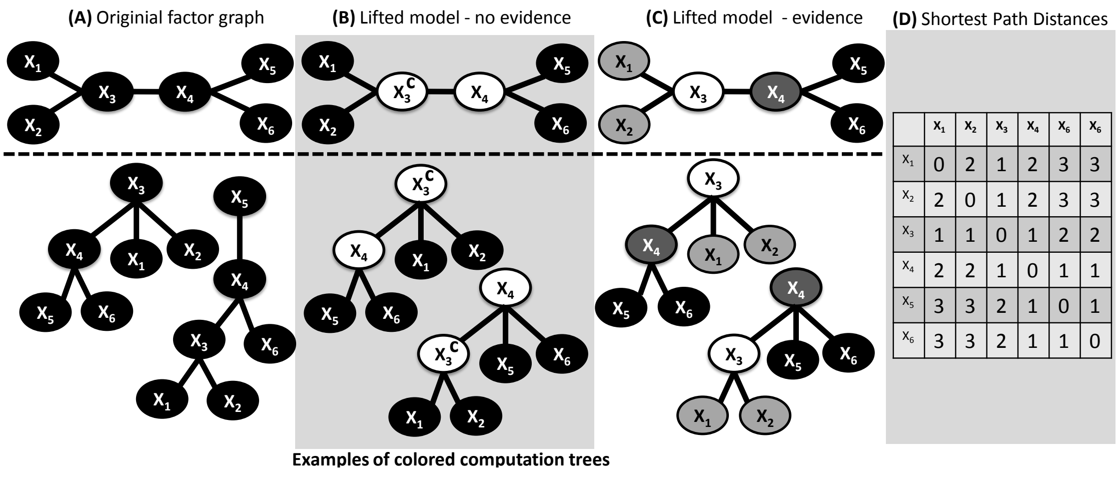
Each CP run scales $O(l \cdot (m + n \log n))$ where n is the number of nodes, m is the number of edges, and l is the length of the longest path without loop. Hence, CLLBP essentially spends $O(k \cdot l \cdot (m + n \log n))$ time just on lifting, when computing a joint distribution of k random variables. Moreover, in contrast to the propositional case, the conditioning order has an effect on the sizes of the lifted networks produced and, hence, the running time of LLBP. It may even cancel out the benefit of lifted inference. Reconsider our chain example[1] from Fig. 5.1. Fig. 5.2 sketches the lifted networks produced over time when using the conditioning order $\pi = (2, 1, 3, 4, \dots, n-2, n-1, n+1, n)$. That is, the lifting without any evidence is shown in the top, similar to the one shown in 3.6 in Sec. 3.2. First we clamp X_2, then all other nodes but X_n in ascending order. As one can see, clamping X_2 dooms all subsequent iterations to run modified LBP on the fully grounded network, canceling the benefits of lifted inference. In contrast, the order $\pi = (1, n+1, 2, n, \dots, n/2-1)$ produces lifted and fully grounded networks alternatingly, the best we can achieve for chain models.

We will address both issues in turn, namely efficient adaptation of the lifted network and finding an efficient conditioning order.

5.3 Shortest-Path Sequence Lifting

Consider the situation depicted in Fig. 5.3. Given the network in **(A)** and the prior lifted network, i.e., the lifted network when no evidence has been set **(B)**, we want to compute $P(X_i|x_3)$ as shown in **(C)**. To do so, recall the *colored computation trees* we have introduced in Sec. 3.1, where one considers for every node X_j the computation tree

[1]When the graph is a chain or a tree there is a single chain connecting any two nodes and LBP together with dynamic programming can be used to efficiently integrate out the internal variables. When cycles exist, however, it is unclear what approximate procedure is appropriate.


(A) Originial factor graph
(B) Lifted model - no evidence
(C) Lifted model - evidence
(D) Shortest Path Distances
Examples of colored computation trees

	X_1	X_2	X_3	X_4	X_6	X_6
X_1	0	2	1	2	3	3
X_2	2	0	1	2	3	3
X_3	1	1	0	1	2	2
X_4	2	2	1	0	1	1
X_5	3	3	2	1	0	1
X_6	3	3	2	1	1	0

Figure 5.3: **(A):** Original factor graph. **(B):** Prior lifted network, i.e., lifted factor graph with no evidence. **(C):** Lifted factor graph when X_3 is set to some evidence. Factor graphs are shown **(top)** with corresponding colored computation trees **(bottom)**. For the sake of simplicity, we assume identical factors that have been omitted here. Ovals denote variables/nodes. The shades in **(B)** and **(C)** encode the supernodes. **(D):** Shortest-path distances of the nodes. The i-th row will be denoted d_i.

rooted in X_j and each node in the tree is colored according to the nodes' initial colors, cf. Fig. 5.3**(bottom)**. Each CCT encodes the root nodes' local communication patterns that show all the colored paths along which node X communicates in the network. Consequently, CP groups nodes with respect to their CCTs: nodes having the same set of rooted paths of colors (node and factor names neglected) are clustered together. For instance, Fig. 5.3**(A)** shows the CCTs for X_3 and X_5. Because their set of paths are different, X_3 and X_5 are clustered into different supernodes as shown in Fig. 5.3**(B)**. The prior lifted network can be encoded as the vector $l = (0, 0, 1, 1, 0, 0)$ of node colors. Now, when we clamp a node, say X_3, to a value x_3, we change the communication pattern of every node having a path to X_3. Specifically, we change X_3's (and only X_3's) color in all CCTs X_3 is involved. This is illustrated in Fig. 5.3**(B)**. For the prior lifted network, the dark and light nodes in Fig. 5.3**(B)** exhibit the same communication pattern in the network and were consequently grouped together. Consequently, X_3 appears at the same positions in all corresponding CCTs. When we now incorporate evidence on node X_3, we change its color in all CCTs as indicated by the "c" in Figs. 5.3**(B)** and **(C)**. This affects nodes X_1 and X_2 differently than X_4 respectively X_5 and X_6 for two reasons:

(a) they have different communication patterns as they belong to different supernodes in the prior network; more importantly,

(b) they have different paths connecting them to X_3 in their CCTs.

Here we refer to the *shortest-paths* as the set of shortest sequences of factor colors connecting two nodes. Since we are not interested in the paths but whether the paths are identical or not, these sets might as well be represented as colors. Note that, in Fig. 5.3 we assume identical factors for simplicity. Thus in this case the sets of path colors reduce to distances. In the general case, however, we compare the full paths, i.e., the sequence of factor colors. We now have to consider the vector d_3 of shortest-paths distances to X_3, cf. Fig. 5.3**(D)**, and refine the initial supernodes correspondingly. Recall that the prior lifted network can be encoded as the vector $l = (0,0,1,1,0,0)$ of node colors. Now we refine the nodes by

(1) $l \oplus d_3$, the element-wise concatenation of two vectors, and

(2) viewing each resulting number as a new color:

$$\begin{aligned}
&(0,0,1,1,0,0) \oplus (1,1,0,1,2,2) \\
&=_{(1)} (01,01,10,11,02,02) \\
&=_{(2)} (0,0,1,2,3,3)
\end{aligned}$$

This results in the lifted network for $P(X|x_3)$ as shown in Fig. 5.3**(C)**. Thus, we can directly update the prior lifted network in linear time without taking the detour through running CP on the ground network. Given the pairwise shortest-path distances of all nodes, we group the nodes based on their shortest-path distance to the new evidence node. The supernodes split up and form new supernodes where each node has the same shortest-distance to the evidence node.

Now, let us compute the lifted network for $P(X|x_4, x_3)$. Essentially, we proceed as before: compute $l \oplus (d_3 \oplus d_4)$. However, the resulting network might be suboptimal. It assumes $x_3 \neq x_4$ and, hence, X_3 and X_4 cannot be in the same supernode. For $x_4 = x_3$, they could be placed in the same supernode, if they are in the same supernode in the prior network. This can be checked by $d_3 \odot d_4$, the element-wise sort of two vectors. In our case, this yields

$$l \oplus (d_3 \odot d_4) = l \oplus l = l \ .$$

Conditioning on x_3, respectively $x4$, is asymmetric. When the evidence is put on both together, however, we obtain the prior lifted network which is maximally compressed. In general, we compute

$$l \oplus (\bigoplus_s (\bigoplus_v d_{s,v})) \ ,$$

where

$$d_{s,v} = \bigodot\nolimits_{i \in s : x_i = v} d_i \ ,$$

s and v are the supernodes and the truth value respectively. For an arbitrary network, however, the shortest-paths might be identical although the nodes have to be split, i.e., they differ in a longer path, or in other words, the shortest-paths of other nodes to the evidence node are different. Consequently, we iteratively apply the shortest-path sequence lifting. Let SN_S denote the supernodes given the set S as evidence. By applying the shortest-path procedure we compute $SN_{\{X_1\}}$ from $SN_\emptyset$. This step might cause initial supernodes to be split into newly formed supernodes. To incorporate these changes in

Algorithm 6: Shorted-Path Sequence Lifting

```
  Input: Set of supernodes $S$, shortest-path sequences $D$
  Output: Adapted set of supernodes $S'$
1 $S' = \emptyset$;
2 for $S_i \in S$ do
3     for $X \in S_i$ do
4         if not exists $S_{i,D(X,Z)}$ then
5             $S_{i,D(X,Z)} = \{\}$;
6         Add $X$ to $S_{i,D(X,Z)}$;
7 return $S'$
```

the network structure the shortest-path sequence lifting procedure has to be iteratively applied. Thus in the next step we compute $SN_{\{X_1\}\cup\Gamma_{X_1}}$ from $SN_{\{X_1\}}$, where Γ_{X_1} denotes the changed supernodes of the previous step. This procedure is iteratively applied until no new supernodes are created. This essentially sketches the proof of the following theorem.

Theorem 5.3.1. *Given the shortest-path colors among all nodes and the prior lifted network, computing the lifted network for $P(X|X_i, \ldots, X_1)$, $i > 0$, takes $O(n \cdot s)$, where n is the number of nodes and s is the number of supernodes. Running modified BP produces the same results as running LBP on the original model.*

Proof. For a Graph $G = (V, E)$, when we set new evidence for a node $X \in V$ then for all nodes within the network the color of node X in the $CCTs$ is changed. If two nodes Y_1, $Y_2 \in V$ were initially clustered together (denoted as $sn_0(Y_1) = sn_0(Y_2)$), i.e., they belong to the same supernode, they have to be split if the $CCTs$ differ. Now we have to consider two cases: If the difference in the $CCTs$ is in the shortest path connecting X with Y_1 and Y_2, respectively, then shortest-path sequnece lifting directly provides the new clustering. If the coloring along the shortest paths is identical the nodes' $CCTs$ might change in a longer path. Since $sn_0(Y_1) = sn_0(Y_2)$ there exists a mapping between the paths of the respective $CCTs$. In particular $\exists Z_1, Z_2$, s.t. $sn_0(Z_1) = sn_0(Z_2)$ from a different supernode, i.e.,

$$sn_0(Z_i) \neq sn_0(Y_j) \;\; i, j \in \{1, 2\}$$

and

$$Y_1, \ldots, \underbrace{Z_1, \ldots, X}_{\Delta_1} \in CCT(Y_1) \; ,$$

$$Y_2, \ldots, \underbrace{Z_2, \ldots, X}_{\Delta_2} \;\in CCT(Y_2)$$

$\Delta_1 \in CCT(Z_1) \neq \Delta_2 \in CCT(Z_2)$ are the respective shortest paths for Z_1 and Z_2. Thus, by iteratively applying shortest-path sequence lifting as explained above, the evidence propagates through and we obtain the new clustering. □

Algorithm 7: Lifted Conditioning

```
input: A factor graph $G$, list of query variables $L$
output: List of clamped variables and values
1  Compress $G$ to obtain $\mathfrak{G}$;
2  Calculate distances $D$ in $G$;
3  Initialize variable list $V = \emptyset$;
4  while $L \neq \emptyset$ do
5      Run lifted inference on $\mathfrak{G}$ to obtain marginals;
6      Pick next variable $X_i$ and value $x_i$ to clamp;
7      Clamp $X_i$ to $x_i$;
8      Adapt the lifting of $\mathfrak{G}$ using Shorted-Path Sequence Lifting;
9      Add $X_i$ and $x_i$ to $V$;
10     remove $X_i$ from $L$;
11 return $V$
```

Furthermore, because shortest-path lifting is linear in the number of nodes, as shown in Alg. 6, and computing the shortest-path distances takes $O(n{\cdot}l)$, the following theorem holds.

Theorem 5.3.2. *The complexity of computing all lifted networks for conditioning using shortest-path lifting is $O(k \cdot n \cdot (l + s))$, where n is the number of nodes, where n is the number of supernodes, and l is the length of the longest path without loop.*

This can be quite fast. In worst case there is not a big difference to naïvely relifting in each iteration, in practice, however, the runtime reduces significantly, as our experimental evaluation will show. If, however, lifting does not pay off at all, computing the pairwise distances in the very first iteration may produce an overhead.

Alg. 7 summarizes the conditioning procedure employing the shortest-path lifting. For a given list of query variable L in the graph G the algorithm first lifts the graph G (**line 1**) and distances D (**line 2**). Now all variables in L have to be processed one by one. For every variable X_i we perform inference on the current model (**line 5**), read the desired marginals (**line 6**) and clamp the variable to the desired state (**line 7**). Now the lifting has to be adapted to the new evidence $X_i = x_i$ using shortest-path lifting (**line 8**). This is repeated until all query variables are processed.

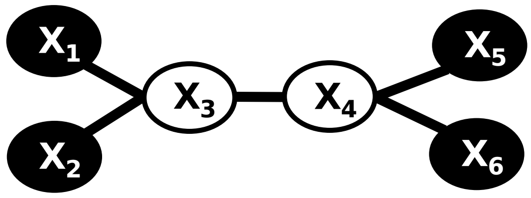
5.4 On Finding a Conditioning Order

Clearly, CLLBP will be most efficient for estimating the probability of a joint state when it produces the smallest lifted networks. This calls for the task of finding the most efficient conditioning order. This question is different from the more common question of finding highly accurate orders. The latter question is an active research area already for the ground case, see e.g. [47], and is also related to the difficult question of convergent LBP variants ([110]).

Here, we provide a generically applicable strategy based on the nodes' shortest-path colors to all other nodes. So, in addition to the hardness of inference, which we solve only approximately runnning LLBP, there is the hardness of finding the best conditioning order. Consequently, we propose a greedy heuristic to select it. The heuristic is based on the nodes' shortest-path sequences to all other nodes. That is, we start from an empty conditioning set. Then, we heuristically add nodes that have the smallest number of unique paths to all other nodes. We keep adding nodes until all nodes have been selected.

That is, in each conditioning iteration, we add the node that has the smallest number of unique paths to all other nodes and, if possible, is a member of a supernode of one of the already clamped nodes. Intuitively, we select nodes that are expected to create the smallest number of splits of existing supernodes in each iteration. Therefore, we call it *min-split*. It somehow resembles the *min-fill* heuristic, see e.g. [17] and references in there.

Example 5.4.1. *Suppose we would like to find the right node to condition on for our previous example shown in Fig. 5.3. To determine the best node to condition on we need the graph with its initial coloring and the shortest-path distances, as shown below.*



	X_1	X_2	X_3	X_4	X_6	X_6
X_1	0	2	1	2	3	3
X_2	2	0	1	2	3	3
X_3	1	1	0	1	2	2
X_4	2	2	1	0	1	1
X_5	3	3	2	1	0	1
X_6	3	3	2	1	1	0

Looking at the matrix of pairwise distances we see that X_1, X_2, X_5, and X_6 have four distinct values of distances to other variable nodes, whereas X_1 and X_2 only have two. Thus, one the two variables would be the choice to condition on, as they are expected to produce the smallest number of splits.

Although this increases the running time — each conditioning iteration now has an additional $O(n^2)$ step — our experiments show that there are important cases such as computing pairwise joint marginals where the efficiency gains achievable due to a better lifting can compensate this overhead.

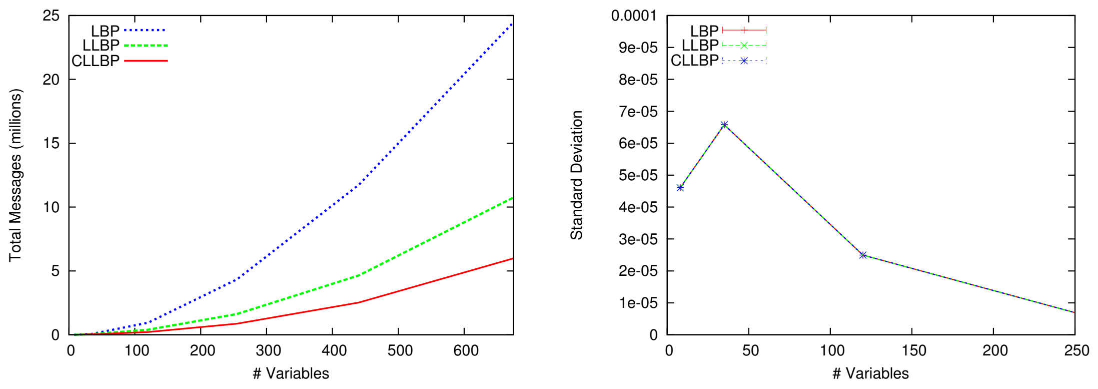


Figure 5.4: Pairwise Probability Estimates: **(Left)** Comparison of the total number of messages sent for LBP, Lifted LBP and "min-split" order CLLBP for "Friends-and-Smokers" MLNs (including clustering messages for LLBP and CLLBP). **(Right)** The Standard Deviation of the error compared to the exact solution computed using the Junction Tree Algorithm.

Evaluation

Our intention here is to answer the following questions:

(Q5.1) Can CLLBP improve the performance compared to naïvely running LLBP and LBP?

(Q5.2) Can we reduce the size of lifted networks produced by using the min-split heuristic?

We implemented CLLBP and its variants in Python and using LIBDAI library [109] and evaluated the algorithms on a number of Markov logic networks.

Joint Marginals

In our first experiment, we compared CLLBP to naïvely running LLBP, i.e., lifting the network each time from scratch, and LBP for computing pairwise probabilities. We generated the "Friends-and-Smokers" Markov logic network [146] with 2, 5, 10, 15, 20, and 25 people, resulting in networks ranging from 8 to 675 nodes. Fig. 5.4 (**left**) shows the results for varying network sizes for LBP, lifted LBP (LLBP) and conditioned LLBP (CLLBP). As one can see, lifting already shows an improvement compared to LBP and additionally the shortest-path lifting clearly pays out in terms of the total messages sent (including CP and shortest-path messages).

Moreover, the accuracy estimates are surprisingly good and confirm Welling and Teh [163]; Fig. 5.4 (**right**) shows the Standard Deviation of the difference compared to the exact solution computed using the Junction Tree (JT). The maximal error we got was below 10^{-4}. Note, however, that running JT with more than 20 persons was impossible due to memory restrictions.

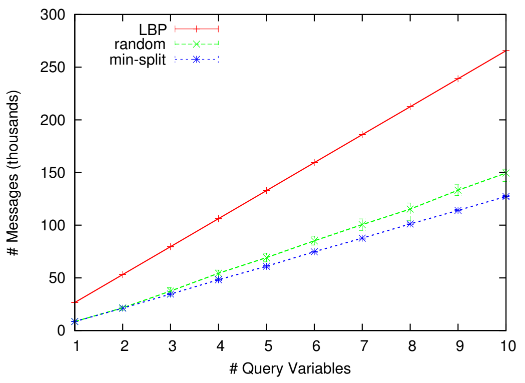


Figure 5.5: Number messages sent for computing joint marginals of varying size for LBP, ”random” and ”min-split” order CLLBP.

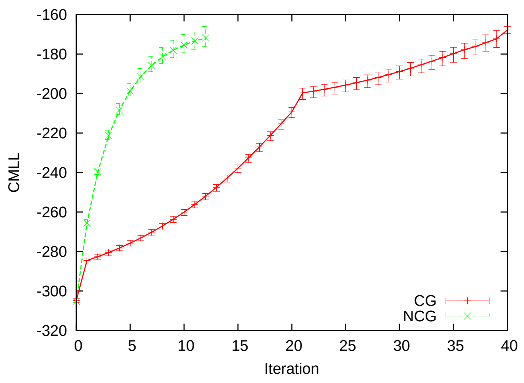


Figure 5.6: Learning curves for ”Friends-and-Smokers” MLN. Optimization using clause covariances shows faster convergence.

