Ludwig-Maximilians-University Munich

Department of Statistics - Master Seminar WS15/16

Probabilistic Graphical Models

Inference

Johannes Langer

Advisor:
M.Sc. Eva Endres
M.Sc. Paul Fink

Date: 04.03.2016

Abstract

Inference with highly multivariate joint distributions is in many cases intractable. Such cases require to factorize the joint distribution into smaller factors, which we can learn, store and recombine. This requires conditional independence assumptions, which are encoded in Probabilistic Graphical Models. Sparser Graphical Networks thereby correspond to more conditional independence assumptions, which lead to smaller and more local structures. Although this approach can cause arbitrary assumptions and wrong inference it is often necessary in terms of computation. To derive queried conditional probability distributions or maximum a posteriori (MAP) values the recombination of the separated information becomes necessary and the computational intractabilities return once again. To avoid exponential blow-ups of calculation special graph structures are used (e.g. Clique Tree Algorithm) or exact inference is dropped (e.g. Loopy Belief Propagation, Sampling Methods). This introduction to inference with Probabilistic Graphical Models is focused on exact inference methods and only scratches approximate methods based on sampling or other iterative models. After introducing factors and queries, the general complexity of working with highly multivariate distributions is illustrated. As mentioned the focus of this work lies on exact inference algorithms, namely the Variable Elimination Algorithm and the Clique Tree Algorithm, which is introduced in a broader perspective as a Message Passing Algorithm and a Dynamic Programming method. Only CPD queries and discrete Networks will be discussed. Also important to mention is that this work is strongly based on the book "Probabilistic Graphical Models: Principles and Techniques" written by D. Koller and N. Friedman.

Contents

1 Factors

To emphasize the underlying calculations during PGM inference algorithms the notion factor is used, which generalizes distributions to unnormalized measures:

Definition 1. Let $\mathbf{X}$ be a set of random variables. We define a **factor** Φ to be a function $\Phi(\mathbf{X}) : Val(\mathbf{X}) \to \mathbb{R}^+$. The set of variables $\mathbf{X}$ is called the scope of the factor and denoted $Scope[\Phi]$. [1, ch. 4.1]

Normalization of a factor (unnormalized measure) gives back a distribution (normalized measure), normally called Gibbs Distribution:

Definition 2. Given a factor $\Phi(\mathbf{X})$ we define a **Gibbs Distribution** P to be a function $P(\mathbf{X}) : Val(\mathbf{X}) \to [0, 1]$, derived by $P(\mathbf{X}) = \frac{1}{Z}\Phi(\mathbf{X})$. The normalization constant $Z = \sum_{\mathbf{x} \in Val(\mathbf{X})} \Phi(\mathbf{x})$ is called partition function. [1, ch. 4.2]

An example of a factor Φ and the corresponding distribution P is shown in figure 1.

factor
$\Phi(\mathbf{X}) : Val(\mathbf{X}) \to \mathbb{R}^+$
i.g. unnormalized measure

Table 1: $\Phi(A, B)$

entry	value
a_0, b_0	π
a_0, b_1	0.1
a_1, b_0	1
a_1, b_1	1000

distribution
$P(\mathbf{X}) : Val(\mathbf{X}) \to [0, 1]$
normalized measure

Table 2: $\frac{1}{Z}\Phi(A, B)$

entry	value
a_0, b_0	$3 \cdot 10^{-3}$
a_0, b_1	$1 \cdot 10^{-4}$
a_1, b_0	$1 \cdot 10^{-3}$
a_1, b_1	0.9959

Figure 1: Factor and Gibbs Distribution example.