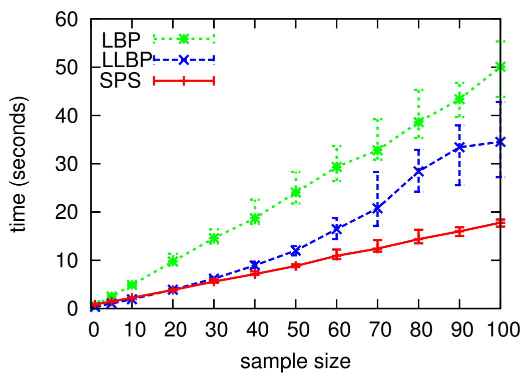
In our second experiment we investigated CLLBP for computing joint marginals. For the ”Friends-and-Smokers” MLN with 20 people we randomly chose 1, 2, . . ., 10 ”cancer” and ”friends” nodes as query nodes. The joint state was randomly chosen. The results are averaged over 10 runs. Fig. 5.5 shows the cumulative number of messages (including CP messages) for LBP and CLLBP using the *min-split* heuristic and a *random* conditioning order. As one can see, *min-split* is indeed better. By choosing the order following our heuristic the cumulative number of supernodes and in turn messages is reduced compared to a random elimination order.

Finally, we learned the parameters for the ”Friends-and-Smokers” MLN with 10 people, maximizing the conditional marginal log-likelihood (CMLL). To do so, we sampled 5 data cases from the joint distribution. We compared conjugate gradient (CG) optimization using Polak-Ribiere with Newton conjugate gradient (NCG) optimization using the covariance matrix of MLN clauses computed using CLLBP. The gradient was computed as described in Sec. 2.2.2 but normalized by the number of groundings of each clause. The results summarized in Fig. 5.6 confirm that information about dependencies among clauses is indeed useful: the second order method exhibits faster convergence.

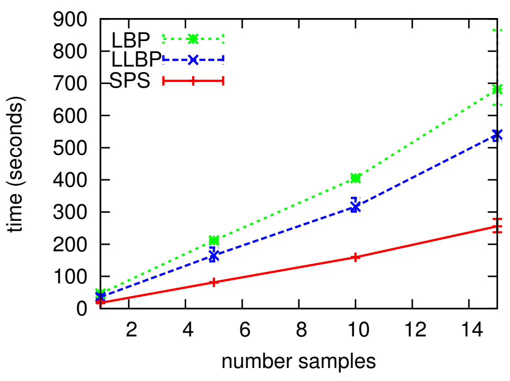
Lifted Sampling

Here we investigated LBP, LLBP and CLLBP using the shortest path sequences for sampling a joint configuration over a set of of variables sequentially, i.e., a sequence of one-variable samples conditioned on a subset as shown in Alg.7. We drew samples from the ”Friends-and-Smokers” dynamic MLN with 10 people over 10 time steps as shown in Sec. 21.

In our first experiment, we randomly chose 1, 5, 10, 20, 30, . . ., 100 “cancer” nodes over all time steps and drew samples from the joint distribution. As one can see in Fig. 5.7(a), LLBP already provides significant improvements compared to LBP, however, as the sample size increases, the speed-up decreases. The more evidence we have



(a) Varying sample size



(b) Varying number of samples

Figure 5.7: (L)LBP-guided sampling.

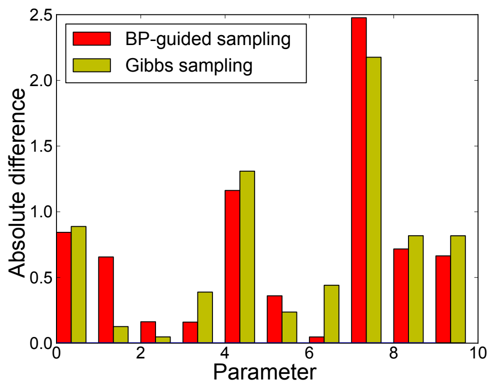


Figure 5.8: Absolute difference of the learned parameter from the parameters of the original distribution the samples were drawn.

in the network, the less lifting is possible. CLLBP lifting with using the shortest path sequences (SPS) has the additional gain in runtime as we do not need to perform the lifting in each step from scratch.

In our second experiment we fixed the sample size to 100, i.e., we sampled from the joint distribution of all $cancer(X, t)$ for all persons X in the domain and all time steps t. We drew 1, 5, 10 and 15 samples and the timings are averaged over 5 runs. The results are shown in Fig. 5.7(a). We see that LLBP is only slightly advantageous compared to LBP, as the sample size is 100, especially in the later iterations we have lots of evidence and long chains to propagate the evidence. Repeatedly running CP almost cancels the benefits. CLLBP on the other hand, shows significant speed-ups compared to both LBP and LLBP. We see that shortest path sequences lead to faster inference in terms of messages sent and in total CPU time.

To evaluate the quality of the samples, we drew 100 samples of the joint distribution of all variables using the LBP-guided approach and Gibbs sampling respectively. We learned the parameters of the model maximizing the conditional marginal log-likelihood (CMLL) using scaled conjugate gradient (SCG). Fig. 5.8 shows the absolute difference of the learned weights from the model the datacases were drawn. As one can see parameter learning with LBP-guided samples performs as good as with samples drawn by Gibbs sampling. The root-mean-square error (RMSE) for the LBP parameters was 0.31 and for Gibbs parameters 0.3.

All experiments together clearly answer **(Q5.1)** and **(Q5.2)** affirmatively.

6

Multiple Inference Tasks

So far we have shown the improvements that can be made to LBP and GaBP, when applied to inference in graphical models containing structural symmetries.

In this chapter, we investigate the question whether we can exploit symmetries across multiple evidence cases not only for lifting but also in the inference stage, thus efficiently solving multiple inference tasks, i.e.,

> *compute the marginal distribution* $P(X_i|E_j)$ *for a subset of nodes* $X_i \in \mathbf{X_A}$, $\mathbf{X_A} \subseteq \mathbf{X}$ *and multiple evidence cases* $E_j \in E$.

Message-passing algorithms have proven empirically effective for solving hard and computationally intensive problems in a range of important and real-world AI tasks. This opens up a wide range of applications for message-passing algorithms, and is beneficial for several important AI tasks such as the ones we have introduced earlier, namely, SAT, model counting, and linear programming. Further important tasks are Kalman filters, the workhorse of robotics, and the well-known problem of web page ranking. Viewing the web as a graph where nodes are web pages and directed edges are hyperlinks between web pages, a highly successful web page ranking metric computed from this graph is the PageRank [23].

The algorithm can be described as the following random walk on the graph: the walker starts at a uniformly chosen random vertex of the graph, then with probability $1-\alpha$ it follows a uniformly selected, random out-leading edge from the vertex, and with probability α it teleports to a uniformly selected, random vertex of the graph, where $0 < \alpha < 1$. Converting this graph to an ergodic Markov chain, the PageRank of a web page node v is the (limit) stationary probability that a random walker is at v. Recently, Bickson *et al.* have shown how to compute the PageRank distributively and efficiently using Gaussian belief propagation (GaBP) [15]. As for the discrete case, however, there are many improvements that can be made to (Ga)BP, especially when applied to graphical models containing structural symmetries.

The core matrix inversion computation in both PageRank and Kalman filtering can be naïvely solved by combining [144] GaBP approach to solving linear systems with our color passing approach for LBP, yielding Lifted GaBP (Ch. 3.3). In fact, this novel LGaBP approach — as we will show — already results in considerable efficiency gains. However, we can do considerably better. Essentially, the computations in personalized PageRanks and Kalman filters require the LGaBP solution of several linear systems with only small changes to the evidence. In turn repeatedly constructing the lifted network for each new inference can be extremely wasteful, because the evidence changes little from one inference to the next. Hence, a less naïve solution exploits recent efficient approaches for updating the structure of an existing lifted network with small changes to the evidence, as we have shown in Ch. 5.

While *conditioned lifted LBP* is efficient for updating the structure of an existing lifted network with incremental changes to the evidence, it still needs to construct a separate lifted network in the inference stage for each evidence case and run a modified message passing algorithm on each lifted network separately. Consequently, symmetries across the inference tasks are not exploited for inference.

An investigation of the question of whether we can achieve additional efficiency gains by exploiting symmetries across the inference tasks leads to our main contribution of this chapter: *multi-evidence lifting*.

In multi-evidence lifting, we first construct the ground networks for all inference tasks. We then run color passing on the union of these networks to compute the joint lifted network that automatically exploits symmetries across inference tasks. Finally, we run a modified message passing algorithm on the joint lifted network that simulates message passing on each ground network in parallel. Intuitively, this sacrifices space complexity for a lower time complexity, and the naïve approach of lifting the joint network will not scale well to large problem sizes. Consequently, we develop an efficient sequential lifting variant that computes the joint lifted network by considering evidence sequentially rather than jointly.

As our experiments show, multi-evidence LGaBP[1] and its sequential variant can yield significantly faster inference than naïve LGaBP and GaBP on synthetic problems and the real-world problems of PageRank and Kalman filter computation.

6.1 Multi-Evidence Lifting

Both applications considered here — lifted PageRank and Kalman filtering — require the inversion of very large, structured real-valued matrices. As shown previously, this task can be reduced to the problem of solving several linear systems with GaBP — more precisely, calling GaBP multiple times, each time with a different evidence case.

Returning to our running example, inverting $\mathbf{A}$ from Eq. (3.4) using GaBP results in

[1]We focus here on matrix inversion, but multi-evidence lifting is generally applicable, also to discrete domains.

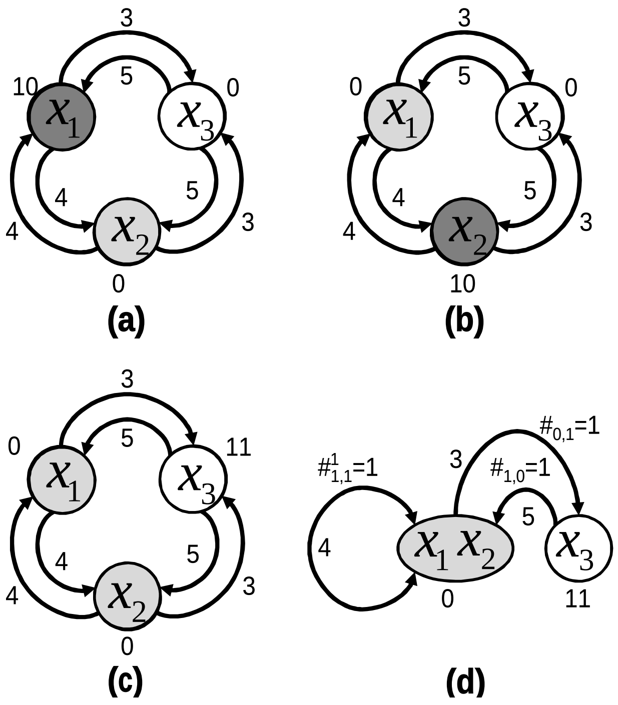
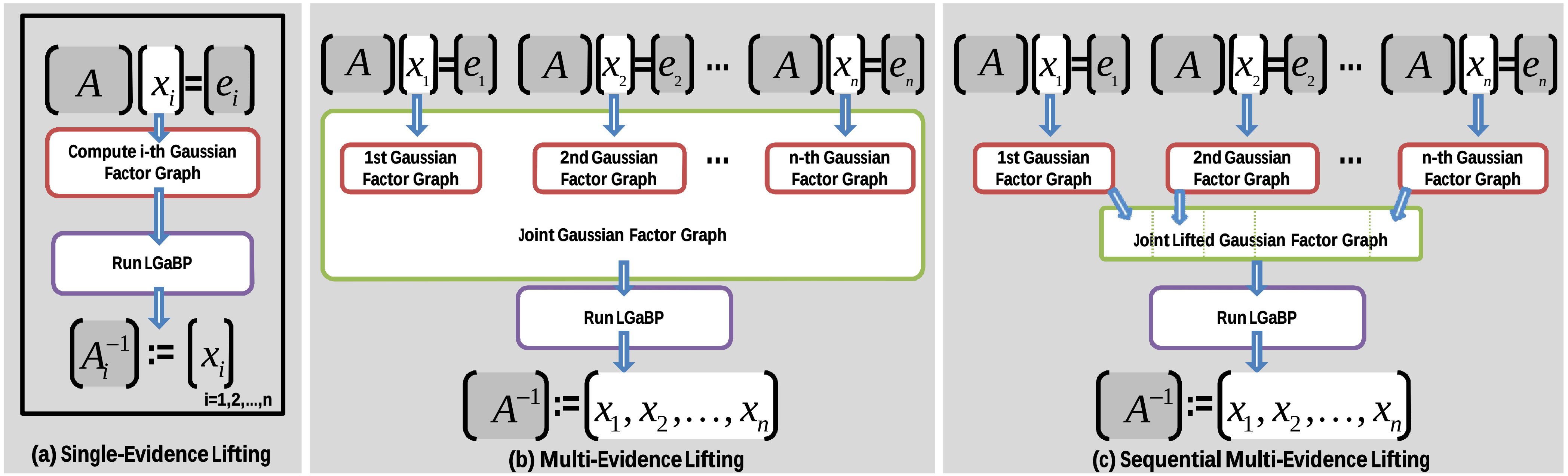


Figure 6.1: Lifted graphical models produced when inverting $\mathbf{A}$ in Eq. (3.4) using LGaBP. An edge from i to j encodes potential ψ_{ij}. The ϕ potentials are associated with the nodes. **(a)** Colored network when computing $\mathbf{A}\mathbf{x}_1 = \mathbf{e}_1$. All nodes get different colors; no compression and lifted inference is essentially ground. **(b)** Colored network for $\mathbf{A}\mathbf{x}_2 = \mathbf{e}_2$. Again no compression. Note, however, the symmetries between (a) and (b). **(c)** Colored network for $\mathbf{A}\mathbf{x}_3 = \mathbf{e}_3$. Nodes x_1 and x_2 get the same color and are grouped together in the lifted network **(d)**.

three graphical models, each of size $3/9$ (nodes/potentials). Given the recent success of LLBP approaches, we ask “can we do better?” A first attempt to affirmatively answer the question is to simply replace GaBP with LGaBP for each evidence case $\mathbf{e_i}$ as illustrated in the three lifted graphical models of Figs. 6.1(a–d). In general, this *single-evidence lifting* approach is illustrated in Fig. 6.2(a). Due to lifting, we can hope to greatly reduce the cost of inference in each iteration.

For our running example, this results in two ground networks both of size $3/9$ and one lifted network of size $2/5$ shown respectively in Figs. 6.1(a,b) and (d). Thus, the total size (sum of individual sizes) drops to $8/23$ compared to the GaBP case. This LGaBP approach — as we will show in our experiments — can already result in considerable efficiency gains. However, we can do even better. Repeatedly constructing the lifted network for each new evidence case can be wasteful when symmetries across multiple evidence cases are not exploited (e.g., the common 0’s amongst all $\mathbf{e}_i$). Compare, for example, the graphs depicted in Fig. 6.1(a) and (b). These two networks are basically symmetric which is not exploited in the single-evidence case.They stay essentially ground if they are processed separately. To overcome this, we propose *multi-evidence lifting* (as illustrated in Fig. 6.2(b) for inverting a matrix):



A x_i = e_i
Compute i-th Gaussian Factor Graph
Run LGaBP
A_i^{-1} := x_i
i=1,2,...,n
(a) Single-Evidence Lifting
A x_1 = e_1 A x_2 = e_2 ... A x_n = e_n
1st Gaussian Factor Graph
2nd Gaussian Factor Graph
...
n-th Gaussian Factor Graph
Joint Gaussian Factor Graph
Run LGaBP
A^{-1} := $x_1, x_2, \ldots, x_n$
(b) Multi-Evidence Lifting
A x_1 = e_1 A x_2 = e_2 ... A x_n = e_n
1st Gaussian Factor Graph
2nd Gaussian Factor Graph
...
n-th Gaussian Factor Graph
Joint Lifted Gaussian Factor Graph
Run LGaBP
A^{-1} := $x_1, x_2, \ldots, x_n$
(c) Sequential Multi-Evidence Lifting

Figure 6.2: Inverting a matrix $\mathbf{A}$ using multi-evidence lifting. (a) Single-evidence lifting runs LGaBP solving $\mathbf{A}\mathbf{x}_i = \mathbf{e}_i$ for each $i = 1, 2, \ldots, n$ separately; $\mathbf{e}_i$ denotes the ith basis vector; thus LGaBP computes each column vector of $\mathbf{A}^{-1}$ separately. (b) Multi-evidence lifting runs LGaBP on the joint graphical model of $\mathbf{A}\mathbf{x}_i = \mathbf{e}_i$ for $i = 1, 2, \ldots, n$; thus LGaBP computes $\mathbf{A}^{-1}$ for all e_i at once. (c) Sequential multi-evidence lifting directly builds the lifted joint graphical model in a sequential fashion, avoiding cubic space complexity

> *compute the graphical models of each evidence case, form their union, and run LGaBP on the resulting joint graphical model.*

This automatically employs the symmetries within and across the evidence cases due to Theorem 3.3.2. In our running example, we get a *single* lifted graph of size $5/14$. Intuitively, multi-evidence lifting only produces (in this example) one of the two ground networks and hence consists of the union of the networks shown in Figs. 6.1(a) and (d). This is clearly a reduction compared to LGaBP's size of $8/23$.

However, there is no free lunch. Multi-evidence lifting sacrifices space complexity for a lower time complexity. Inverting a $n \times n$ matrix may result in a joint ground graphical model with n^2 nodes (n nodes for each of the n systems of linear equations) and $O(n^3)$ edges ($O(n^2)$ edges for each of the n systems of linear equations; edges are omitted if $\mathbf{A}_{ij} = 0$). This makes multi-evidence essentially intractable for large n. For instance, already for $n > 100$ we have to deal with millions of edges, easily canceling the benefits of lifted inference. We develop an efficient sequential multi-evidence lifting approach that computes the joint lifted network by considering one evidence case after the other (Fig. 6.2(c)).

Indeed, one is tempted to employ *conditioned LLBP*, as just introduced in Ch. 5, for updating the structure of an existing lifted network with incremental changes to the evidence to solve the problem. While employing the symmetries in the graphical model across multiple evidence cases for the lifting, in the inference stage they need to construct a separate lifted network for each evidence case and run a modified message passing algorithm on each lifted network separately. Thus, symmetries across evidence cases are missed.

6.2 Sequential Multi-Evidence Lifting

We seek a way to efficiently construct the joint lifted network while still being able to lift across multiple evidence cases. To do this, we can modify sequential single-evidence lifting to the multi-evidence case. Ch. 5 gives a clear characterization of the core information required for sequential clamping for lifted message passing, namely the shortest-paths connecting the variables in the network which resemble the computation paths along which the nodes communicate in the network.

Recall that to be able to adapt the lifted network for incoming evidence in the single-evidence case, we compute in a first step the set of shortest paths connecting any two nodes in the graph. Now, when there is new evidence for a node, the adapted lifted network is computed as a combination of the nodes' initial coloring, i.e., the lifting without evidence, and the set of shortest paths to the nodes in our evidence.

Algorithm 8 describes how we can adapt this to the multi-evidence case. Intuitively, we would like to view each $\mathbf{A}\mathbf{x}_i = \mathbf{e}_i$ ($i = 1, 2, \ldots, n$) as conditioning an initial network on node i and efficiently compute its contribution to the resulting lifted joint network directly from the initial one. But what should be the initial network? It is not provided by the task itself. Thus, to be able to efficiently find the lifted network structure, we propose to introduce an additional system of linear equations, namely the one with no

Algorithm 8: Sequential Multi-Evidence Lifted GaBP

```
   Input: Matrix $\mathbf{A}$
1  Construct network $G_0$ for $\mathbf{Ax} = \mathbf{0}$;
2  Compute path color matrix $\mathbf{PC}$ on $G_0$;
3  Lift $G_0$ to obtain lifted network $H_0$;
4  foreach unit vector $e_i$ do
5      colors($H_i$) = colors($H_0$);
6      evidence = $\{i{:}A_{ii}\}$;
7      repeat
8          colors($H_i$) = newColor ($H_0$, $PC$, evidence);
9          Add nodes that have changed color to evidence;
10     until colors($H_i$) does not change;
11     if $H_i$ is previously unseen then
12         Add $H_i$ to joint lifted network $H_{\{1,..,i-1\}}$
13     else
14         Bookmark the corresponding index $j$
15 Run modified GaBP on joint lifted network $H_{\{1,..,n\}}$;
16 return $\mathbf{X} = (x_1, x_2, \ldots, x_n)$, the inverse of $\mathbf{A}$;
```

evidence: $\mathbf{Ax} = \mathbf{0}$ (**Alg. 8, line 1**). Indeed, it is not needed in the inversion task per se but it allows us to significantly speed up the multi-evidence lifting process. Each $\mathbf{e}_i$ only differs from $\mathbf{0}$ in exactly one element so it serves as the basis for lifting all subsequent networks. To do this, we compute the path colors for all pairs of nodes (**line 2**) to know how it affects the other nodes when conditioning on a variable i and the initial lifting without evidence (**line 3**). The lifted graph H_i for the ith system of linear equations $\mathbf{Ax}_i = \mathbf{e}_i$ can now be adaptively computed by combining the initial lifting and the path colors to node i (**line 8**). The combination is essentially an element-wise concatenation of the two respective vectors.

Example 6.2.1. *Consider computing the lifted network for* $\mathbf{Ax_3} = \mathbf{e_3}$ *for our running example. When we have no evidence the nodes* x_1 *and* x_2 *are clustered together, and* x_3 *is in a separate cluster. Thus, we obtain an initial color vector* $C = (0, 0, 1)$. *Since we want to compute the network conditioned on* x_3 *we have to combine this initial clustering* C *with the path colors with respect to* x_3*,*

$$PC_{x_3} = (\{3, 5\}, \{3, 5\}, \emptyset) = (0, 0, 1) \ .$$

To obtain the lifted network conditioned on x_3 *we have to (1) do an element-wise concatenation (in the following depicted by* $\oplus$*) of the two vectors and (2) interpret the result as a new color vector:*

$$H_3 = (0, 0, 1) \oplus (0, 0, 1) =_{(1)} (00, 00, 11) =_{(2)} (2, 2, 3) \ .$$

Since only the shortest paths are computed, adapting the colors has to be performed iteratively to let the evidence propagate as in Ch. 5.

Moreover, we can implement a type of memoization when we perform the lifting in this fashion. Because we know each resulting lifted network H_i in advance, we can check whether an equivalent lifted network was already constructed: if the same color pattern exists already, we simply do not add H_i and instead only bookmark the correspondence of nodes (**line 11-14**) [1]. This does not affect the counts at all and, hence, still constructs the correct joint lifted network. This argument together with the correctness of the sequential single-evidence lifting and multi-evidence lifting effectively proves the correctness of sequential multi-evidence lifting:

Theorem 6.2.2. *Sequential multi-evidence lifting computes the same joint lifted model as in the batch case. Hence, running the modified GaBP on it produces the same marginals.*

Proof. In the following let G_i denote a factor graph, $\mathfrak{G}_i$ its corresponding lifting and H_i the shortest-paths lifting. Sequential multi-evidence lifting first lifts the model without evidence to obtain $H_0 = \mathfrak{G}_0$, then adapts this lifting to all the evidence cases we have, i.e., to lifted networks $H_1, \ldots, H_n$. There are two differences to the case of jointly lifting the whole model, namely (i) each lifted network $H_i, i = 1, \ldots, n$, is obtained from H_0 using the shortest-paths and the new evidence and (ii) the joint lifted network is built sequentially.

Th. 5.3.1 proves the first part, that is, lifting the networks using *shortest-paths lifting* and adapting the lifted networks H_i from H_0 we obtain the same results as lifting from scratch, that is, $H_i = \mathfrak{G}_i, i = 1, \ldots, n$.

We prove the second part, $\mathfrak{G} = H$, by contradiction. Suppose the two lifted networks $\mathfrak{G}$ and H are different. Then there exists a variable node X that is in supernode $\mathfrak{X}_1$ in $\mathfrak{G}$ and in supernode $\mathfrak{X}_2$ in H, $\mathfrak{X}_1 \neq \mathfrak{X}_2$; or similarly for some superfactor $\mathfrak{f}$.

Thus, there has to be a difference in the shortest paths of X. In the respective $CCTs$ there has to be a path in CCT_H that does not exist in $CCT_{\mathfrak{G}}$, or vice versa. However, since $E_G \supseteq E_{G_i}$, every path from G_i is necessarily included in G and in turn $CCT_{\mathfrak{G}} \supseteq CCT_H$. So is there a path that exists in $\mathfrak{G}$ but not in H? Since the joint network G is the union of the G_i and in particular the set of edges $E = \bigcup_{i \in 1,\ldots,n} E_i$, there are no edges connecting nodes from different subnetworks and $CCT_{\mathfrak{G}} \subseteq CCT_H$. $\square$

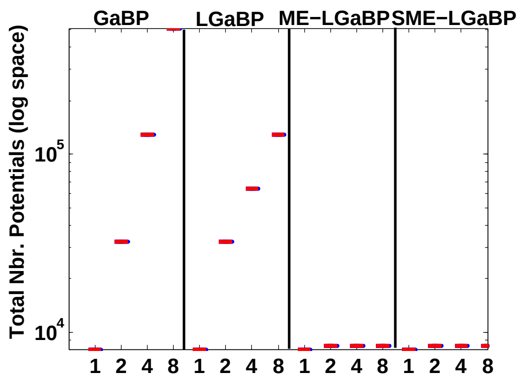
Evaluation

Our intention here is to investigate the following questions:

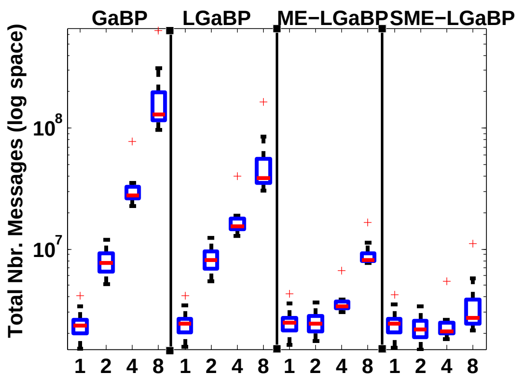
(Q6.1) Can LGaBP be faster than GaBP?

(Q6.2) Can multi-evidence lifting produce smaller inference problems than single-evidence lifting?

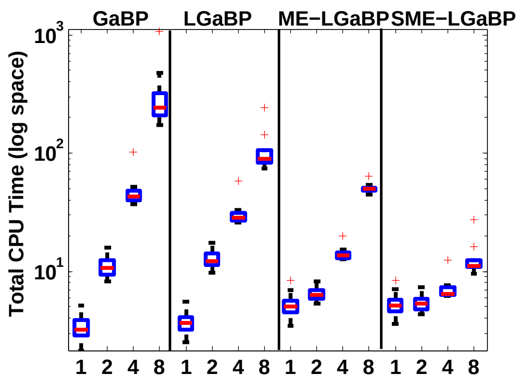
[1] This is not as hard as solving (sub)graph-isomorphisms. We only have to check whether two networks have the same colorings. If the color pattern of H_i was previously seen, the result has already been memoized.



(a) Number of potentials for varying number of Blocks



(b) Number of messages for varying number of Blocks



(c) CPU Time for varying number of Blocks

Figure 6.3: Experimental results on random matrices showing the total number of potentials, the number of messages and the CPU-time respectively. One can see, SME-LGaBP < ME-LGaBP < LGaBP < GaBP for all cases (best viewed in color)

(Q6.2) Does sequential multi-evidence lifting scale better than just multi-evidence lifting.

We implemented all variants in Python using the LIBDAI library[1] and evaluated their performances on (a) random matrices, (b) PageRank computations of graphs induced by a Markov logic network (MLN), and (c) Kalman filtering problems. The GaBP variants ran using parallel message updates, no damping and convergence threshold $\epsilon = 10^{-8}$. They all converged within ϵ of the correct solution.

Inverting Random Matrices

We generated a random matrix $\mathbf{R} \in \mathbb{R}^{20\times 20}$ with $\mathbf{R}_{ij} \in [0, 1]$ and added 10 to the diagonal to ensure non-singularity. Using $\mathbf{R}$, we constructed a diagonal matrix with

[1]http://www.libdai.org/

$1, 2, 4$ and 8 blocks. On each of the matrices we run all four algorithms measuring the number of potentials created and total messages sent (including coloring messages), and the CPU time. Figs. 6.3(a)-(c) show the results averaged over 10 random reruns. As one can see, SME-LGaBP < ME-LGaBP < LGaBP < GaBP in terms of the size of the networks measured by the number of potentials (a). In turn, the two multi-evidence approaches are also more efficient as shown by the number of messages sent (b) and the CPU-time in seconds in (c). Furthermore, note that sequential ME-LGaBP and ME-LGaBP create — as expected — the same number of potentials and significantly less than the other two GaBP and LGaBP(b).

Lifted Personalized PageRank

Recall the problem of ranking web pages already touched upon which among other important AI tasks, motivated our multi-evidence lifting approach.

In a nutshell, a Markov chain transition matrix $\mathbf{M}$ is constructed out of a given graph G. Let $\mathbf{A}$ be the $n \times n$ adjacency matrix of G, that is, $\mathbf{A}_{ij} = 1$ if there is an edge from vertex j to vertex i and zero otherwise. PageRank now constructs the probability transition matrix $\mathbf{M}$ by adding $1/n$ to all entries of $\mathbf{A}$ and renormalizes each row of $\mathbf{A}$ to sum to 1.

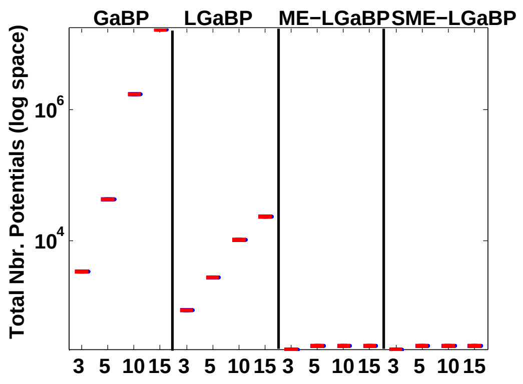
The random walk now defines a Markov chain on the vertices, with transition matrix $\alpha \cdot \mathbf{U} + (1-\alpha)\mathbf{M}$ where $\mathbf{U}$ is the transition matrix of uniform transition probabilities, i.e., $\mathbf{U}_{ij} = 1/n$ for all i, j, and α is the so called teleportation probability that the random walk follows this uniform distribution; Google usually takes $p = 0.85$. The vector of PageRank scores $\mathbf{p}$ for all vertices v is then defined to be the stationary distribution of this Markov chain:

$$\big(\alpha \cdot \mathbf{U} + (1-\alpha)\mathbf{M}\big) \cdot \mathbf{p} = \mathbf{p} \tag{6.1}$$

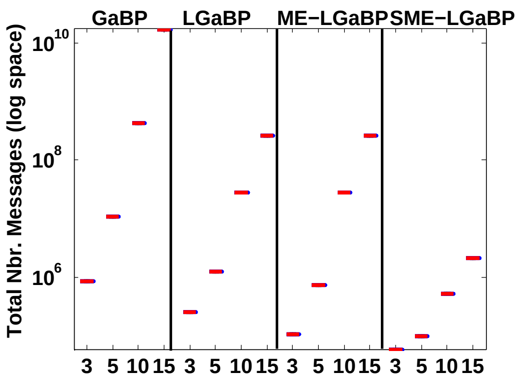
Because elements of $\alpha \cdot \mathbf{U} + (1-\alpha)\mathbf{M}$ are all strictly between zero and one and its row sums are all equal to one, the Perron-Frobenius Theorem applies to it and concludes that a nonzero solution of the equations exists and is unique to within a scaling factor. Recently, Bickson *et al.* have shown how to compute the personalized PageRank distributively and efficiently using GaBP [15]. Hence, we can also plug in LGaBP in place of GaBP to compute the PageRank in a lifted fashion.

In several PageRank applications, however, one needs to know, in addition to the rank of a given page, which pages or sets of pages contribute most to its rank. These PageRank contributions have been used for link spam detection and in the classification of web pages. The contribution that a vertex $\mathbf{v}$ makes to the PageRank of a vertex $\mathbf{u}$ is defined rigorously in terms of personalized PageRank for all vertices, i.e., for $\mathbf{v}_i = \mathbf{e}_i$, $i = 1, 2, \ldots, n$. So, we are interested in $\mathbf{PRM}_\alpha$ — the matrix whose u-th row is the personalized PageRank vector of u. The PageRank contribution of u to v is then the entry (u, v) of this matrix. This, personalized PageRank can then be computed by solving the following system of linear equations

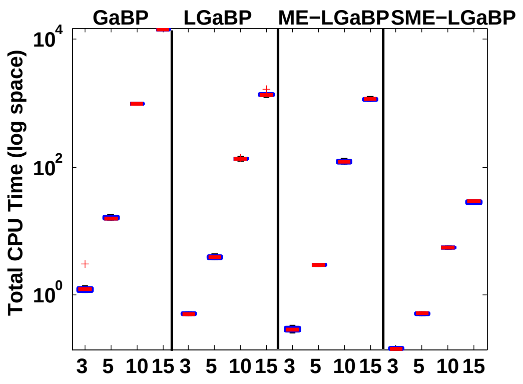
$$(\mathbf{I} - \alpha\mathbf{M})\mathbf{x} = \mathbf{v} \tag{6.2}$$



(a) Number of potentials for varying number of people



(b) Number of messages for varying number of people



(c) CPU time for varying number of people

Figure 6.4: Experimental results on PageRank computations. showing the total number of potentials, the number of messages and the CPU-time respectively. One can see, SME-LGaBP < ME-LGaBP < LGaBP < GaBP for all cases (best viewed in color)

where α trades off speed of convergence with the accuracy of the solution and $\mathbf{I}$ is the identity matrix.

We can now make use of multi-evidence lifting to compute the inverse of $(\mathbf{I} - \alpha \mathrm{M})$.

We computed $\mathbf{PRM}_{0.9}$ for the ground network induced by the *Friends & Smokers* Markov logic network [134]. We varied the number of people in the domain, namely $3, 5, 10$. Figs. 6.4 (a)–(c) show the results. The total number of (lifted) potentials (a), total numbers of messages sent (b) and the total CPU times in seconds (c) per $\mathbf{PRM}_{0.9}$ computation averaged over 10 runs are shown. Again one can see, SME-LGaBP < ME-LGaBP < LGaBP < GaBP. In particular, we see the benefit of multi-evidence lifting. The number of generated potentials saturates for more than 3 people whereas single-evidence lifting generates more and more potentials. Furthermore, we can again see the additional benefit we gain by sequentially lifting the models. Although ME-LGaBP achieves additional benefits from lifting the whole model and thereby exploiting symmetries across evidence cases, it may suffer from having to handle significantly larger networks at once. SME-LGaBP on the other hand combines lifting across evidence cases with an efficient per evidence case lifting.

Lifted Kalman Filter

Kalman Filtering is a computational tool with widespread application in robotics, financial and weather forecasting, and environmental engineering. Given observation and state transition models, the Kalman Filter (KF) recursively estimates the state $\mathbf{x} \in \mathbb{R}^n$ of a discrete-time controlled process that is governed by the linear stochastic difference equation

$$\mathbf{x}_k = \mathbf{A}\mathbf{x}_{k-1} + \mathbf{B}\mathbf{u}_{k-1} + \mathbf{w}_{k-1} \tag{6.3}$$

with a measurement $\mathbf{z} \in \mathbb{R}^m$, that is,

$$\mathbf{z}_k = \mathbf{H}\mathbf{x}_k + \mathbf{v}_k \; . \tag{6.4}$$

Here, $\mathbf{w}_k$ and $\mathbf{v}_k$ represent the process and measurement noise (respectively) and are assumed to be independent (of each other), white, and with normal density, i.e., $p(\mathbf{w}) \sim \mathcal{N}(0, \mathbf{Q})$ and $p(\mathbf{v}) \sim \mathcal{N}(0, \mathbf{R})$. The matrix $\mathbf{A}$ relates consecutive states $\mathbf{x}_{k-1}$ and $\mathbf{x}_k$. The matrix $\mathbf{B}$ relates the optional control input $\mathbf{u}$ to the state, and $\mathbf{H}$ relates the state to the measurement $\mathbf{z}$. In practice, the matrices $\mathbf{A}, \mathbf{B}, \mathbf{H}, \mathbf{Q}$ and $\mathbf{R}$ might change with each time step or measurement, however, here we assume they are constant.

Now, each step of the KF consists of two updates: compute a time and a measurement update. To compute the time update:

$$\hat{\mathbf{x}}_k^- = \mathbf{A}\hat{\mathbf{x}}_{k-1} + \mathbf{B}\mathbf{u}_{k-1} \tag{6.5}$$

$$\mathbf{P}_k^- = \mathbf{A}\mathbf{P}_{k-1}\mathbf{A}^t + \mathbf{Q} \tag{6.6}$$

Then, in the measurement updates

$$\mathbf{K}_k = \mathbf{P}_k^-\mathbf{H}^t\big(\mathbf{H}\mathbf{P}_k^-\mathbf{H}^t + \mathbf{R}\big)^{-1} \tag{6.7}$$

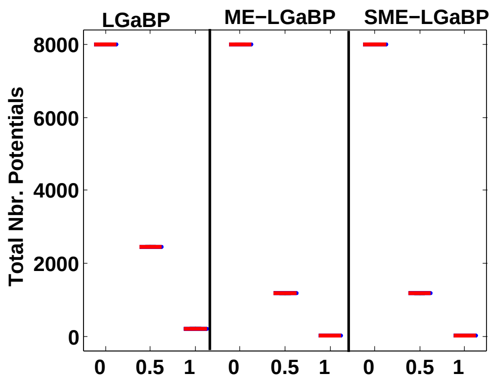
$$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_k^- + \mathbf{K}_k\big(\mathbf{z}_k - \mathbf{H}\hat{\mathbf{x}}_k^-\big) \tag{6.8}$$

$$\mathbf{P}_k = (\mathbf{I} - \mathbf{K}_k\mathbf{H})\mathbf{P}_k^- \tag{6.9}$$

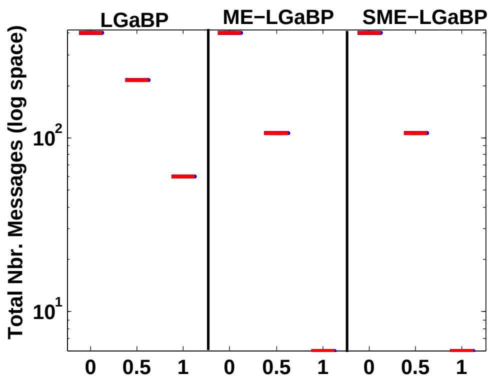
$$\mathbf{P}_k^- = \mathbf{A}\mathbf{P}_{k-1}\mathbf{A}^t + \mathbf{Q} \tag{6.10}$$

where $\mathbf{K}_k$ is often called the Kalman gain. Thus, the Kalman filter requires to invert a matrix at every time step allowing us to apply multi-evidence lifting.

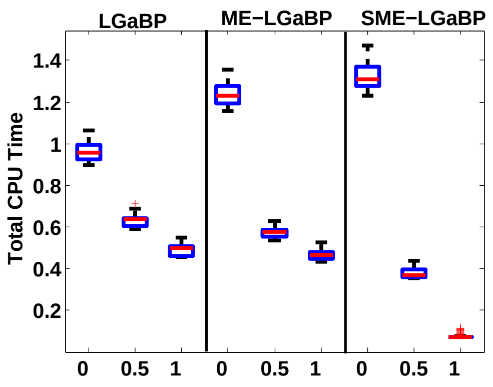
One of the challenges of tracking multiple objects is how to correctly assign the observations to the respective object of the Kalman filters. This is difficult as there may be temporary occlusions, for example, or very close measurements that makes it hard to match the new observations to the previous iteration. We formulate the association task as a linear assignment problem which identifies the desired association as the one that minimizes a summed assignment cost of pairings between blob observations and Kalman filters. Once this is accomplished multiple object tracking is a matter of updating each Kalman filter with its associated observation.



(a) Number of potentials for varying motion symmetries ratio



(b) Number of potentials for varying motion symmetries ratio



(c) Number of potentials for varying motion symmetries ratio

Figure 6.5: Experimental results for Kalman filtering with varying degree of motion symmetries. For motion symmetries ratio of 0 single-evidence lifting is faster. With higher amount of symmetries multi-evidence lifting significantly outperforms single-evidence lifting. (best viewed in color)

The task is to ensure that the overall assignment is the most likely one, given a cost function, while respecting matching constraints (see e.g. [73]). In this work we do not solve the problem and work with a simplified version.

In our experiments, we tracked 10 people randomly spread among k groups. Each group had its own (local) motion model (mm). We varied the number of groups: 1 (all people have the same mm), 5 (two people have the same mm), and 10 (everybody has its own mm). Figs. 6.5 shows (a) the total number of (lifted) potentials, (b) the total number of messages sent and (c) the CPU times in seconds averaged over all matrix inversion tasks in a Kalman filtering over 10 steps (GaBP is omitted). It clearly shows: the larger the number of groups, the lower the gain of (sequential) multi-evidence lifting. For 10 groups, when there are no mm symmetries — motion symmetries ratio 0 — across people, single-evidence lifting is faster. However, when we have symmetries across evidence cases — 5 and 1 groups, i.e., ratios of 0.5 and 1.0 — ME lifting significantly outperforms single-evidence lifting.

All experimental results together clearly answer questions **(Q6.1)**-**(Q6.3)** affirmatively.