1.1 Reduction

Factor Reduction is an intermediate step to achieve a conditional probability distribution (CPD) with given **evidence**. If some variables of the scope $\mathbf{X}$ are given, meaning they are set to fixed values, the number of factor entries is reduced, causing the name **factor reduction**. Normalization of a reduced factor gives back the CPD (example in figure 2) [1, ch. 4.2] :

$$P(\mathbf{Y}|\mathbf{E} = \mathbf{e}) = \frac{1}{Z}\Phi(\mathbf{Y}, \mathbf{E} = \mathbf{e}) \tag{1.1}$$

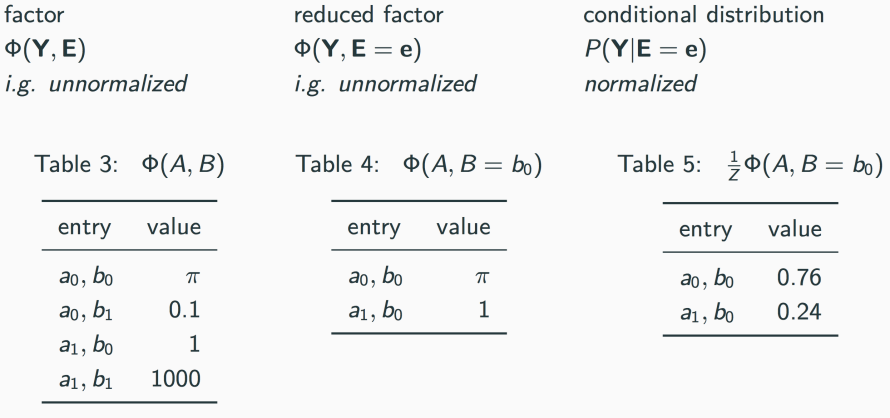


Figure 2: Factor Reduction example.

1.2 Marginalization

Factor Marginalization is an intermediate step to achieve a marginal distribution over a subset of the scope. For a fixed instantiation of the remaining variables there is one entry for each value of the variable to eliminate. If all those entries are summed up, the variable vanishes from the factor. Summing some variables of the scope $\mathbf{X}$ out of the factor means that the number of factor entries is again reduced, leading to a marginalized factor. 'Calculating the marginal for $\mathbf{Y}$' or 'Summing out the variables $\mathbf{W} = \mathbf{X} \backslash \mathbf{Y}$' indicate the same procedure. Normalization of a marginalized factor gives back a marginal distribution (example in figure 3)[1, ch. 9.3]:

$$P(\mathbf{Y}) = \frac{1}{Z} \sum_{\mathbf{W}} \Phi(\mathbf{Y}, \mathbf{W}) \tag{1.2}$$

Further we want to discuss the operations needed to marginalize variable X_j out of a factor $\Phi(\mathbf{X})$. Let $d_j = |Val(X_j)|$ be the domain cardinality of variable X_j, $N_j = \prod_{i=1}^{n} d_i$ be the number of entries of the factor before summing out X_j and $d = \max_k d_k$ be an upper bound for the domain cardinalities of all variables. For every entry in the resulting factor it takes $d_j - 1$ additions to sum out the variable X_j. The number of entries of the original factor N_j divided by d_j gives the number of entries of the resulting factor leading to the total number of additions [1, ch. 9.4]:

$$N_{add} = \frac{d_j - 1}{d_j} \prod_{i=1}^{n} d_i < d^n \tag{1.3}$$

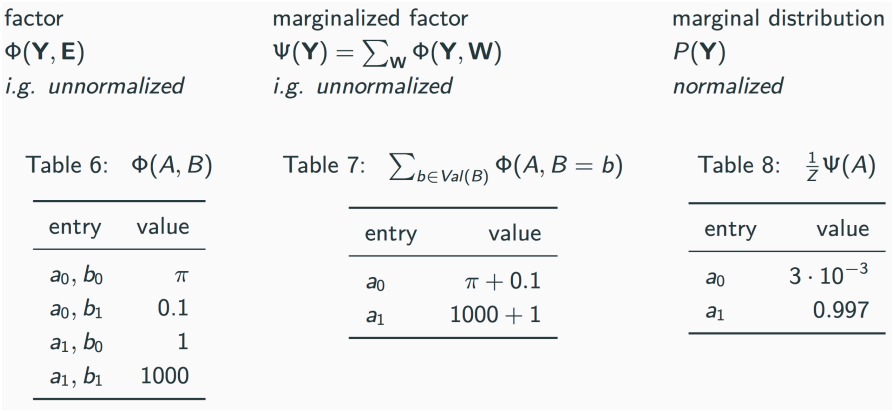


Figure 3: Factor Marginalization example.