So far and yet to come

In the first part *we tackled the problem of efficient inference in graphical models and have shown how to employ symmetries to speed up inference by loopy belief propagation.*

We focused on a number of different queries for graphical models and important AI tasks such as Boolean satisfiability and model counting, content distribution, linear programming, Markov decision processes, and Kalman filters, and showed that by compressing the graph lifted LBP *can compute marginal probabilities very efficiently.*

There are a number of approaches to speed up training in probabilistic graphical models in general. These approaches include parallelization, approximation of the marginals or the objective, or local training among others. Now that we have reached efficient inference by exploiting symmetries in the model, so why not just making use of the same techniques to improve training of relational models – which are a prominent example where symmetries abound?

Indeed, multi-evidence lifting, for example, as we introduced in the previous chapter in the context of GaBP, is also applicable in discrete cases, and we can use it to perform training of graphical models.

We performed parameter estimation – another natural case for multi-evidence lifting – in discrete domains. For the Friends & Smokers *MLN with ten people. we maximized the conditional marginal log-likelihood (CMLL) using scaled conjugate gradient (SCG) for 10 data cases sampled from the joint distribution. Our results in Table 6.1 show that we can perform lifted LBP which took 0.089 seconds for a single iteration — already a reduction compared to LBP's 0.1 sec. — but ME-LLBP exploits the additional symmetries across the data cases resulting in smaller networks and more efficient inference and in turn took only 0.049 sec.*

	#Potentials	#Messages	Time (sec.)
LBP	4600	38700	0.101
LLBP	1547	23817	0.089
ME-LLBP	**322**	**19235**	**0.049**

Table 6.1: Results for parameter estimation in discrete domains. MEL-GaBP results in a smaller overall network size (number of variables and potentials) and thus a smaller number of messages and faster inference (CPU-time).

While this is a very promising initial attempt for a lifted training approach, the results only tell half of the story. As we have learned in Sec. 2.2.2, each iteration of parameter estimation typically involves inference over two instances of the model: one without any evidence and one with evidence *which is provided by the data we train the model on. The previous chapters have shown that evidence causes harm to lifted approaches in that it may render them useless. This is exactly what happens here. The lifting only happens in the evidence free part and the data case mostly remains grounded. Unfortunately, it is not easy to exploit the symmetries in the data part and thus lifting has not been used for training yet, let alone combining symmetry exploiting approaches with traditional scaling techniques.*

In the second part *we will show how to overcome the limitations that prevented the use of lifting in training and we will show how the symmetries can be used to efficiently train relational graphical models.*

After introducing lifted online training of relational models in Ch. 7, we will show how to further scale up the lifted training with a fully parallel lifted inference approach. Therefore we introduce MapReduce lifting for loopy belief propagation in Ch. 8.

Part II

Symmetries in Training

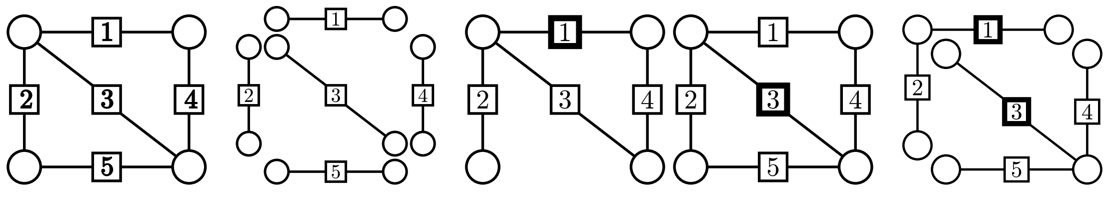
7

Lifted Online Training of Relational Graphical Models

When training a relational model for a given set of observations the presence of evidence on the variables mostly destroys the symmetries. This makes lifted approaches virtually of no use if the evidence is asymmetric. In the fully observed case, this may not be a major obstacle since we can simply count how often a clause is true. In relational learning, however, we typically face a single large set of inter-connected facts – the *mega-example* – which unfortunately is incomplete in many real-world domains, i.e., the truth values of some ground atoms may not be observed. For instance, in medical domains, a patient rarely gets all of the possible tests. In case of missing data, however, the maximum likelihood estimate typically cannot be written in closed form. It is a numerical optimization problem, and typically involves nonlinear, iterative optimization and multiple calls to a relational inference engine as subroutine.

Since efficient lifted inference is troublesome in the presence of partial evidence and most lifted approaches easily fall back to the ground case, we need to seek a way to make the learning task tractable. As we have seen in Sec. 2.2.2 one way to cope with large complex network is to decompose a factor graph into tractable pieces and to train these pieces independently and to thus exploit symmetries across multiple pieces for lifting.

Intuitively, however, this discards dependencies of the model parameters when we decompose the mega-example into pieces. Although the piecewise model helps to significantly reduce the cost of training, the way we shatter the full model into pieces greatly effects the learning and lifting quality. Strong influences between variables might get broken. Consequently, in Sec. 7.1 we propose a shattering approach that aims at keeping strong influence but still features lifting. The lifted online training approach introduced in Sec. 7.2 can greatly benefit from these tree-pieces.



(a) Orig. model (b) Depth $d = 0$ (c) Depth $d = 1$ for f_1 and f_3 (d) Trees $d = 1$

Figure 7.1: Schematic factor-graph depiction of the difference between likelihood (a), standard piecewise (b,c) and treewise training (d). Likelihood training considers the whole mega-example, i.e., it performs inference on the complete factor graph induced over the mega-example. Piecewise training normalizes over one factor at a time (b) or higher-order, complete neighborhoods of a factor (c) taking longer dependencies into account, here shown factors f_1 and f_3. Treewise training (d) explores the spectrum between (b) and (c) in that it also takes longer dependencies into account but does not consider complete higher neighborhoods; shown for tree features for factors f_1 and f_3. In doing so it balances complexity and accuracy of inference.

7.1 Relational Tree Shattering

Assume that the mega-example has been turned into a single factor graph for performing inference, cf. Fig. 7.1**(a)**. Now, starting from each factor, we extract networks of depth d rooted in this factor. A local network of depth $d = 0$ thus corresponds to the standard piecewise model as shown in Fig. 7.1**(b)**, i.e., each factor is isolated in a separate piece. Networks of depth $d = 1$ contain the factor in which it is rooted and all of its direct neighbors, Fig. 7.1**(c)**. Thus when we perform inference in such local models using say loopy belief propagation the messages in the root factor of such a network resemble the LBP messages in the global model up to the d-th iteration. Longer range dependencies are neglected. A small value for d keeps the pieces small and makes inference and hence training more efficient, while a large d is more accurate. However, it has a major weakness since pieces of densely connected networks may contain considerably large subnetworks, rendering the standard piecewise learning procedure useless. To overcome this, we now present a shattering approach that randomly grows piece patterns forming trees.

Formally, a tree is defined as a set of factors such that for any two factors f_1 and f_n in the set, there exists one and only one ordering of (a subset of) factors in the set f_1, f_2 , . . . f_n such that f_i and f_{i+1} share at least one variable, i.e., there are no loops. A tree of factors can then be generalized into a tree pattern, i.e., conjunctions of relational ”clauses” by variablizing their arguments. For every clause of the MLN we thus form a tree by performing a random walk rooted in one ground instance of that clause. This process can be viewed as a form of relational pathfinding [133].

Note that, the term *shattering* has previously been defined in a slightly different way. In the context of lifted inference, a set of first-order factors or clauses are often called shattered, if the following two conditions hold (see e.g. [40]):

Algorithm 9: RELTREEFINDING: Relational Treefinding

```
   Input: Set of clauses F, a mega example E, depth d, and discount $t \in [0, 1]$
   Output: Set of tree pieces T
   // Tree-Pattern Finding
1  Initialize the dictionary of tree patterns to be empty, i.e., $P = \emptyset$ ;
2  for each clause $F_i \in F$ do
3      Select a random ground instance $f_j$ of $F_i$;
4      Initialize tree pattern for $F_i$, i.e., $P_i = \{f_j\}$ ;
       // perform random walk in a breadth-first manner
          starting in $f_j$
5      for $f_k$ = BFS.next() do
6          if $current\_depth > d$ then break;
7          sample $p$ uniformly from $[0, 1]$ ;
8          if $p > t^{|P_i|}$ or $f_k$would induce a cycle then
9              skip branch rooted in $f_k$ in $BFS$ ;
10         else
11             add $f_k$ to $P_i$;
12     Variabilize $P_i$ and add it to dictionary $P$;
   // Construct tree-based pieces using the relational
      tree patterns
13 for each $f_j \in E$ do
14     Find $P_k \in P$ matching $f_j$, i.e., the tree pattern rooted in the clause $F_k$
       corresponding to factor $f_j$;
15     Unify $P_k$ with $f_j$ to obtain piece $T_j$ if possible and add $T_j$ to $T$ ;
16 return $T$;
```

(i) For any pair the set of groundings are either equal or disjoint, that is, they do not overlap.

(ii) For every clause, the random variables contained are either the same or can not be grounded to the same ground variable.

Example 7.1.1. *The set*

$$\{Friends(Anna, y), Friends(x, y)\}\ ,\ x, y \in Anna, Bob$$

violates condition (i) since both can be grounded to Friends(Anna, Bob) *but only the second clause can be grounded to* Friends(Bob, Anna). *The clause*

$$Friends(x, Anna) \leftrightarrow Friends(Bob, y)$$

violates condition (ii) since both variables can be grounded to Friends(Bob,Anna).

English	**First-Order Logic**	**Weight**
Only one person can be promoted	`Promotion(x)` $\Leftrightarrow$`!Promotion(y)`	2.0
A promotion comes with an increased income	`Promotion(x)` $\Rightarrow$ `HighIncome(x)`	1.5
	`Promotion(x)` $\Rightarrow$`!HighIncome(y)`	1.1

Table 7.1: Example of a Markov Logic network about promotions and a resulting raise in income. Grounding this example leads to the ground model in Fig. 7.2

In our *shattering*, however, the trees (sets of factors) may overlap. Since we do not allow partial groundings and constrained quantifiers the second condition holds.

The relational treefinding is summarized in Alg. 9. For a given set of Clauses F and a mega example E the algorithm starts off by constructing a tree pattern for each clause F_i **(lines 1-12)**. Therefore, it first selects a random ground instance f_j **(line 3)** from where it grows the tree. Then it performs a breadth-first traversal of the factor's neighborhood and samples uniformly whether they are added to the tree or not **(line 7)**. If the sample p is larger than $t^{|P_i|}$, where $t \in [0, 1]$ is a discount threshold and $|P_i|$ the size of the current tree, or the factor would induce a cycle, the factor and its whole branch are discarded and skipped in the breadth-first traversal, otherwise it is added to the current tree **(lines 8-11)**. A small t basically keeps the size of the tree small while larger values for t allow for more factors being included in the tree. The procedure is carried out to a depth of at most d, and then stops growing the tree. This is then generalized into a piece-pattern by variablizing its arguments **(line 12)**. All pieces are now constructed based on these piece patterns. For f_j we apply the pattern P_k of clause F_k which generated the factor **(lines 13-15)**.

These *tree-based pieces* can balance efficiency and quality of the parameter estimation well. To see this, let us have a look at the following example.

Example 7.1.2. *Tab. 7.1 shows an example MLN. It has three rules stating that there is only one position available and if a person gets a promotion the income will be high. Grounding this example for a domain with two constants* Anna *and* Bob *leads to the ground model in Fig. 7.2 (left). In fact, grounding the MLN would lead to more factors however for illustration purposes we assume, e.g.,* "`Promotion(Anna)` $\Leftrightarrow$`!Promotion(Bob)`" *and* "`Promotion(Bob)` $\Leftrightarrow$`!Promotion(Anna)`" *are simplified to a single factor. Fig. 7.2(center) shows the set of clauses that corresponds to the tree rooted in the factor f_3 where green colors show that the factors have been included in the piece while all red factors have been discarded. The neighborhood of the factor f_3 which corresponds to the ground clause*

`Promotion(Anna)` $\Leftrightarrow$`!Promotion(Bob)`

is traversed in a breadth-first manner, i.e., first its direct neighbors in random order. Assume we have first reached the factor f_4 =

`Promotion(Bob)` $\Rightarrow$ `HighIncome(Bob)` .

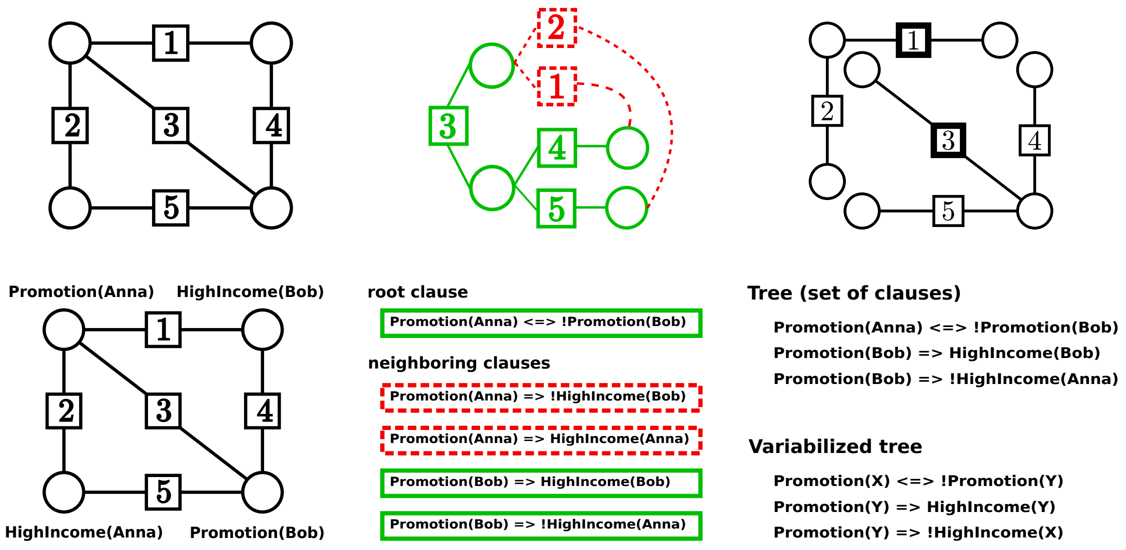

Promotion(Anna)
HighIncome(Bob)
HighIncome(Anna)
Promotion(Bob)
root clause
Promotion(Anna) <=> !Promotion(Bob)
neighboring clauses
Promotion(Anna) => !HighIncome(Bob)
Promotion(Anna) => HighIncome(Anna)
Promotion(Bob) => HighIncome(Bob)
Promotion(Bob) => !HighIncome(Anna)
Tree (set of clauses)
Promotion(Anna) <=> !Promotion(Bob)
Promotion(Bob) => HighIncome(Bob)
Promotion(Bob) => !HighIncome(Anna)
Variabilized tree
Promotion(X) <=> !Promotion(Y)
Promotion(Y) => HighIncome(Y)
Promotion(Y) => !HighIncome(X)

Figure 7.2: Illustration of tree shattering for a factor graph **(top row)** and for the corresponding Markov logic network from Tab. 7.1 for a domain with two persons **(bottom row)**: from the original model **(left)** we compute a tree piece **(center)**. Starting from factor f_3 (the root clause), we randomly follow the tree-structured "unrolling" of the graphical model rooted at f_3. Green shows that the factor has been included in the random walk while all red factors have been discarded. This results in the tree pattern for f_3 shown on the right hand side. A similar random walk generated the other shown tree pattern for f_1 **(top right)**. The resulting tree is variablized **(bottom right)** to be applied to other instantiations of the MLN clause.

The two ground clauses share the ground atom Promotion(Bob). *We uniformly sample $p \in [0,1]$. It was small enough, say $p = 0.3 < (0.9)^1$, so f_4 is added to the tree. For the ground clause $f_2 =$*

$$\texttt{Promotion(Anna)} \Rightarrow \texttt{HighIncome(Anna)}$$

we sample $p = 0.85 > (0.9)^2$ so f_2 and all of its branches are discarded. Continuing, for the next ground clause $f_1 =$

$$\texttt{Promotion(Anna)} \Rightarrow \texttt{HighIncome(Bob)}$$

we sample $p = 0.5 < (0.9)^2$ so f_1 could be added. If we added f_1, however, it would together with f_3 and f_4 form a cycle, so its branch is discarded. For $f_5 =$

$$\texttt{Promotion(Bob)} \Rightarrow \texttt{HighIncome(Anna)}$$

we sample $p = 0.4 < (0.9)^2$ so it is added to the tree. Note that now we cannot add any more edges without including cycles. The set of clauses is then variablized, as shown in Fig. 7.2(right), to obtain the tree pattern P_k that can be applied to all groundings of the clause the root originated from.

$$\texttt{Promotion(Anna)} \Leftrightarrow \texttt{!Promotion(Bob)}$$

was the root clause of the tree pattern. Thus if we encounter another ground instance of the same clause, in this case

$$\texttt{Promotion(Bob)} \Leftrightarrow !\texttt{Promotion(Anna)}\ ,$$

we find the substitution to apply this pattern, e.g.,

$$P_k\{X \mapsto Bob, Y \mapsto Anna\}\ .$$

If the substitution is not unique, for example, when we have more than two co-workers seeking for a promotion, we choose a variable at random.

In case a pattern is not applicable we apply it to the root clause, which is always possible, and as far down the tree as we can, sacrificing lifting potential for that particular piece. The connectivity of a piece and thereby its size can be controlled via the discount t. In this way, we include longer range dependencies in our pieces without sacrificing efficiency. And more importantly, by forming tree patterns and applying them to all factors we ensure that we have a potentially high amount of lifting: *Since we have decomposed the model into smaller pieces, the influence of the evidence is limited to a shorter range and hence allows lifting the local models.*

Maximum-likelihood learning can be phrased in terms of maximizing the log-partition function $A(\theta)$ of a graphical model, and we can actually use a network decomposed into trees to minimize an upper bound on $A(\theta)$, i.e., we introduce a piecewise approximation of maximum likelihood training of relational models.

This follows from the convexity of $A(\theta)$. To see why this is the case, we first write the original parameter vector θ as a mixture of parameter vectors θ_{T_t} induced by the non-overlapping tractable subgraphs.[1] For each edge in our mega-example E, we add a tree T_t which contains all the original vertices but only the edges present in t. With each tree T_t we associate an exponential parameter vector θ_{T_t}. Let μ be a strictly positive probability distribution over the non-overlapping tractable subgraphs of each clause, such that the original parameter vector θ can be written as a combination of per-tree clause parameter vectors

$$\theta = \sum_F \sum_{t \in F} \mu_{t,F} \theta_{T_t}\ ,$$

where we have expressed parameter sharing among the ground instance of the clauses. Now using Jensen's inequality, we can state the following upper bound to the log partition function:

$$A(\theta) = A(\sum_F \sum_{t \in F} \mu_{t,F} \theta_{T_t}) = A(\sum_t \mu_t \theta_{T_t}) \leq \sum_t \mu_t A(\theta_{T_t}) \tag{7.1}$$

with $\mu_t = \sum_F \mu_{t,F}$. Since the $\mu_{t,F}$ are convex, the μ_t are convex, too, and applying Jensen's inequality is safe. So we can follow Sutton and McCallumn's [149] arguments.

[1]The subgraphs produced by Alg. 9 may overlap. We will show how to account for this in Sec. 7.2

Namely, for tractable subgraphs and a tractable number of models the right-hand side of Eq. (7.1) can be computed efficiently. Generally, it forms an optimization problem, which according to [160] can be interpreted as free energy and depends on a set of marginals and edge appearance probabilities, in our case the probability that an edge appears in a tree, i.e., is visited in the random walk. Also, it is easy to show that standard pieces, i.e., one factor per piece, are an upper bound to this bound since, we can apply Jensen's inequality again when breaking the trees into independent paths from the root to the leaves.

Now, we show how to turn this upper bound into a lifted online training for relational models.

7.2 Lifted Online Training via Stochastic Meta-Descent

Stochastic gradient descent algorithms update the weight vector in an online setting. We essentially assume that the pieces are given one at a time. The algorithms examine the current piece and then update the parameter vector accordingly. They often scale sub-linearly with the amount of training data, making them very attractive for large training data as targeted by statistical relational learning. To reduce variance, we may form *mini-batches* consisting of several pieces on which we learn the parameters locally. In contrast to the propositional case, however, mini-batches have another important advantage: we can now make use of the symmetries within and *across* pieces for lifting.