1.3 Product

For every factor in the product there has to exist at least one other factor sharing at least one same variable, since only then the factor product is defined [1, ch. 4.2]:

$$\Psi(\mathbf{X}, \mathbf{Y}, \mathbf{Z}) = \Phi_1(\mathbf{X}, \mathbf{Z})\Phi_2(\mathbf{Y}, \mathbf{Z}) \tag{1.4}$$

The number of entries of a factor is a product of the domain cardinalities of all variables in the scope $N = \prod_{i=1}^{n} d_i$. Since the scope of the product always increases, the number of entries also increases: the product of smaller factors leads to a bigger factor. The example in figure 4 illustrates how the number of entries increases in the generated factor.

Hence we conclude that a product of many small factors can generate large factors, which is important to understand the computational cost and the complexity of the inference task. We further discuss the operations needed to calculate a general factor product of the type:

$$P(\mathbf{X}) \propto \prod_{k=1}^{K} \Phi_k(\mathbf{D_k}) \tag{1.5}$$

For every entry in the resulting factor we have to multiply K values, which takes $K - 1$ multiplications. Let again d_k be the domain cardinality of variable X_k and $d = \max_k d_k$ be an upper bound for the domain cardinalities of all variables, then the total number of necessary operations is [1, ch. 9.4]:

$$N_{prod} = (K - 1) \prod_{i=1}^{n} d_i \leq (K - 1)d^n \tag{1.6}$$

$\Phi_1(A, C)$

entry	value
a_0, c_0	1
a_0, c_1	2
a_1, c_0	3
a_1, c_1	4

$\Phi_2(B, C)$

entry	value
b_0, c_0	4
b_0, c_1	3
b_1, c_0	2
b_1, c_1	1

factor product $\Psi(\mathbf{X}, \mathbf{Y}, \mathbf{Z})$
$= \Phi_1(\mathbf{X}, \mathbf{Z})\Phi_2(\mathbf{Y}, \mathbf{Z})$

Table 9: $\Psi(A, B, C)$

entry	value
a_0, b_0, c_0	$1 \cdot 4$
a_0, b_0, c_1	$2 \cdot 3$
a_0, b_1, c_0	$1 \cdot 2$
a_0, b_1, c_1	$2 \cdot 1$
a_1, b_0, c_0	$3 \cdot 4$
a_1, b_0, c_1	$4 \cdot 3$
a_1, b_1, c_0	$3 \cdot 2$
a_1, b_1, c_1	$4 \cdot 1$

joint distribution
$P(\mathbf{X}, \mathbf{Y}, \mathbf{Z})$

Table 10: $P(A, B, C)$

entry	value
a_0, b_0, c_0	0.083
a_0, b_0, c_1	0.125
a_0, b_1, c_0	0.042
a_0, b_1, c_1	0.042
a_1, b_0, c_0	0.250
a_1, b_0, c_1	0.250
a_1, b_1, c_0	0.125
a_1, b_1, c_1	0.083

Figure 4: Factor Product example.

1.4 Factorization

Factorization serves as the reverse operation of generating a factor product by breaking down a big factor into a product of smaller factors. In practice we will always start with smaller factors and try to generate bigger factor products. But the factorization has a deeper meaning, in fact it corresponds with independence assumptions made over the variables. To understand this we remember that graphical models are a way to encode conditional independence assumptions [3] and recapitulate the definitions of graphical networks. For a Bayesian Network we know that a variable X_k given its parents in the graph $Pa^{\mathcal{G}}_{X_k}$ is independent of all non-descendant variables [1, ch. 3.2]:

Definition 3. A distribution $P(\mathbf{X}) := P(X_1, ..., X_n)$ factorizes over a **Bayesian Network** $\mathcal{G}$, if P can be expressed as a product:

$$P_{\mathcal{G}}(X_1, ..., X_n) = \prod_{k=1}^{n} P_k(X_k | Pa^{\mathcal{G}}_{X_k}) \overset{as\ factor}{=} \prod_{k=1}^{n} \Phi_k(X_k, Pa^{\mathcal{G}}_{X_k}) \tag{1.7}$$