The bound of Eq.(7.1) holds for tree pieces that partition the graph into disjoint sets of factors. The relational tree shattering finds a pattern for each clause and applies it to all the ground instances. To cope for the multiplicity introduced we normalize the gradient by the number of appearances of a clause. More formally, the gradient in (2.13) is approximated by

$$\sum\nolimits_i \#_i^{-1} \cdot \frac{\partial \ell(\theta, D_i)}{\partial \theta_k}, \tag{7.2}$$

where the mega-example D is broken up into pieces respectively mini-batches of pieces D_i. Here the vector $\#_i$ denotes a per-clause normalization that counts how often each clause appears in mini-batch D_i and $\cdot$ is the component-wise multiplication. This is a major difference to the propositional case and avoids "double counting" parameters which otherwise would be the case when training from the tree pieces due to potential multiplicity of factors. Recall that we build a piece from every grounding of each clause. During the random walk ground clauses can be visited repeatedly such that they appear in multiple different pieces. The normalization by $\#_i$ accounts for this fact. Let g_i be a gradient over the the mini-batch D_i. For a single piece we count how often a ground instance of each clause appears in the piece D_i. If D_i consists of more than one piece we add the count vector of all pieces together. For example, if for a model with 4 clauses the single piece mini-batch D_i has counts $(1, 3, 0, 2)$ the gradient is normalized by the respective counts. If the mini-batch, however, has an additional piece with counts $(0, 2, 1, 0)$ we normalize by the sum, i.e. by $(1, 5, 1, 2)$.

Since the gradient involves inference per batch only, inference is again feasible and more importantly liftable. Consequently, we can scale to problem instances traditional

Algorithm 10: Lifted Online Training of Relational Models

```
   Input: Markov Logic Network $M$, mega-example $E$, decay factors $t$, $\gamma$, and $\lambda$
   Output: Parameter vector $\theta$
   // Generate mini-batches
1  Generate set of tree pieces $\mathcal{T}$ using RelTreeFinding;
2  Randomly form mini-batches $\mathcal{B} = \{B_1, \dots, B_m\}$ each consisting of $l$ pieces;
   // Perform lifted stochastic meta-descent
3  Initialize $\theta$ and $v_0$ with zeros and the covariance matrix $C$ to the zero matrix;
4  while not converged do
5      Shuffle mini-batches $\mathcal{B}$ randomly;
6      for $i = 1, 2, \dots, m$ do
7          Compute gradient $g$ for $B_i$ using lifted loopy belief propagation;
8          Update covariance matrix $\mathbf{C}$ using (7.3) or some low-rank variant;
9          Update parameter vector $\theta$ using (7.4) and the involved equations;
10 return $\theta$;
```

relational methods can not easily handle. However, the asymptotic convergence of first-order stochastic gradients to the optimum can often be painfully slow if , e.g., the step-size is too small. One is tempted to just employ standard advanced gradient techniques such as L-BFGS. As the gradient is stochastically approximated by random subsamples, the measurements are inherently noisy. This confuses the line searches of conjugate gradient and quasi-Newton methods as conjugacy of the search direction over multiple iterations can not be maintained [141]. Gain adaptation methods like Stochastic Meta-Descent (SMD) overcome these limitations by using second-order information to adapt a per-parameter step size [159]. However, while SMD is very efficient in Euclidean spaces, Amari [6] showed that the parameter space is actually a Riemannian space of the metric C, the covariance of the gradients. Consequently, the ordinary gradient does not give the steepest direction of the target function which is instead given by the natural gradient, that is, by $C^{-1}\boldsymbol{g}$. Intuitively, the natural gradient is more conservative and does not allow large variances. If the gradients highly disagree in one direction, one should not take the step. Thus, whenever we have computed a new gradient $\boldsymbol{g}_t$ we integrate its information and update the covariance at time step t by the following expression:

$$C_t = \gamma C_{t-1} + \boldsymbol{g}_t \boldsymbol{g}_t^T \tag{7.3}$$

where $C_0 = \mathbf{0}$, and γ is a parameter that controls how much older gradients are discounted. Now, let each parameter θ_k have its own step size η_k. We update the parameter by

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \boldsymbol{\eta}_t \cdot \boldsymbol{g}_t \ , \tag{7.4}$$

where $\cdot$ denotes component-wise multiplication. The gain vector $\boldsymbol{\eta}_t$ thus serves as a diagonal conditioner. The vector $\boldsymbol{\eta}$ containing the individual per-parameter step sizes is adapted with the meta-gain μ:

$$\boldsymbol{\eta}_{t+1} = \boldsymbol{\eta}_t \cdot \exp(-\mu \boldsymbol{g}_{t+1} \cdot \boldsymbol{v}_{t+1}) \approx \boldsymbol{\eta}_t \cdot \max(\frac{1}{2}, 1 - \mu \boldsymbol{g}_{t+1} \cdot \boldsymbol{v}_{t+1}) \tag{7.5}$$

where $\boldsymbol{v} \in \Theta$ characterizes the long-term dependence of the system parameters on gain history. The time scale is determined by the decay factor $0 \leq \lambda \leq 1$. The vector $\boldsymbol{v}$ is iteratively updated by

$$\boldsymbol{v}_{t+1} = \lambda \boldsymbol{v}_t - \boldsymbol{\eta} \cdot (\boldsymbol{g}_t + \lambda C^{-1} \boldsymbol{v}_t) \ . \tag{7.6}$$

To ensure a low computational complexity and a good stability of the computations, one can maintain a low rank approximation of C, see [93] for more details. Using per-parameter step-sizes considerably accelerates the convergence of stochastic natural gradient descent.

Putting everything together, we arrive at the lifted online learning for relational models as summarized in Alg. 10. We form mini-batches of tree pieces **(lines 1-2)**. After initialization **(lines 3)**, we then perform lifted stochastic meta-descent **(lines 4-9)**. That is, we randomly select a mini-batch, compute its gradient using lifted inference, and update the parameter vector. Note that pieces and mini-batches can also be computed on the fly and thus their construction be interweaved with the parameter update. We iterate these steps until convergence, e.g., by considering the change of the parameter vector in the last l steps. If the change is small enough, we consider it as evidence of convergence. To simplify things, we may also simply fix the number of times we cycle through all mini-batches. This also allows to compare different methods.

Now, we have everything together to investigate scalable lifted inference and training.

Evaluation

Our intention here is to investigate the following questions:

(Q7.1) Does piecewise lifted inference help in non-symmetric cases?

(Q7.2) Can we efficiently train relational models using stochastic gradients?

(Q7.3) Are there symmetries within mini-batches that result in lifting?

(Q7.4) Can relational treefinding produce pieces that balance accuracy and lifting well?

(Q7.5) Is it even possible to achieve one-pass relational training?

To this aim, we implemented lifted online learning for relational models in Python and C++. As a batch learning reference, we used *scaled conjugate gradient (SCG)* [107]. SCG chooses the search direction and the step size by using information from the second order approximation. Inference as a subroutine for the training methods was also carried out by LLBP.

For the evaluation of the training approaches, we computed the *conditional marginal log-likelihood* (CMLL) [94]. Instead of the global log-partition function CMLL is defined in terms of the marginal probabilities and measures the ability to predict

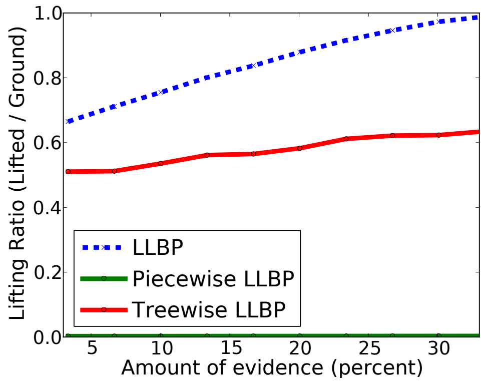


Figure 7.3: Lifting Ratio for CORA entity resolution MLN with varying amount of evidence. Although there is some lifting initially for LLBP, with a certain amount of evidence lifted loopy belief propagation is basically ground. Piecewise lifted inference, on the other hand, achieves networks that are orders of magnitude smaller than standard lifting. The trees keep some dependencies compared to the standard pieces and still manage to achieve compression for LLBP.

each variable separately. More precisely, we first divide the variables into two groups: X_{hidden} and $X_{observed}$. Then, we compute

$$\text{CMLL} = \sum\nolimits_{X \in X_{hidden}} \log P(X|X_{observed}) \tag{7.7}$$

for the given mega-example. To stabilize the metric, we divided the variables into four groups and calculated the average CMLL when observing only one group and hiding the rest. All experiments were conducted on a single machine with 2.4 GHz and 64 GB of RAM.

Lifted Piecewise Inference

The experiments in the previous sections have shown that we can considerably speed up inference by lifting. However, in cases where there are no symmetries or symmetries are broken by asymmetric evidence we do not gain and lifted loopy belief propagation basically falls back to the propositional case. This is the case when naïvely applying lifted inference to the training of relational models.

As we have argued above, the influence is limited when we break the model into pieces and we can gain significant lifting of the model by running inference locally. Fig. 7.3 shows the results for the Cora entity resolution MLN, which we will also train later. We sampled 10 bibliographies and extracted all facts corresponding to these bibliography entries. Then we varied the amount of evidence on the facts. One can see that lifted LBP achieves some compression at the beginning. However, with a certain amount of evidence lifted loopy belief propagation is basically ground and achieves very little compression. Piecewise lifted inference, on the other hand, achieves networks that are orders of magnitude smaller compared to standard lifting. However, by shattering the model into pieces all dependencies are broken. Tree-pieces balance this trade-off.

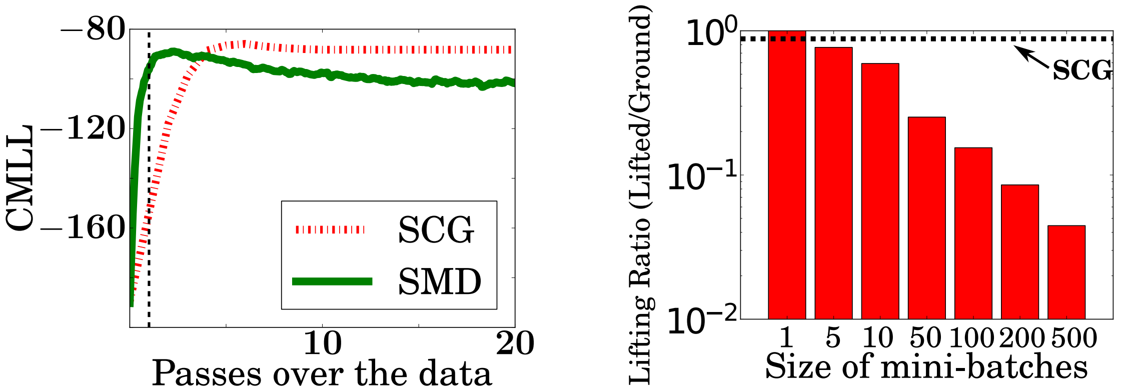


Figure 7.4: "Passes over mega-example" vs. Test-CMLL for the Friends-and-Smokers **(left)** (the higher the better). lifted online learning has already learned before seeing the mega example even once (black vertical line). **(right)** Benefit of local training for lifting. Lifting ratio for varying mini-batch size versus the full batch model on the Friends-and-Smokers MLN. Clearly for a batch size of 1 there is no lifting but with larger mini-batch sizes there is more potential to lift the pieces within each batch; the size can be an order of magnitude smaller. (Best viewed in color)

The trees keep some dependencies and still manage to achieve compression for LLBP. As we will see in the following training experiments these additional dependencies will help to train the model faster. Lifted Piecewise inference clearly enables lifting in non-symmetric cases where standard lifting fails **(Q7.1)**.

Training of Friends-and-Smokers MLN

In our first training experiment we learned the parameters for the "Friends-and-Smokers" MLN [146], which basically defines rules about the smoking behavior of people, how the friendship of two people influences whether a person smokes or not, and that a person is more likely to get cancer if he smokes. The "Friends-and-Smokers" MLN, however, is an oversimplification of the effects and rich interactions in social networks. Thus we enriched the network by adding two clauses: if someone is stressed he is more likely to smoke and people having cancer should get medical treatment. For a given set of parameters we sampled 5 datasets from the joint distribution of the MLN with 10 persons. For each dataset we learned the parameters on this dataset and evaluated on the other four. The ground network of this MLN contains 380 factors and 140 variables. The batch size was 10 and we used a stepsize of 0.2. Other parameters for SMD were chosen to be $\lambda = .99$, $\mu = 0.1$, and γ the discount for older gradients as 0.9. Fig. 7.4(left) shows the CMLL averaged over all of the 5 folds.

As one can see, the lifted SMD has a steep learning curve and has already learned the parameters before seeing the mega example even once (indicated by the black vertical line. Note that we learned the models without stopping criterion and for a fixed number of passes over the data thus the CMLL on the test data can decrease. SCG on the other hand requires four passes over the entire training data to have a similar

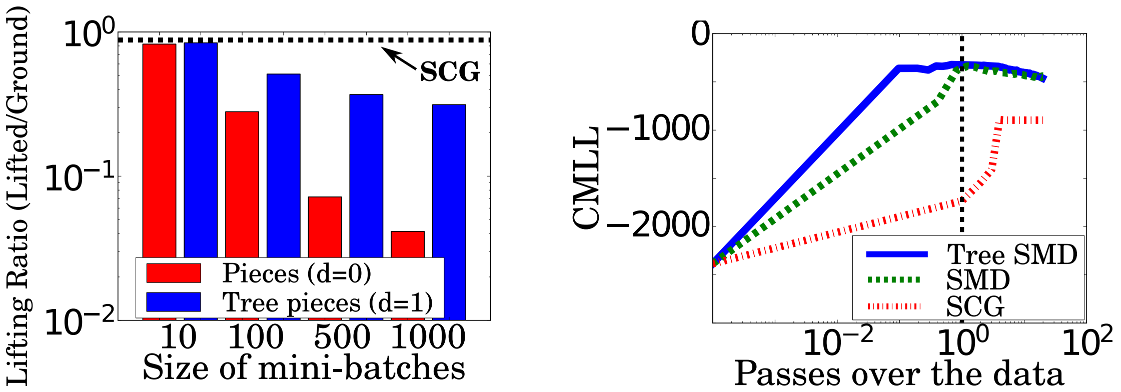


Figure 7.5: Experimental results. **(left)** Lifting ratio for standard pieces vs. tree pieces on the Voting MLN. Due to rejoining of pieces, additional symmetries are broken and the lifting potential is smaller. However, the sizes of the models per mini-batch still gradually decrease with larger mini-batch sizes. **(right)** "passes over mega-example" vs. Test-CMLL for the CORA MLN (the higher the better). (Best viewed in color)

result in terms of CMLL. Thus **(Q7.2)** can be answered affirmatively. Moreover, as Fig. 7.4(right) shows, piecewise learning greatly increases the lifting compared to batch learning, which essentially does not feature lifting at all. Thus, **(Q7.3)** can be answered affirmatively.

Voting MLN

To investigate whether tree pieces although more complex can still yield lifting, we considered the Voting MLN from the Alchemy repository. The network contains 3230 factors and 3230 variables. Note that it is a propositional Naive Bayes (NB) model. Hence, depth 0 pieces will yield greater lifting but hamper information flow among attributes if the class variable is unobserved. Tree pieces intuitively couple depth 0 hence will indeed yield lower lifting ratios. However, with larger mini-batches they should still yield higher lifting than the batch case. This is confirmed by the experimental results summarized in Fig. 7.5(right) that shows the lifting ratio for standard pieces vs. tree pieces with depth $d = 1$ with a threshold of $t = 0.9$. Thus, **(Q7.4)** can be answered affirmatively.

Training of CORA Entity Resolution MLN

Here we learned the parameters for the Cora entity resolution MLN, one of the standard datasets for relational learning. In this thesis, however, it is used in a non-standard, more challenging setting. For a set of bibliography entries (papers) the Cora MLN has facts, e.g., about word appearances in the titles and in author names, the venue a paper appeared in, its title, etc. The task is now to infer whether two entries in the bibliography denote the same paper (predicate *samePaper*), two venues are the same (*sameVenue*), two titles are the same (*sameTitle*), and whether two authors are the same (*sameAuthor*). We sampled 20 bibliography entries and extracted all facts corresponding to these bibliography entries. We constructed five folds then trained on four folds and tested on the fifth. Each fold can be seen as a training data set and thus the mega-example E is composed of the four folds we train on. We employed a transductive learning setting for this task. The MLN was parsed with all facts for the bibliography entries from the five folds, i.e., the queries were hidden for the test fold. The query consisted of all four predicates (*sameAuthor*, *samePaper*, *sameBib*, *sameVenue*). The resulting ground network consisted of $36,390$ factors and $11,181$ variables. We learned the parameters using SCG, lifted stochastic meta-descent with standard pieces as well as pieces using relational treefinding with a depth d of 1 and a threshold t of 0.9. The trees consisted of around ten factors on average. So we updated with a batch size of 100 for the trees and 1000 for standard pieces with a stepsize of 0.05. Furthermore, other parameters were chosen to be $\lambda = .99$, $\mu = 0.9$, and $\gamma = 0.9$. Fig. 7.5(right) shows the averaged learning results for this entity resolution task. Again, online training does not need to see the whole mega-example; it has learned long before finishing one pass over the entire data. Thus, **(Q7.4)** can be answered affirmatively.

Moreover, Fig. 7.5(right) also shows that by building tree pieces one can considerably speed-up the learning process. They convey a lot of additional information such that one obtains a better solution with a smaller amount of data. This is due to the fact that the Cora dataset contains a lot of strong dependencies which are all broken if we form one piece per factor. The trees on the other hand preserve parts of the local structure which significantly helps during learning. Thus, **(Q7.5)** can be answered affirmatively.

Lifted Imitation Learning in the Wumpus Domain

To further investigate **(Q7.4)** and **(Q7.5)**, we considered imitation learning in a relational domain for a Partially Observed Markov Decision Process (POMDP). We created a version of the Wumpus task (see e.g. [137]) where the location of Wumpus is partially observed. In the Wumpus world the agent is placed in a cave with connected rooms and there is a Wumpus that eats everyone it encounters. It can be shot by the agent but the agent only has one arrow. The Wumpus is surrounded by stench and that is how the agent might recognize its position. The agent performs different actions in every round such as move and shoot.

We used a 5×5 grid with a Wumpus placed in a random location in every training trajectory. The Wumpus is always surrounded by stench on all four sides. We do not have any pits or breezes in our task. The agent can perform eight possible actions: she can move or shoot in any direction (left, right, up, down). The agent's task is to move to a cell so that she can fire an arrow to kill the Wumpus. The Wumpus is not observed in all the trajectories although the stench is always observed. Trajectories were created by real human users who play the game.

The cells of the 5×5 grid were numbered and we use predicates such as

cellAtRow(cell,row) and *cellAbove(cell,cell)*

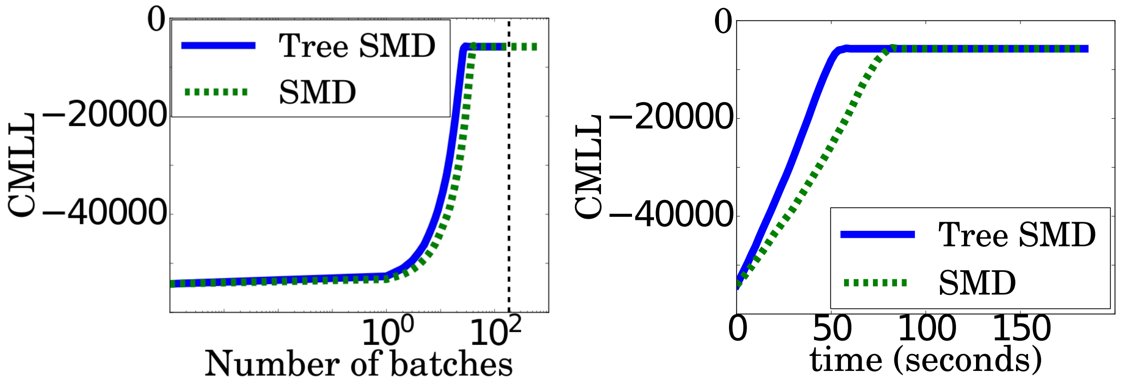


Figure 7.6: Experimental results for the Wumpus MLN. **(left)** ”number of batches” vs. Test-CMLL (the higher the better). **(right)** Runtime vs. CMLL. As one can see, lifted online learning has already converged before seeing the mega example even once (black vertical line). For the Wumpus MLN, SCG did not converge within 72 hours. (Best viewed in color)

to define the structure of the grid and were the cell is located. These facts were always given. Other predicates were

wumpus(cell), *stench(cell)*, *agent(t,cell)*

and actions move and shoot for all directions, e.g., *shootUp(t)*. The rules we learn the weights for describe the state or whether an action should be performed. Two examples of such rules are:

w_1 : *stench(scell)* $\wedge$ *cellAbove(scell,wcell)* $=>$ *wumpus(wcell)*

w_2 : *wumpus(wcell)* $\wedge$ *agent(acell, t)* $\wedge$ *cellCol(acell,acol)* $\wedge$ *cellCol(wcell,wcol)* $\wedge$ *less(acol,wcol)* $=>$ *shootRight(t)*

The resulting network contains 182400 factors and 4469 variables. We updated with a batch size of 200 for the trees (depth $d = 0$, threshold $t = 0.9$) and 2000 for standard pieces with a stepsize of 0.05. As for the Cora dataset used $\lambda = .99$, $\mu = 0.9$, and $\gamma = 0.9$. Figure 7.6 shows the result on this dataset for lifted SMD with standard pieces as well as pieces using relational treefinding with a threshold t of 0.9. For this task, SCG did not converge within 72 hours. Note that this particular network has a complex structure with lots of edges and large clauses. This makes inference on the global model intractable. Fig. 7.6 (left) shows the learning curve for the total number of batches seen as well as the runtime for one pass over the data (right). As one can see, tree pieces actually yield faster convergence, again long before having seen the dataset even once. Thus, **(Q7.4)** and **(Q7.5)** can be answered affirmatively.

Taking all experimental results together, all questions **(Q7.1)-(Q7.5)** can be clearly answered affirmatively.

8 Distributed Lifted Training

So far, we have shown how the model can be shattered into smaller pieces to efficiently learn the parameters. This shattering makes training large models tractable and improves on the speed of convergence, as we have shown in the previous chapter. Even more importantly, as each lifted piece is processed one after the other it naturally paves the way for a MapReduce approach. The gradients of the shattered pieces can be computed locally in a distributed fashion which in turn allows a *MapReduce* friendly parallel approach without bandwidth constrains and considerable latency, see e.g. [92, 172]. Lifted inference as a subroutine, however, is still carried out on a single core.

After briefly covering distributed inference approaches in Sec. 8.1 We will show how to lift the model in a distributed fashion (Sec. 8.2), the missing piece to prove the following theorem:

Theorem 8.0.1. *Lifted approximate training of relational models is MapReduce-able.*

8.1 Distributed Inference

With the ubiquitous sensor technology, data services, and mobility of people, among other sources, the amount of available data and the need to make sense of it and find interesting information is rapidly increasing. Therefore we need intelligent and efficient machine learning and AI algorithms. The scale of these algorithms, however, will be limited due to physical limitations that exist for the computer hardware. The sheer computing power can not keep pace with the overwhelming exponential growth of the amount of data. With the availability of affordable commodity hardware and high performance networking, however, we have increasing access to computer clusters providing an additional dimension to scale all our algorithms, in particular lifted inference. This is appealing since in addition to other dimensions of scaling, such as symmetries and approximations, among others, parallel machine learning and AI algorithms can benefit from the number of processors and get an additional speed-up in the number of cores. The parallelism can be realized at different levels and with various architectures. Central processing units (CPUs) of modern computers already have multiple cores that can be utilized in parallel, graphics processing units (GPUs) of modern graphics card offer hundreds and thousands of parallel cores, and clusters of many interconnected computers and clouds offer new opportunities for computing. Loopy belief propagation is naturally parallelizable on a graph level as all computations are only local.

As already mentioned in Sec. 1.3 there exist a lot of approaches to scaling probabilistic inference and training, for different settings and technologies. Some of which have grown into powerful frameworks.

Two notable examples are *Felix*[1] and *GraphLab*[2]. *Felix* is a relational optimizer for statistical inference using the *Tuffy* MLN inference engine written in Java and using an SQL Database. Making use of relational database technologies, for example, for more efficient grounding, this can speed up relational inference. *GraphLab* is an open-source software platform for machine learning on graphs that also includes parallel inference using loopy belief propagation. Although Gonzalez *et al.* [58, 59] report on map-reducing lifted inference within MLNs, they actually assume the lifted network to be given.

Unfortunately, the lifting is not directly able to efficiently utilize cluster resources as they are specialized to particular models or settings. In the following section we will show that this is indeed possible, that is, we can distribute the color-passing procedure for lifting message-passing using the MapReduce framework [43]. We hereby demonstrate for the first time that lifting per se is MapReduce-able, thus putting scaling lifted SRL to “Big Data” within reach.

As our experimental results will illustrate, by combining the two orthogonal scaling approaches we can achieve orders of magnitude improvement over existing methods.

[1]http://hazy.cs.wisc.edu/hazy/felix/

[2]http://graphlab.org

Algorithm 11: MR-CP: MapReduce Color-passing

```
1  repeat
       // Color-passing for the factor nodes
2      forall the $f \in G$ do in parallel
3          $s(f) = (s(X_1), s(X_2), \ldots, s(X_{d_f}))$ , where $X_i \in nb(f), i = 1, \ldots, d_f$
4      Sort all of the signatures $s(f)$ in parallel using MapReduce;
5      Map each signature $s(f)$ to a new color, s.t. $col(s(f_i)) = col(s(f_j))$ iff
       $s(f_i) = s(f_j)$
       // Color-passing for the variable nodes
6      forall the $X \in G$ do in parallel
7          $s(X) = (s(f_1), s(f_2), \ldots, s(f_{d_X}))$ , where $f_i \in nb(X), i = 1, \ldots, d_X$
8      Sort all of the signatures $s(X)$ in parallel using MapReduce;
9      Map each signature $s(X)$ to a new color, s.t. $col(s(X_i)) = col(s(X_j))$ iff
       $s(X_i) = s(X_j)$
10 until grouping does not change;
```

8.2 MapReduce Lifting

If one takes a closer look at color-passing one notices that each iteration of color-passing basically consists of three steps, namely

1. Form the colorsignatures
2. Group similar colorsignatures
3. Assign a new color to each group

for the variables and the factors, respectively. We now show how each step can be carried out within the MapReduce framework.

The resulting MapReduce color-passing is summarized in Alg. 11 and has to be repeated in each iteration for the factors and the nodes respectively. Specifically, recall that color-passing is an iterative procedure so that we only need to take care of the direct neighbors in every iteration, i.e., building the colorsignatures for the variables (factors) requires the variables' (factors') own color and the color of the neighboring factors (variables). *Step-1* can thus be distributed by splitting the network into k parts and forming the colorsignatures within each part independently in parallel. However, care has to be taken at the borders of the splits. Fig. 8.1 shows the factor graph from Fig. 2.2 and how it can be split into parts for forming the colorsignatures for the variables (Fig. 8.1 **(left)**) and the factors (Fig. 8.1 **(right)**). We see that to be able to correctly pass the colors in this step we have to introduce copynodes at the borders of the parts.[1] In the case of the node signatures, for example (Fig. 8.1 **(left)**), both f_1 and f_2 have to be duplicated to be present in all signatures of the variables. The structure of the network does not change

[1] Note that in the shared memory setting this is not necessary. Here we use the MapReduce framework, thus we have to introduce copynodes.

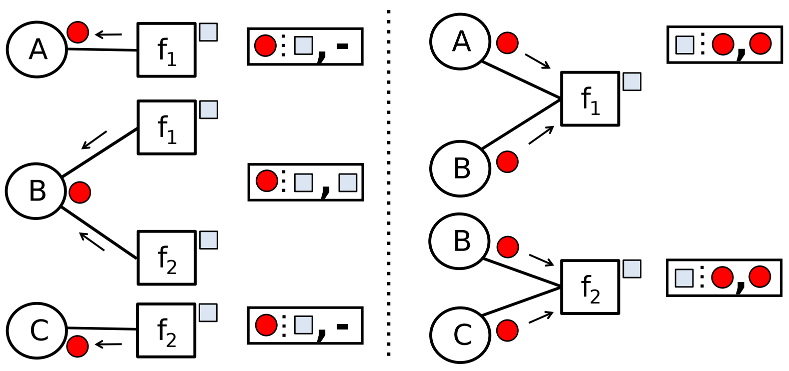


Figure 8.1: The partitions when passing colors to build the signatures for the variables **(left)** and factors **(right)** and the corresponding colorsignatures that have to be sorted. Copynodes (-factors) have to be introduced at the borders to form the correct neighborhood.

during color-passing, only the colors may change in two subsequent iterations and have to be communicated. To reduce communication cost we thus split the network into parts before we start the CP-procedure and use this partitioning in all iterations. [1]

The next step that has to be carried out is the sorting of the signatures of the variables (factors) within the network (*Step-2*). These signatures consist of the node's own color and the colors of the respective neighbors. Fig. 8.1 also shows the resulting colorsignatures after color-passing in the partitions of the example from Fig. 2.2. For each node (Fig. 8.1 **(left)**) and factor (Fig. 8.1 **(right)**) we obtain the colorsignatures given the current coloring of the network. These colorsignatures have to be compared and grouped to find nodes (factors) that have the same one-step communication pattern. Finding similar signatures by sorting them can be efficiently carried out by using *radix sort* [34] which has been shown to be MapReduce-able [171]. Radix sort is linear in the number of elements to sort if the length of the keys is fixed and known. It basically is a non-comparative sorting algorithm that sorts data with grouping keys by the individual digits which share the same significant position and value. The signatures of our nodes and factors can be seen as the keys we need to sort and each color in the signatures can be seen as a digit in our sorting algorithm. Radix sort's efficiency is $O(kn)$ for n keys which have k or fewer digits. The close connection of color-passing to radix sort paves the way for an efficient MapReduce implementation of the lifting procedure.

Indeed, although radix sort is very efficient — the sorting efficiency is in the order of edges in the ground network — one may wonder whether it is actually well suited for an efficient implementation of lifting within a concrete MapReduce framework such as Hadoop. The Hadoop framework performs a sorting between the *Map-* and the *Reduce-*

[1]The way we partition the model greatly affects the efficiency of the lifting. Finding an optimal partitioning that balances communication cost and CPU-load, however, is out of the scope of this work. In general, the partitioning problem for parallelism is well-studied [26], there are efficient tools, e.g., `http://glaros.dtc.umn.edu/gkhome/metis/parmetis/`. We show that CP is MapReduce-able and dramatically improves scalability even with a naïve partitioning.

Algorithm 12: Map Function for Hadoop Sorting

```
  Input: Split $S$ of the network
  Output: Signatures of nodes $X_i \in S$ as key value pairs $(s(X_i), X_i)$
1 forall the $X_i \in S$ do in parallel
2     $s(X_i) = (s(f_1), s(f_2), \ldots, s(f_{d_X}))$ , where $f_j \in nb(X_i), j = 1, \ldots, d_{X_i}$
3 return key-value pairs $(s(X_i), X_i)$
```

Algorithm 13: Reduce Function for Hadoop Sorting

```
  Input: Key-value pairs $(s(X_i), X_i)$
  Output: Signatures with corresponding list of nodes
1 Form list $L$ of nodes $X_i$ having same signature
2 return key-value pair $(s(L), L)$
```

step. The key-value pair that is returned by the mappers has to be grouped and is then sent to the respective reducers to be processed further. This sorting is realized within Hadoop using an instance of quick sort with an efficiency $O(n \log n)$. If we have a bounded degree of the nodes in the graph as in our case, however, this limits the length of the signatures and radix sort is still the algorithm of choice.

Moreover, our main goal is to illustrate that — in contrast to many other lifting approaches — color-passing is naturally MapReduce-able. Consequently, for the sake of simplicity, we stick to the Hadoop internal sorting to group our signatures and leave a highly efficient MapReduce realization for future work. Alg. 12 and Alg. 13 show the map and the reduce function respectively. In the map phase we form the signatures of all nodes (factors) in a split of the graph and reduce a key-value pair of $(s(X_i), X_i)$ which are the signature and the id of the node (factor). These are then grouped by the reduce function and a pair $(s(L), L)$ for all nodes having the same signature, i.e., $s(L) = s(X_i)$ for all $X_i \in L$. In practice one could gain additional speed-up the if the sorting of Hadoop is avoided, for example, by realizing a *Map*-only task using a distributed ordered database like *HBase* [1]. The signatures that are formed in *Step-1* would be written directly into the database as $(s_(X_i)\_X_i, X_i)$, i.e., for each node an entry with a key that is composed of the signature and the node's id is inserted and the groupings could be read out sequentially for the next step.

The *Map*-phase of (*Step-2*) could also be integrated into the parallel build of the signatures (*Step-1*), such that we have one single *Map*-step for building and sorting the signatures.

Finally, reassigning a new color (*Step-3*) is an additional linear time operation that does one pass over the nodes (factors) and assigns a new color to each of the different groups. That is, we first have to build the color signatures. The splits of the network are the input and are distributed to the mappers. In the example shown in Fig. 8.2 there is one line for every node and its local neighborhood, i.e., ids of the factors it is connected to. The mappers take this input and form the signatures of the current

[1]http://hbase.apache.org/

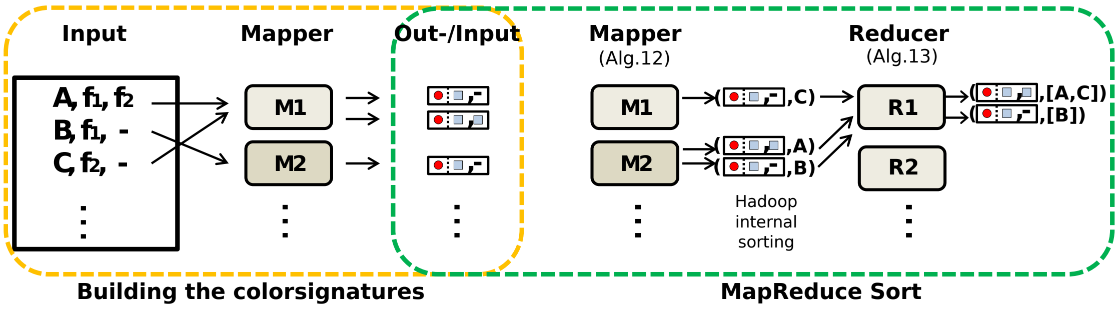


Figure 8.2: MapReduce jobs for *Step-1* **(yellow)** and *Step-2* **(green)** of color-passing. For building the signatures the local parts are distributed to the mappers that form the signature based on the colorings of the current iteration. The output is sorted and for each signature a list of nodes with the same signature is returned.

iteration independently in parallel. These signatures are then passed on to MapReduce sort. Fig. 8.2 shows the sorting procedure for the signatures. The mappers in the sorting step, output a tuple of key-value pairs consisting of the the signature and the *id* of the respective node (Alg. 12).[1] These are then grouped by the reduce operation, such that all nodes having the same signatures are returned in a list (Alg. 13). The internal sorting of the Hadoop framework takes place between the map- and the reduce-phase, such that the key-value pairs can be sent to the right reducers.

Taking the MapReduce arguments for (*Step-1*) to (*Step-3*) together this proves that color-passing itself is MapReduce-able. Together with the MapReduce-able results of Gonzalez *et al.* [58, 59] for the modified loopy belief propagation, this proves the following Theorem.

Theorem 8.2.1. *Lifted loopy belief propagation is MapReduce-able.*

Moreover, we have the following time complexity, which essentially results from running radix sort h times.

Theorem 8.2.2. *The runtime complexity of the color-passing algorithm with h iterations is $O(hm)$, where m is the number of edges in the graph.*

Proof. Assume that every graph has n nodes (ground atoms) and m edges (ground atom appearances in ground clauses). Defining the signatures in (*Step-1*) for all nodes is an $O(m)$ operation. The elements of a signature of a factor are of $s(f) = (s(X_1), s(X_2), \ldots, s(X_{d_f}))$, where $X_i \in nb(f)$, $i = 1, \ldots, d_f$. Now there are two sorts that have to be carried out. The first sort is within the signatures. We have to sort the colors within a node's signatures and in the case where the position in the factor does not matter, we can safely sort the colors within the factor signatures while compressing the factor graph. Sorting the signatures is an $O(m)$ operation for all nodes. This efficiency can be achieved by using counting sort, which is an instance of bucket sort, due to

[1]Since this is a essentially the output of the previous step that is passed through the two steps can easily be integrated. We keep them separate for illustration purposes of the two distinct steps of color-passing.

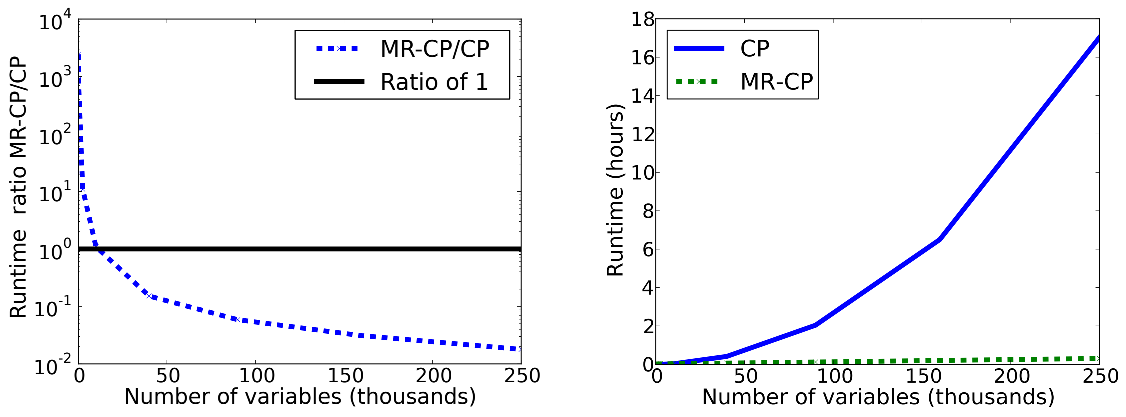


Figure 8.3: **(left)** The ratio of the runtime for single-core color-passing and MapReduce color-passing (MP-CP) on grids of varying size. The ratio is 1 at the black horizontal line which indicates that at this point both methods are equal. We see that due to the overhead of MapReduce for smaller networks the single-core method is faster, MR-CP is advantageous for larger networks. **(right)** Runtimes for color-passing and MR-CP. One can clearly see that MR-CP scales lifting to much larger instances.

the limited range of the elements of the signatures. The cardinality of this signature is upper-bounded by n, which means that we can sort all signatures in $O(m)$ by the following procedure. We assign the elements of all signatures to their corresponding buckets, recording which signature they came from. By reading through all buckets in ascending order, we can then extract the sorted signatures for all nodes in a graph. The runtime is $O(m)$ as there are $O(m)$ elements in the signatures of a graph in iteration i. The second sorting is that of the resulting signatures to group similar nodes and factors. This sorting is of time complexity $O(m)$ via radix sort. The label compression requires one pass over all signatures and their positions, that is, $O(m)$. Hence all these steps result in a total runtime of $O(hm)$ for h iterations. □

Evaluation

We have just shown for the first time that lifting per se can be carried out in parallel. Thus, we can now combine the gains we get from the two orthogonal approaches to scaling inference, putting scaling lifted SRL to Big Data within reach. That is, we investigate the question:

(Q8.1) Does the MapReduce lifting additionally improve scalability.

We compare the color-passing procedure with a single-core Python/C++ implementation and a parallel implementation using MRJob[1] and Hadoop [2]. We partitioned the network per node (factor) and introduced copynodes (copyfactors) at the borders. The grouping of the signatures was found by a MapReduce implementation of Alg. 12 and

[1]`https://github.com/Yelp/mrjob`
[2]`http://hadoop.apache.org/`

Alg. 13. Note that for this experiment the amount of lifting is not important. Here, we compare how the lifting methods scale. Whether we achieve lifting or not does not change the runtime of the lifting computations. As Theorem 8.2.2 shows, the runtime of color-passing depends on the number of edges present in the graph. Thus, we show the time needed for lifting on synthetic grids of varying size to be able to precisely control their size and the number of edges in the network.

Fig. 8.3 shows the results on grids of varying size. We scaled the grids from 5×5 to 500×500, resulting in networks with 25 to 250.000 variables. The ratio is 1 at the black horizontal line which indicates that at this point both methods are equal. Fig. 8.3 **(left)** shows the ratio of the runtime for single-core color-passing and MapReduce color-passing (MR-CP). We see that due to the overhead of MapReduce the single-core method is faster for smaller networks. However, as the number of variables grows, this changes rapidly and MR-CP scales to much larger instances. Fig. 8.3 **(right)** shows the running time of single-core color-passing and MapReduce color-passing. One can see that MR-CPs scales orders of magnitude better than single-core color-passing. The results clearly favor MR-CP for larger networks and affirmatively answer **(Q8.1)**

Conclusion

Symmetries can be found almost everywhere, in arabesques and French gardens, in the rose windows and vaults in Gothic cathedrals, in the meter, rhythm, and melody of music, in the metrical and rhyme schemes of poetry as well as in the patterns of steps when dancing. Symmetric faces are even said to be more beautiful to humans. Actually, symmetry is both a conceptual and a perceptual notion often associated with beauty-related judgments [168]. Or, to quote Hermann Weyl ”Beauty is bound up with symmetry” [164]. This link between symmetry and beauty is often made by scientists. In physics, for instance, symmetry is linked to beauty in that symmetry describes the invariants of nature, which, if discerned, could reveal the fundamental, true physical reality [168]. In mathematics, as Herr and Bödi note, ”we expect objects with many symmetries to be uniform and regular, thus not too complicated” [64].