The graph structure encodes independence assumptions and therefore also a factorization of the bigger joint distribution into smaller conditional probability distributions. The same thing applies for undirected graphs called Markov Networks [1, ch. 4.2]:

Definition 4. A distribution $P(\mathbf{X})$ factorizes over a **Markov Network** $\mathcal{H}$, if each

subset of variables $\mathbf{D_k} \subset \mathbf{X}$ $k \in (1, ..., K)$ builds a complete subgraph (clique) of $\mathcal{H}$:

$$P_{\mathcal{H}}(X_1, ..., X_n) = \frac{1}{Z}\Phi(X_1, ..., X_n) \propto \prod_{k=1}^{K} \Phi_k(\mathbf{D_k}) \tag{1.8}$$

This means that all scopes $\mathbf{D_k}$ have to be fully connected in the Markov Network. Both graph networks lead to a factorization of a bigger factor into smaller factors. This feature is mandatory to treat factors over a large scope of variables, as will be discussed in section Complexity 3. Further basic definitions can be found in section Appendix 7. We close this chapter with the definition of moralization of a graph, which is also an important concept for inference [1, ch. 4.5]:

Definition 5. The **moral graph** $\mathcal{M}[G]$ of $\mathcal{G}$ is an undirected graph containing edges between X and Y if there is a directed edge between them or they are both parents of the same node.

Figure 5 shows a corresponding moral graph of a Bayesian Network. The name moral stems from the mandatory 'marrying' of the parents.

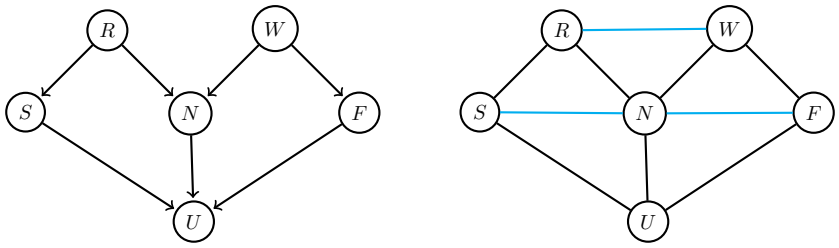


Figure 5: Bayesian Network and corresponding Moral Graph

2 Queries

Given are all factors or distributions which factorize the joint into K factors over subsets of variables $\mathbf{D}_k \subset \mathbf{X}$ we get:

$$P(\mathbf{X}) \propto \prod_{k=1}^{K} \Phi_k(\mathbf{D_k}) \tag{2.1}$$

Further, the subsets of variables $\mathbf{Y}, \mathbf{E}, \mathbf{W}$ build a partition of the scope: $\mathbf{X} = \mathbf{Y} \cup \mathbf{E} \cup \mathbf{W}$ and $\mathbf{Y} \cap \mathbf{E} \cap \mathbf{W} = \emptyset$.

2.1 Conditional Probability Density

The most common query is about the **Conditional Probability Density** (CPD) [1, ch. 9]:

$$P(\mathbf{Y} \mid \mathbf{E} = \mathbf{e}) = \frac{P(\mathbf{Y}, \mathbf{E} = \mathbf{e})}{P(\mathbf{E} = \mathbf{e})} \propto P(\mathbf{Y}, \mathbf{E} = \mathbf{e}) = \sum_{\mathbf{W}} P(\mathbf{Y}, \mathbf{W}, \mathbf{E} = \mathbf{e}) \propto \sum_{\mathbf{W}} \prod_{k=1}^{K} \Phi_k(\mathbf{D_k})[\mathbf{E} = \mathbf{e}] \tag{2.2}$$

The query searches the probability distribution of the variables of interest Y (e.g. diseases, next robot position, etc.) given the evidence $E = e$ (e.g. symptoms, robot observations, etc.) and ignoring variables W. We conclude that the derivation of the conditional probability distribution takes four steps:

1. **reduction** reduce all factors by evidence $\Phi(\mathbf{D_k})[\mathbf{E} = \mathbf{e}]$
2. **multiplication** calculate the factor product of the reduced factors
3. **addition** sum out all other variables $\sum_W$
4. **normalization** re-normalize the final factor to achieve a proper CPD

In the Variable Elimination chapter 4 we will discuss what drives the computational cost answering this query. Immediately noticeable is that a high number of given variables E lowers and a high number of query variables Y increases the overall cost. Just compare the cost of calculating the full conditional $P(X_1|X_{2:n} = \mathbf{e})$ with calculating $(X_{2:n}|X_1 = e)$, which is "close" to the cost of calculating the joint, since only one variable is given. The ease to calculate the full conditionals will later be used for Gibbs Sampling 6.2.

2.2 Maximum a Posteriori

Another very common query is about the maximum of the CPD $P(Y|E = e)$, called **Maximum a Posteriori** (MAP). Important to mention is that efficient algorithms answering this query avoid calculating the CPD. Y gets set to all non given variables and optimization methods try to find a maximum instantiation. This optimization leads to the same result for all strictly monotonic transformations, which is why we use a logarithmic transformation, which gets rid of the product [1, ch. 13.2]:

$$\hat{\mathbf{y}} = \underset{\mathbf{y}}{\operatorname{argmax}}\ P(\mathbf{Y} = \mathbf{y}|\mathbf{E} = \mathbf{e}) = \underset{y_1,\ldots,y_p}{\operatorname{argmax}}\ log \prod_k \Phi_k(\mathbf{D_k})[\mathbf{E} = \mathbf{e}] = \underset{y_1,\ldots,y_p}{\operatorname{argmax}}\ \sum_{k=1}^{K} log\Phi_k(\mathbf{D_k})[\mathbf{E} = \mathbf{e}] \tag{2.3}$$

We conclude that the MAP estimation takes three steps:

1. **reduction**: reduce all factors by evidence $\Phi(\mathbf{D_k})[\mathbf{E} = \mathbf{e}]$
2. **logarithm**: calculate the *log* for all entries
3. **search/optimize**: find a maximum instantiation of all non-given variables Y

The search task can be tackled with combinatorial optimization techniques. Note that only instantiations of the CPD get calculated during the search task, but not the complete CPD. [1, ch. 13.5]

3 Complexity

Example 1. Let's say we want to perform inference over 100 binary variables $X_1, ..., X_{100}$. The number of entries/values of the joint distribution over those variables would be 2^{100}. Further let's say the storage of one entry needs approximately 1 byte storage. Since $2^{10} = 1024 \sim 10^3$ this would take 10^{30} bytes or 10^{12} exabytes. The estimated storage of all Google servers in 2015 were 10 exabytes. Therefore 100 billion times the storage of all Google servers would be needed, just to store all entries of such a joint distribution.

As the scope of a factor gets bigger the number of its entries increases exponentially. This means that we can almost never calculate or store a factor over a big scope of variables. Therefore, calculation or storage of joint distributions is almost never possible. The only way to store the information of a big factor is to find a factorization into smaller factors. Thus factorization and therefore conditional independence assumptions are mandatory to work with joints over many variables. Note here that we cannot work without them. To really break down a big factor, we need as many conditional independence assumptions as possible, which in fact is similar to say we need a graph structure with as little connections as possible. And this is were graphical models come into play, since they graphically decode the conditional independence assumptions made. Hence, a sparser graph corresponds to more conditional independence assumptions. [1, ch. 9.1]

The made independence assumptions allow us to learn, store and later recombine the smaller (local) factors in which our (global) joint distribution factorizes into. At this point the major problem with inference becomes visible: To answer global queries/inference tasks it is necessary to recombine the prearranged separated information, which directly leads us back to the exponential blow-up we came from. We have to find a way to answer the queries avoiding this exponential blow-up of calculations to which the derivation of bigger factors lead. Computational complexity theory uses the notion of *NP-hard* for problems like that, which (probably) require exponential time to be calculated in worst case scenarios [1, ch. 9.1].