Summary

In the **first part** we have shown how to use symmetries within graphical models to speed up inference using loopy belief propagation (LBP) by focusing on a number of different queries or inference task. We have introduced *lifted LBP*, that exploits symmetries by grouping together random variables and factors that send and receive the same messages. By compressing the graph we can compute marginal probabilities very efficiently using slightly modified LBP equations, and we have shown that the very same process is applicable to both discrete and continuous models, as we have shown for GaBP.

In some applications, however, the lifting criterion is too strict as it does not make use of the fact that messages decay along paths. *Informed lifted LBP* is generally applicable and interleaves lifting and modified LBP iterations, thus can group together nodes, respectively factors, if their actual LBP messages are identical. Intuitively, it adaptively explores the subgraph around each node, respectively factor, and groups them on an as-needed basis making use of decaying error messages.

To be able to efficiently answer MAP queries, we presented the first application of lifted inference techniques to linear programming. The resulting *lifted linear programming* approach compiles a given LP into an equivalent but potentially much smaller LP by grouping variables, respectively constraints, that are indistinguishable given the objective function and apply a standard LP solver to this lifted LP.

Indeed, the link established here is related to symmetry-aware approaches in (mixed–) integer programming [97]. However, they are vastly different to LPs in nature. Symmetries in ILP are used for pruning the symmetric branches of search trees, thus the dominant paradigm is to add symmetry breaking inequalities, similarly to what has been done for SAT and CSP [142]. In contrast, lifted linear programming achieves speed-up by reducing the problem size. Furthermore, state-of-the-art symmetry detection for ILPs computes the so-called orbit partition of the graph whose colored adjacency matrix is the skeleton equation. This is a "graph isomorphism"-complete problem, whereas our approach detects symmetries in time $O(l \cdot (m + n \log n)$ [19]. Moreover, the orbit partition of a graph is a refinement of the coarsest equitable partition, thus our approach results in more compression. Regarding LPs, the work by Boedi *et al.* is probably the closest in spirit [18]. They showed that the set of combinatorial symmetries of the polytope that respect the objective can be used for compression. However, no polynomial algorithm for finding those symmetries was presented; instead they fell back to orbit partition-based methods in their experiments.

We then presented *conditioned lifted LBP* for computing arbitrary joint marginals using lifted LBP. It relates conditioning to computing shortest-paths. By combining lifted LBP and variable conditioning, it can readily be applied to models of realistic domain size. These contributions significantly advance our understanding of lifted inference and are relevant for several important AI tasks such as finding the MAP assignment, sequential forward sampling, cutset conditioning algorithms, and structure learning. Furthermore, *multi-evidence lifting* for loopy belief propagation exploits symmetries within and across different evidence cases and is applicable to a wide range of problems as shown on two novel tasks for lifted inference, namely computing PageRank contributions and the Kalman filter.

In the **second part** we have shown that scaling relational training of graphical models can actually greatly benefit from symmetries. However, already in 1848, Louis Pasteur recognized "Life as manifested to us is a function of the asymmetry of the universe". This remark characterizes somehow one of the main challenges we are facing: Not only are almost all large graphs asymmetric [48], but *even if there are symmetries within a model, they easily break when it comes to inference and training since variables become correlated by virtue of depending asymmetrically on evidence*. This, however, does not mean that lifted training is hopeless. We have demonstrated that breaking long-term dependencies via piece-wise inference and training naturally breaks asymmetries and paves the way to lifted online respectively MapReduced relational training.

Lessons learned

I would like to share a snippet of a conversation I once overheard. Two people at an undisclosed workshop I attended who were not familiar with lifted inference:

Person 1: "What is this whole lifted inference about?"

Person 2: "I don't know really. I think it is something about not making the same computations multiple times."

Person 1: "Hmm, to me it sounds more like common sense."

What sounds dismissive at first, actually is a very strong argument for lifting and symmetry aware approaches in general. Why would we dismiss something that is "common sense"? Humans make use of all kinds of symmetries in everyday activities. Such symmetries may exist in the task at hand, the shape of objects, or their exchangeability. When a human wants to pick up a ball, it is symmetric in two aspects. The shape of the ball is symmetric and by experience we know that it does not make a difference where we grab the ball. All of its reachable spots are equally good and thus exchangeable. A less obvious symmetry lies in the task of picking up the ball. Since we have learned how to pick up ball shaped objects, the cognitive work is minimal and we can simply apply our learned knowledge to this particular case. Unfortunately, automated systems do not make use of these symmetries while performing their tasks. The majority of our algorithms is not able to find and exploit these unless the symmetries are explicitly encoded.

Symmetries are not the single ingredient that make the algorithms efficient. However, lifting is more than just a grain of salt. It should rather be understood as one ingredient of a culinary composition. A meal is generally perceived as delicious if it has the right mixture of different flavors, that is, sweet, bitter, umami, salty and sour.

In the case of efficient algorithms the ingredients can be, for example, symmetries, parallelization, and approximation among others. These can be combined such that we obtain more efficient algorithms for more and more complex tasks we have to solve, as we have also seen in this thesis.

In this sense, I am more than ever convinced of the need for symmetry aware approaches, not only for lifted inference and training for graphical models, but for a "lifted" machine learning in general. In recent years lifting has been put on the map and I hope this thesis and its resulting publications have helped to do so.

It is important, however, to widen the scope and to reach out to other communities that have also seen the need for such approaches, although it is not always called lifting. A common ground is a precise mathematical formulation of the symmetries. Automorphisms have a long history in mathematics and graph theory, and this could be the starting point if the research of this thesis was going to be conducted again. In the case of graphical models, for example, the symmetries intuitively correspond to automorphisms of subnetworks. What kind of symmetries can we exploit in the different tasks? What do the other communities look at? What is their common mathematical basis? What can we learn from the research about automorphisms in graph theory?

Future work

The symmetry-aware framework for inference and learning outlined in the present thesis puts many interesting research goals into reach.

Besides lifted versions for loopy belief propagation and Gaussian belief propagation that we have introduced, other message passing approaches such as warning propagation and survey propagation [61] have been shown to be liftable too. Other algorithms are, for example, non-parametric BP [148] and particle BP [69]. These approaches can handle non-Gaussian continuous distributions where in general there is no closed form representation. Combining ideas from particle filters and loopy belief propagation, the propagated messages consist of particles obtained by a Gibbs sampling procedure. Although LBP has empirically shown to be a good choice in many areas, it is not exact on loopy graphs and does not necessarily converge. Generalized BP [167] is a BP variant that can improve on the approximation and the convergence properties of BP, like other convergent BP variants such as tree-reweighted BP (TRW-BP) [161] which performs reparameterizations of the original parameter vector. It remains to be shown that these other message-passing algorithms can exploit symmetries as well.

Furthermore, the definition of an abstract lifted message passing framework that unifies all these lifted message-passing algorithms remains to be done and is very interesting future work.

Another promising avenue for future work is to establish a similar strong link between MAP inference and LPs at the lifted level as it is known for the ground level [54, 87, 88, 147, 166] and to employ the link within complex problems such as protein-design, stereo-vision and image segmentation.

In this thesis we have also seen that random variables easily become correlated within complex applications by virtue of sharing propagated evidence. One way to cope with this issue is iLLBP, as we have introduced in this thesis. One should further develop approximate lifted approaches that can still gain compression.

One should also tackle one-pass relational learning by investigating different ways of gain adaption and scheduling of pieces for updates. Since piecewise training is a simple form of dual decomposition, further exploration of dual decomposition methods is an attractive future direction. One should also investigate budget constraints on both the number of examples and the computation time per iteration.

Another interesting avenue for future work is to use sequences of increasingly finer approximations to control the trade-off between lifting and accuracy [81].

Exploring the close connection to symmetry breaking in ILPs, CSPs, and MDPs and how the ideas carry over to lifted LPs is a promising future direction. Finally, symmetries are not exclusively inherent in probabilistic inference as we have seen through out this thesis and one should start investigating symmetries in general machine learning approaches such as support-vector machines and Gaussian processes.

While there have been considerable advances, there are more than enough problems, in particular asymmetric ones, to go around to really establish symmetry-aware machine learning.

References

[1] U. Acar, A. Ihler, R. Mettu, and O. Sumer. Adaptive inference on general graphical models. In *Proceedings of the Twenty-Fourth Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-08)*, pages 1–8, Corvallis, Oregon, 2008. AUAI Press. 4, 83

[2] B. Ahmadi, K. Kersting, and F. Hadiji. Lifted belief propagation: Pairwise marginals and beyond. In *Proceedings of the 5th European Workshop on Probabilistic Graphical Models (PGM–10)*, pages 9–16, Helsinki, Finland, 2010.

[3] B. Ahmadi, K. Kersting, M. Mladenov, and S. Natarajan. Exploiting symmetries for scaling loopy belief propagation and relational training. *Machine Learning Journal*, 92(1):91–132, July 2013.

[4] B. Ahmadi, K. Kersting, and S. Natarajan. Lifted online training of relational models with stochastic gradient methods. In N. Cristianini P. Flach, T. De Bie, editor, *Proceedings of the European Conference Machine Learning and Knowledge Discovery in Databases (ECML-PKDD-12)*, pages 585–600, Bristol, UK, Sept. 24–28 2012. Springer.

[5] B. Ahmadi, K. Kersting, and S. Sanner. Multi-evidence lifted message passing with application to PageRank and the Kalman filter. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*, IJCAI-11, pages 1152–1158. AAAI Press, 2011.

[6] S. Amari. Natural gradient works efficiently in learning. *Neural Computation*, 10(2):251–276, February 1998. 120

[7] U. Apsel and R. Brafman. Extended lifted inference with joint formulas. In *Proceedings of the Twenty-Seventh Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-11)*, pages 11–18. AUAI Press, 2011. 9

[8] U. Apsel and R. Brafman. Lifted meu by weighted model counting. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence (AAAI-12)*. AAAI Press, July 2012. 10

[9] C. J. Augeri. *On graph isomorphism and the Pagerank algorithm*. PhD thesis, Air Force Institute of Technology, WPAFB, Ohio, USA, 2008. 72

[10] D. Barbará, H. Garcia-Molina, and D. Porter. The management of probabilistic data. *IEEE Transactions on Knowledge and Data Engineering*, 4(5):487–502, October 1992. 13

[11] K. Beedkar, L. Del Corro, and R. Gemulla. Fully parallel inference in Markov logic networks. In *15th GI-Symposium Database Systems for Business, Technology and Web (BTW 2013)*, Magdeburg, Germany, 2013. Bonner Köllen. 13

[12] J. Besag. Statistical Analysis of Non-Lattice Data. *Journal of the Royal Statistical Society. Series D (The Statistician)*, 24(3):179–195, 1975. 11

[13] D. Bickson. *Gaussian Belief Propagation: Theory and Aplication*. PhD thesis, Hebrew University of Jerusalem, November 2008. 58

[14] D. Bickson, D. Dolev, and Y. Weiss. Efficient large scale content distribution. Technical Report Leibniz Center TR-2006-07, School of Computer Science and Engineering, The Hebrew University, 2006. 50, 55

[15] D. Bickson, D. Malkhi, and L. Zhou. Peer to peer rating. In *7th IEEE International Conference on Peer-to-Peer Computing (P2P-07)*, pages 211–218, Galway, Ireland, Sept. 2007. 95, 103

[16] D. Bickson, Y. Tock, O. Shental, and D. Dolev. Polynomial linear programming with Gaussian belief propagation. In *Proceedings of the 46th Annual Allerton Conference on Communication, Control and Computing*, pages 895–901, Allerton House, Illinois, Sept. 2008. 69, 70

[17] B. Bidyuk and R. Dechter. Cutset sampling for bayesian networks. *Journal of Artificial Intelligence Research*, 28(1):1–48, January 2007. 90

[18] R. Bödi, K. Herr, and M. Joswig. Algorithms for highly symmetric linear and integer programs. *Mathematical Programming, Series A*, 137(1-2):65–90, 2013. 3, 11, 136

[19] P. Boldi, V. Lonati, M. Santini, and S. Vigna. Graph fibrations, graph isomorphism, and PageRank. *RAIRO , Informatique Théorique*, 40:227–253, 2006. 136

[20] C. Boutilier, R. Reiter, and B. Price. Symbolic dynamic programming for first-order MDPs. In *Proceedings of the 17th international Joint Conference on Artificial Intelligence (IJCAI-01)*, pages 690–697, Seattle, WA, USA, 2001. Morgan Kaufmann Publishers Inc. 11

[21] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, March 2004. 30

[22] X. Boyen and D. Koller. Tractable inference for complex stochastic processes. In *Proceedings of the Fourteenth Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-98)*, pages 33–42, San Francisco, CA, 1998. Morgan Kaufmann. 45

[23] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30:197–117, 2006. 95

[24] H. H. Bui, T. N. Huynh, and R. de Salvo Braz. Exact lifted inference with distinct soft evidence on every object. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence (AAAI-12)*, 2012. 10

[25] H. H. Bui, T. N. Huynh, and S. Riedel. Automorphism groups of graphical models and lifted variational inference. *CoRR*, 2012. 10

[26] B. L. Chamberlain. Graph partitioning algorithms for distributing workloads of parallel computations. Technical report, University of Washington, 1998. 130

[27] R. Cheng, D. V. Kalashnikov, and S. Prabhakar. Evaluating probabilistic queries over imprecise data. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, SIGMOD '03, pages 551–562, New York, NY, USA, 2003. ACM. 13

[28] D. M. Chickering and D. Heckerman. Targeted advertising with inventory management. In *Proceedings of the 2nd ACM Conference on Electronic Commerce (EC-00)*, pages 145–149, 2000. 66

[29] J. Choi and E. Amir. Lifted relational variational inference. In *Proceedings of the Twenty-Eighth Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-12)*, pages 196–206. AUAI Press, 2012. 9

[30] J. Choi, E. Amir, and D. Hill. Lifted inference for relational continuous models. In *Proceedings of the Twenty-Sixth Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-10)*, pages 126–134. AUAI Press, 2010. 9, 57

[31] J. Choi, R. de Salvo Braz, and H. Bui. Efficient methods for lifted inference with aggregate factors. In *Proceeding of the Twenty-Fifth AAAI Conference on Artificial Intelligence (AAAI-11)*, San Francisco, California, USA, August 7–11, 2011 2011. AAAI Press. 9

[32] J. Choi, A. Guzman-Rivera, and E. Amir. Lifted relational Kalman filtering. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI-11)*, pages 2092–2099. AAAI Press, 2011. 11

[33] E. F. Codd. A relational model of data for large shared data banks. *Commununications of the ACM*, 13(6):377–387, June 1970. 13

[34] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 2nd edition, 2001. 130

[35] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. *The VLDB Journal*, 16(4):523–544, October 2007. 13

[36] G. Dantzig and M. Thapa. *Linear Programming 2: Theory and Extensions*. Springer, 2003. 75

[37] A. Darwiche. Recursive conditioning. *Artificial Intelligence*, 126(1–2):5–41, February 2001. 10

[38] L. De Raedt. *Logical and Relational Learning*. Springer, 2008.

[39] L. De Raedt, P. Frasconi, K. Kersting, and S.H. Muggleton, editors. *Probabilistic Inductive Logic Programming*, volume 4911 of *Lecture Notes in Computer Science*. Springer, 2008. 2, 12

[40] R. de Salvo Braz, E. Amir, and D. Roth. Lifted first-order probabilistic inference. In *In Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI-05)*, pages 1319–1325. Morgan Kaufmann, 2005. 9, 114

[41] R. de Salvo Braz, E. Amir, and D. Roth. MPE and Partial Inversion in Lifted Probabilistic Variable Elimination. In *Proceedings of the 21st AAAI Conference on Artificial Intelligence (AAAI-06)*, 2006. 9

[42] R. de Salvo Braz, S. Natarajan, H. Bui, , J. Shavlik, and S. Russell. Anytime lifted belief propagation. In *Working Notes of the International Workshop on Statistical Relational Learning (SRL-09)*, Leuven, Belgium, 2009. 4

[43] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Commununications of the ACM*, 51(1):107–113, 2008. 30, 128

[44] T. Dean and R. Givan. Model minimization in markov decision processes. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, pages 106–111. AAAI Press, 1997. 3, 11, 80

[45] A. L. Delcher, A. J. Grove, S. Kasif, and J. Pearl. Logarithmic-time updates and queries in probabilistic networks. *Journal of Artificial Intelligence Research*, 4:37–59, 1996. 83

[46] 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. 12

[47] F. Eaton and Z. Ghahramani. Choosing a variable to clamp: Approximate inference using conditioned belief propagation. In *Proceedings of the 12th International Conference on Artificial Intelligence and Statistics (AIStats-09)*, 2009. 89

[48] P. Erdös and A. Rényi. Asymmetric graphs. *Acta Mathematica Academiae Scientiarum Hungarica*, 14:295–315, 1963. 136

[49] P. Flener, J. Pearson, and M. Sellmann. Static and dynamic structural symmetry breaking. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming (CP-06)*, pages 695–699. Springer, 2006. 11

[50] J. H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, pages 1189–1232, 2001. 2, 12

[51] A. C. Gallagher, D. Batra, and D. Parikh. Inference for order reduction in markov random fields. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR-11)*, pages 1857–1864. IEEE, June 2011. 21

[52] L. Getoor, N. Friedman, D. Koller, and B. Taskar. Learning probabilistic models of link structure. *Journal of Machine Learning Research*, 3:679–707, 2002. 12

[53] L. Getoor and B. Taskar, editors. *An Introduction to Statistical Relational Learning*. MIT Press, 2007. 2, 12, 15

[54] A. Globerson and T. Jaakkola. Fixing max-product: Convergent message passing algorithms for map LP-relaxations. In *Proceedings of the 21st Annual Conference on Neural Information Processing Systems (NIPS)*, 2007. 63, 66, 77, 138

[55] C. D. Godsil and G. Royle. *Algebraic Graph Theory*. Springer, 2007. 28, 29

[56] V. Gogate and P. Domingos. Exploiting logical structure in lifted probabilistic inference. In *Working Notes of the AAAI-10 Workshop on Statistical Relational Artificial Intelligence (StaRAI)*, 2010. 10

[57] V. Gogate and P. Domingos. Probabilistic theorem proving. In *Proceedings of the Twenty-Seventh Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-11)*, pages 256–265, Corvallis, Oregon, 2011. AUAI Press. 10, 11

[58] J. E. Gonzalez, Y. Low, and C. Guestrin. Residual splash for optimally parallelizing belief propagation. In *Proceeding of the Twelfth International Conference on Artificial Intelligence and Statistics (AISTATS-09)*, pages 177–184, 2009. 4, 8, 13, 128, 132

[59] J. E. Gonzalez, Y. Low, C. Guestrin, and D. O'Hallaron. Distributed parallel inference on large factor graphs. In *Proceedings of the Twenty-Fifth Conference*

Annual Conference on Uncertainty in Artificial Intelligence (UAI-09), pages 203–212, Corvallis, Oregon, July 2009. AUAI Press. 4, 8, 13, 128, 132

[60] B. Gutmann, I. Thon, and L. De Raedt. Learning the parameters of probabilistic logic programs from interpretations. In *Proceedings of the European Conference Machine Learning and Knowledge Discovery in Databases (ECML-PKDD-11)*, pages 581–596, 2011. 12

[61] F. Hadiji, B. Ahmadi, and K. Kersting. Efficient sequential clamping for lifted message passing. In *Proceedings of the 34th Annual German Conference on Artificial Intelligence (KI–11)*, Berlin, 2011. Springer. 11, 138

[62] W. H. Haemers. Interlacing eigenvalues and graphs. *Linear Algebra and its Applications*, 226/228:593–616, 1995. 72

[63] B. Hendrickson, R. Leland, and S. Plimpton. An efficient parallel algorithm for matrix-vector multiplication. *International Journal of High Speed Computing*, 7:73–88, 1995. 13

[64] K. Herr and R. Bödi. Symmetries in linear and integer programs. *CoRR*, abs/0908.3329, 2010. 3, 11, 135

[65] G. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8):1771–1800, August 2002. 12

[66] S. Hölldobler and O. Skvortsova. A logic-based approach to dynamic programming. In *In In AAAI-04 Workshop on Learning and Planning in Markov Processes – Advances and Challenges*, pages 31–36. AAAI Press, 2004. 11

[67] R. A. Horn and C. A. Johnson, editors. *Matrix Analysis*. Cambridge University Press, 1985. 72

[68] T. N. Huynh and R. Mooney. Online max-margin weight learning for markov logic networks. In *In Proceedings of the Eleventh SIAM International Conference on Data Mining (SDM-11)*, pages 642–651, Mesa, Arizona, USA, April 2011. 4, 12

[69] A. T. Ihler and D. A. McAllester. Particle belief propagation. *Journal of Machine Learning Research - Proceedings Track*, 5:256–263, 2009. 138

[70] A.T. Ihler, J.W. Fisher III, and A.S. Willsky. Loopy Belief Propagation: Convergence and Effects of Message Errors. *Journal of Machine Learning Research*, 6:905–936, 2005. 6, 40, 50, 51

[71] M. Jaeger. Parameter learning for Relational Bayesian networks. In *Proceedings of the 24th International Conference on Machine learning (ICML-07)*, pages 369–376, New York, NY, USA, 2007. ACM. 12, 15

[72] A. Jaimovich, O. Meshi, and N. Friedman. Template-based inference in symmetric relational Markov random fields. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI-07)*, pages 191–199. AUAI Press, 2007. 9, 38

[73] R. Jonker and A. Volgenant. A shortest augmenting path algorithm for dense and sparse linear assignment problems. *Computing*, 38(4), November 1987. 106

[74] L. Pack Kaelbling, M. Littman, and A. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996. 78

[75] K. Kersting. Lifted probabilistic inference. In L. De Raedt, C. Bessiere, D. Dubois, P. Doherty, P. Frasconi, F. Heintz, and P. Lucas, editors, *Proceedings of 20th European Conference on Artificial Intelligence (ECAI–2012)*, Montpellier, France, August 27–31, 2012 2012. ECCAI, IOS Press. (Invited Talk at the Frontiers of AI Track).

[76] K. Kersting, B. Ahmadi, and S. Natarajan. Counting belief propagation. In *Proceedings of the Twenty-Fifth Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-09)*, pages 277–284, Montreal, Canada, 2009. AUAI Press.

[77] K. Kersting and L. De Raedt. Adaptive bayesian logic programs. In *Proceedings of the 11th International Conference on Inductive Logic Programming (ILP-01)*, pages 104–117. Springer-Verlag, 2001. 12, 15

[78] K. Kersting, Y. El Massaoudi, B. Ahmadi, and F. Hadiji. Informed lifting for message–passing. In D. Poole M. Fox, editor, *Twenty–Fourth AAAI Conference on Artificial Intelligence (AAAI–10)*, Atlanta, USA, July 11 – 15 2010. AAAI Press.

[79] K. Kersting, M. Van Otterlo, and L. De Raedt. Bellman goes relational. In Carla E. Brodley, editor, *Proceedings of the Twenty–First International Conference on Machine Learning (ICML–04)*, pages 465–472, Banff, Alberta, Canada, July 4–8 2004. 11

[80] T. Khot, S. Natarajan, K. Kersting, and J. Shavlik. Learning markov logic networks via functional gradient boosting. In *Proceedings of the IEEE International Conference on Data Mining (ICDM-11)*, pages 320–329, 2011. 12

[81] C. Kiddon and P. Domingos. Coarse-to-fine inference and learning for first-order probabilistic models. In *Proceedings of the 25th AAAI Conference on Artificial Intelligence (AAAI 2011)*, 2011. 138

[82] R. Kindermann and J. Snell. *Markov Random Fields and Their Applications (Contemporary Mathematics ; V. 1)*. American Mathematical Society, 1980. 21

[83] J. Kisynski and D. Poole. Constraint processing in lifted probabilistic inference. In *Proceedings of the Twenty-Fifth Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-09)*, pages 293–302. AUAI Press, 2009. 9

[84] S. Kok and P. Domingos. Learning Markov logic network structure via hypergraph lifting. In *Proceedings of the 26th Annual International Conference on Machine Learning (ICML-09)*, pages 505–512, Montreal, Quebec, Canada, 2009. ACM. 12

[85] S. Kok and P. Domingos. Learning Markov logic networks using structural motifs. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 551–558, Haifa, Israel, 2010. 12

[86] D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009. 22, 64, 65

[87] V. Kolmogorov. Convergent tree-reweighted message passing for energy minimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(10):1568–1583, 2006. 138

[88] V. Kolmogorov and C. Rother. Minimizing nonsubmodular functions with graph cuts-a review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(7):1274–1279, 2007. 138

[89] L. Kroc, A. Sabharwal, and B. Selman. Leveraging Belief Propagation, Backtrack Search, and Statistics for Model Counting. In *Proceedings of the 5th International Conference on the Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR-08)*, pages 127–141, 2008. 47

[90] S. Kudose. Equitable partitions and orbit partitions. *Acta Mathematica Sinica*, pages 1–9, 2009. 29

[91] L. V. S. Lakshmanan, N. Leone, R. Ross, and V. S. Subrahmanian. Probview: A flexible probabilistic database system. *ACM Transactions on Database System*, 22(3):419–469, 1997. 13

[92] J. Langford, E. J. Smola, and M. Zinkevich. Slow learners are fast. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2331–2339, 2009. 127

[93] N. Le Roux, P.-A. Manzagol, and Y. Bengio. Topmoumoute online natural gradient algorithm. In *Advances in Neural Information Processing Systems 20 (NIPS)*, page 849856, 2007. 121

[94] S.-I. Lee, V. Ganapathi, and D. Koller. Efficient structure learning of Markov networks using L1-regularization. In *In Advances in Neural Information Processing Systems 19 (NIPS)*, pages 817–824, 2007. 12, 121

[95] M. L. Littman, T. L. Dean, and L. Pack Kaelbling. On the complexity of solving Markov decision problems. In *Proceedings of the 11th International Conference on Uncertainty in Artificial Intelligence (UAI-95)*, pages 394–402, 1995. 78

[96] D. Lowd and P. Domingos. Efficient weight learning for markov logic networks. In *Proceedings of the 11th European conference on Principles and Practice of Knowledge Discovery in Databases*, PKDD 2007, pages 200–211. Springer-Verlag, 2007. 12

[97] F. Margot. Symmetry in integer linear programming. In M. Jünger, T.M. Liebling, D. Naddef, G.L. Nemhauser, W.R. Pulleyblank, G. Reinelt, G. Rinaldi, and L.A. Wolsey, editors, *50 Years of Integer Programming 1958–2008: From the Early Years to the State-of-the-Art*, pages 1–40. Springer, 2010. 3, 11, 136

[98] B. D. McKay. Practical graph isomorphism. *Congressus Numerantium*, 30(30):45–87, 1981. 29

[99] M. Mézard and A. Montanari. *Information, Physics, and Computation*. Oxford University Press, Inc., New York, NY, USA, 2009. 81

[100] M. Mézard, G. Parisi, and R. Zecchina. Analytic and algorithmic solution of random satisfiability problems. *Science*, 297:812–815, 2002. 83

[101] L. Mihalkova, T. N. Huynh, and R. J. Mooney. Mapping and revising Markov logic networks for transfer learning. In *Twenty-Second Conference on Artificial Intelligence (AAAI-07)*, pages 608–614, 2007. 5

[102] B. Milch and S. J. Russell. General-purpose MCMC inference over relational structures. In *Proceedings of the 22nd Conference in Uncertainty in Artificial Intelligence (UAI-2006)*, pages 349–358. AUAI Press, 2006. 10, 11

[103] B. Milch, L. Zettlemoyer, K. Kersting, M. Haimes, and L. Pack Kaelbling. Lifted Probabilistic Inference with Counting Formulas. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI-08)*, July 13-17 2008. 9

[104] M. Mladenov, B. Ahmadi, and K. Kersting. An implementation of lifted linear programming. http://www-kd.iai.uni-bonn.de/index.php?page=software_details&id=25, 2012. 76

[105] M. Mladenov, B. Ahmadi, and K. Kersting. Lifted linear programming. In *15th International Conference on Artificial Intelligence and Statistics (AISTATS 2012)*, pages 788–797, La Palma, Canary Islands, Spain, April 21–23, 2012 2012. Volume 22 of JMLR: W&CP 22.

[106] M. Mladenov and K. Kersting. Lifted inference via k-locality. In *Working Notes of the AAAI-13 Workshop on Statistical Relational Artificial Intelligence (StaRAI)*, 2013. 10

[107] M. Møller. A scaled conjugate gradient algorithm for fast supervised learning. *NEURAL NETWORKS*, 6(4):525–533, 1993. 121

[108] A. Montanari, F. Ricci-Tersenghi, and G. Semerjian. Solving constraint satisfaction problems through belief propagation-guided decimation. In *Proceedings of the 45th Allerton Conference on Communications, Control and Computing*, 2007. 81

[109] J. M. Mooij. libDAI: A free and open source C++ library for discrete approximate inference in graphical models. *Journal of Machine Learning Research*, 11:2169–2173, August 2010. 53, 91

[110] J. M. Mooij, B. Wemmenhove, H. Kappen, and T. Rizzo. Loop corrected belief propagation. In *Proceedings of the 11th International Conference on Artificial Intelligence and Statistics (AIStats-09)*, 2007. 89

[111] K.P. Murphy and Y. Weiss. The Factored Frontier Algorithm for Approximate Inference in DBNs. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI-01)*, pages 378–385, 2001. 45

[112] K.P. Murphy, Y. Weiss, and M.I. Jordan. Loopy Belief Propagation for Approximate Inference: An Empirical Study. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI-99)*, pages 467–475, 1999. 25

[113] S.M. Narayanamurthy and B. Ravindran. On the hardness of finding symmetries in markov decision processes. In *Proceedings of the 25th International Conference on Machine Learning (ICML-08)*, pages 688–695, 2008. 80

[114] S. Natarajan, T. Khot, K. Kersting, B. Guttmann, and J. Shavlik. Gradient-based boosting for statistical relational learning: The relational dependency network case. *Machine Learning*, 86(1):25–56, January 2012. 12

[115] S. Natarajan, P. Tadepalli, T. G Dietterich, and A. Fern. Learning first-order probabilistic models with combining rules. *Annals of Mathematics and Artificial Intelligence*, 54(1–3):223–256, November 2008. 12

[116] A. Nath and P. Domingos. Efficient lifting for online probabilistic inference. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI-10)*, Atlanta, Georgia, USA, July 2010. 4, 83

[117] M. Neumann, K. Kersting, and B. Ahmadi. Markov logic sets: Towards lifted information retrieval using pagerank and label propagation. In D. Roth W. Burgard, editor, *Proceedings of the Twenty–Fifth AAAI Conference on Artificial Intelligence (AAAI–11)*, San Francisco, CA, USA, August 7 – 11 2011. AAAI Press. 11

[118] J. Neville and D. Jensen. Relational dependency networks. *Journal of Machine Learning Research*, 8:653–692, 2007. 15

[119] M. Niepert. Markov chains on orbits of permutation groups. In *Proceedings of the 28th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 624–633, Catalina Island, California, USA, August 15–17 2012. AUAI Press. 10, 11

[120] M. Niepert. Symmetry-aware marginal density estimation. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence (AAAI-13)*. AAAI Press, July 2013. 10, 11

[121] F. Niu, C. Ré, A. Doan, and J. Shavlik. Tuffy: Scaling up statistical inference in markov logic networks using an rdbms. *PVLDB*, 4(6):373–384, 2011. 13

[122] J. Noessner, M. Niepert, and H. Stuckenschmidt. Rockit: Exploiting parallelism and symmetry for map inference in statistical relational models. *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence (AAAI-13)*, 2013. 13

[123] J.D. Park. MAP complexity results and approximation methods. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI-02)*, pages 388–396, 2002. 83

[124] J. Pearl. *Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 2. edition, 1991. 4, 19, 20, 21

[125] N. Piatkowski and K. Morik. Parallel loopy belief propagation in conditional random fields. In *Working Notes of the LWA 2011 - Learning, Knowledge, Adaptation*, 2011. 13

[126] D. Poole. First-Order Probabilistic Inference. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-05)*, pages 985–991, 2003. 9

[127] D. Poole, F. Bacchus, and J. Kisyński. Towards completely lifted search-based probabilistic inference. *CoRR*, abs/1107.4035, 2011. 10

[128] H. Poon and P. Domingos. Joint unsupervised coreference resolution with markov logic. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP-08)*, pages 650–659, Stroudsburg, PA, USA, 2008. Association for Computational Linguistics. 12

[129] H. Poon and P. Domingos. Sum-product networks: A new deep architecture. In *Proceedings of the Twenty-Seventh Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-11)*, pages 337–346. AUAI Press, 2011. 10

[130] F. Potra and S. J. Wright. Interior-point methods. *Journal of Computational and Applied Mathematics*, 124:281–302, 2000. 69

[131] B. Ravindran and A. G. Barto. Symmetries and model minimization in Markov decision processes. Technical Report 1–43, University of Massachusetts, Amherst, MA, USA, 2001. 3, 11, 80

[132] C. Ré, N. Dalvi, and D. Suciu. Efficient top-k query evaluation on probabilistic data. In *Proceedings of the 23rd International Conference on Data Engineering (ICDE-07)*, pages 886–895, 2007. 13

[133] B. L. Richards and R. J. Mooney. Learning relations by pathfinding. In *Proceedings of the tenth national conference on Artificial intelligence (AAAI-92)*, pages 50–55, San Jose, California, 1992. AAAI Press. 114

[134] M. Richardson and P. Domingos. Markov Logic Networks. *Machine Learning*, 62:107–136, 2006. 11, 12, 16, 66, 104

[135] J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, January 1965. 17

[136] F. Rosenblatt. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Spartan, 1962. 5

[137] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 edition, 2003. 125

[138] S. Sanner and K. Kersting. Symbolic dynamic programming for first–order pomdps. In D. Poole M. Fox, editor, *Twenty–Fourth AAAI Conference on Artificial Intelligence (AAAI–10)*, Atlanta, USA, July 11 – 15 2010. AAAI Press. 11

[139] A. D. Sarma, O. Benjelloun, A. Halevy, and J. Widom. Working models for uncertain data. In *Proceedings of the 22nd International Conference on Data Engineering (ICDE-06)*, Washington, DC, USA, 2006. IEEE Computer Society. 13

[140] T. Sato and Y. Kameya. Parameter learning of logic programs for symbolic-statistical modeling. *Journal of Artificial Intelligence Research (JAIR)*, 15:391–454, 2001. 12

[141] N. Schraudolph and T. Graepel. Combining conjugate direction methods with stochastic approximation of gradients. In *Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics (AISTATS-03)*, pages 7–13, Key West, Florida, 2003. 120

[142] M. Sellmann and P. Van Hentenryck. Structural symmetry breaking. In *Proceedings of 19th International Joint Conference on Artificial Intelligence (IJCAI-05)*, 2005. 3, 11, 136

[143] P. Sen, A. Deshpande, and L. Getoor. Exploiting shared correlations in probabilistic databases. In *Proceedings of the International Conference on Very Large Data Bases (VLDB-08)*, pages 809–820, 2008. 9

[144] O. Shental, D. Bickson, P. H. Siegel, J. K. Wolf, and D. Dolev. Gaussian belief propagation solver for systems of linear equations. In *IEEE International Symposium on Information Theory (ISIT)*, Toronto, Canada, July 2008. 57, 58, 59, 96

[145] A. Shirazi and E. Amir. First-order logical filtering. *Artificial Intelligence*, 175(1):193–219, January 2011. 11

[146] P. Singla and P. Domingos. Lifted First-Order Belief Propagation. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI-08)*, pages 1094–1099, Chicago, IL, USA, July 13–17 2008. 4, 9, 23, 38, 42, 43, 44, 52, 91, 123

[147] D. Sontag, A. Globerson, and T. Jaakkola. Clusters and coarse partitions in LP relaxations. In *Proceedings of the 22nd Annual Conference on Neural Information Processing Systems (NIPS)*, pages 1537–1544, 2008. 138

[148] E. B. Sudderth, A. T. Ihler, W. T. Freeman, and A. S. Willsky. Nonparametric belief propagation. In *Proceedings of the 2003 IEEE computer society conference on Computer vision and pattern recognition (CVPR-03)*, pages 605–612. IEEE Computer Society, 2003. 138

[149] C. Sutton and A. McCallum. Piecewise training for structured prediction. *Machine Learning*, 77(2–3):165–194, 2009. 12, 26, 118

[150] R. S. Sutton and A. G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998. 78

[151] N. Taghipour, D. Fierens, J. Davis, and H. Blockeel. Lifted variable elimination with arbitrary constraints. *Journal of Machine Learning Research - Proceedings Track*, 22:1194–1202, 2012. 9

[152] N. Taghipour, D. Fierens, J. Davis, and H. Blockeel. Lifted variable elimination: Decoupling the operators from the constraint language. *Journal of Artificial Intelligence Research (JAIR)*, 47:393–439, July 2013. 9

[153] I. Thon, N. Landwehr, and L. De Raedt. Stochastic relational processes: Efficient inference and applications. *Machine Learning*, 82(2):239–272, 2011. 12

[154] G. Van den Broeck. On the completeness of first-order knowledge compilation for lifted probabilistic inference. In *Advances in Neural Information Processing Systems 24 (NIPS)*, pages 1386–1394, 2011. 10

[155] G. Van den Broeck, A. Choi, and A. Darwiche. Lifted relax, compensate and then recover: From approximate to exact lifted probabilistic inference. In *Proceedings of the Twenty-Eighth Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-12)*, pages 131–141. AUAI Press, 2012. 10

[156] G. Van den Broeck and J. Davis. Conditioning in first-order knowledge compilation and lifted probabilistic inference. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence (AAAI-2012)*, 2012. 10, 83

[157] G. Van den Broeck, N. Taghipour, W. Meert, J. Davis, and L. De Raedt. Lifted probabilistic inference by first-order knowledge compilation. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI-11)*, pages 2178–2185, 2011. 10

[158] D. Venugopal and V. Gogate. On lifting the Gibbs sampling algorithm. In *Advances in Neural Information Processing Systems 25 (NIPS)*, pages 1664–1672, 2012. 10, 11

[159] S. V. N. Vishwanathan, N. N. Schraudolph, M. W. Schmidt, and K. P. Murphy. Accelerated training of conditional random fields with stochastic gradient methods. In *In Proceedings of the 23rd International Conference on Machine Learning (ICML-06)*, pages 969–976. ACM Press, 2006. 120

[160] M. J. Wainwright, T. S. Jaakkola, and A. S. Willsky. A new class of upper bounds on the log partition function. In *Proceedings of the Eighteenth Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-02)*, pages 536–543. Morgan Kaufmann, 2002. 119

[161] M. J. Wainwright, T. S. Jaakkola, and A. S. Willsky. Tree-reweighted belief propagation algorithms and approximate ml estimation by pseudo-moment matching. In *Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics (AISTATS-03)*, 2003. 138

[162] Y. Weiss and W.T. Freeman. Correctness of belief propagation in gaussian graphical models of arbitrary topology. *Neural Computation*, 13(10):2173–330, 2001. 58

[163] M. Welling and Y.W. Teh. Linear response for approximate inference. In *Advances in Neural Information Processing Systems (NIPS)*, pages 191–199, 2003. 84, 91

[164] H. Weyl. *Symmetry*. Princeton University Press, 1952. 135

[165] G. Winkler. *Image Analysis, Random Fields and Dynamic Monte Carlo Methods*. Springer-Verlag, 1995. 11

[166] C. Yanover, T. Meltzer, and Y. Weiss. Linear programming relaxations and belief propagation an empirical study. *Journal of Machine Learning Research*, 7:1887–1907, 2006. 66, 138

[167] J. S. Yedidia, W. T. Freeman, and Y. Weiss. Generalized belief propagation. In *Advances in Neural Information Processing Systems (NIPS)*, pages 689–695. MIT Press, 2000. 138

[168] D.W. Zaidel and M. Hessamian. Asymmetry and symmetry in the beauty of human faces. *Symmetry*, 2:136–149, 2010. 135

[169] Z. Zamani, S. Sanner, P. Poupart, and K. Kersting. Symbolic dynamic programming for continuous state and observation pomdps. In *Advances in Neural Information Processing Systems 26 (NIPS–2012)*, Lake Tahoe, Nevada, USA, December 3–6 2012. MIT Press. 11

[170] L. S. Zettlemoyer, H. M. Pasula, and L. Pack Kaelbling. Logical particle filtering. In *Proceedings of the Dagstuhl Seminar on Probabilistic, Logical, and Relational Learning*, 2007. 10, 11

[171] S. Zhu, Z. Xiao, H. Chen, R. Chen, W. Zhang, and B. Zang. Evaluating splash-2 applications using mapreduce. In *Proceedings of the 8th International Symposium on Advanced Parallel Processing Technologies*, APPT '09, pages 452–464. Springer-Verlag, 2009. 130

[172] M. A. Zinkevich, A. Smola, M. Weimer, and L. Li. Parallelized stochastic gradient descent. In *Advances in Neural Information Processing Systems 23 (NIPS)*, pages 2595–2603, 2010. 4, 5, 8, 127