Summary: Firstly we need sparse graphical models for our variables, meaning as many conditional independence assumptions as possible, to factorize our big joint distribution into smaller factors, which we can learn, store and later recombine. Note that conditional independence assumptions are necessary but sometimes arbitrary, which could lead to wrong inference. Secondly we need clever algorithms to recombine bigger factors to answer global queries. Note that in worst case scenarios even clever algorithms cannot

overcome the exponential blow-up to which big factors lead (problems are in worst case NP-hard).

4 Variable Elimination Algorithm

The **Variable Elimination** algorithm is applicable for Bayes Nets and Markov Nets, as well as for CPD and MAP queries. The key idea is to sum (argmax) out one variable after another, which affects only the fraction of factors containing the variable to eliminate. Local structures should only depend on a small number of variables. By buffering earlier results and reusing them for later calculations, we can avoid generating them exponentially many times. This procedure can be classified as dynamic programming, since the local information results are buffered and passed to the next local structures, to propagate against the desired global conclusion. [1, ch. 9.1]

Note that the usage of factors allows to skip unnecessary normalization of intermediate results and to treat Bayesian and Markov Network Structures in the same manner. Remember that without normalization a factor(product) is in general not a distribution. But in the end every non-negative factor can be normalized to achieve a proper probability distribution.

4.1 Variable Elimination

We explore the Variable Elimination procedure with the graph used by Schwaferts' shown in figure 6. [3]

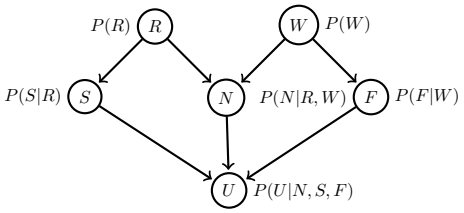


Figure 6: Schwaferts' Graph

Also we will be interested in the following CDP query:

$$P(U|F = f_1) \overset{convenience}{=} P(U|f_1) \tag{4.1}$$

We derive this CPD using the characteristics of Bayesian Networks to factorize the joint distribution and then applying factor reduction and marginalization:

$$\begin{aligned} P(U|f_1) &\propto \sum_{R,W,S,N} P(R)P(W)P(S|R)P(N|R,W)P(f_1|W)P(U|S,N,f_1) \\ &\propto \sum_{R,S,N} P(R)P(S|R)P(U|S,N,f_1) \sum_{W} \underbrace{P(W)P(N|R,W)P(f_1|W)} \end{aligned} \tag{4.2}$$

Given that the associative property holds in factor products, we can rearrange the factors and push in the summation over W. By building the factor product of the factors containing W and summing out W two intermediate factors get generated:

- **product W:** $\Psi_1(R, W, N) = P(W)P(N|R, W)P(f_1|W)$
- **sum out W:** $\tau_1(R, N) = \sum_W \Psi_1(R, W, N)$

At this point it is very important to understand, that summing out different variables, requires the generation of different factor products, which need a differing number of operations. We should proceed summing out the variable, which needs the least calculations, which we analyze using equation 1.6:

- **product R:** $\Psi_{2R}(R, S, N) = P(R)P(S|R)\tau_1(R, N)$ **mltp:** $2d_R d_S d_N$
- **product S:** $\Psi_{2S}(R, S, N, U) = P(S|R)P(U|S, N, f_1)$ **mltp:** $1d_R d_S d_N d_U$
- **product N:** $\Psi_{2N}(R, S, N, U) = \tau_1(R, N)P(U|S, N, f_1)$ **mltp:** $1d_R d_S d_N d_U$

We conclude that if the domain cardinality d_U of variable U is bigger then binary ($d_U > 2$), the factor product $2R$ takes the least multiplications. We continue in the same manner and after generating three more intermediate factors, we reach the final factor, which is proportional to the queried CPD:

- **sum out R:** $\tau_2(S, N) = \sum_R \Psi_2(R, S, N)$
- **product S,N:** $\Psi_3(S, N, U) = P(U|S, N, f_1)\tau_2(S, N)$
- **sum out S,N:** $\tau_3(U) = \sum_{S,N} \Psi_3(S, N, U) \propto P(U|f_1)$

4.2 Graphical Interpretation

In this section we want to see how each Variable Elimination step changes our graph. The general treatment via factors requires to look at undirected graphs, meaning we have to take the moral graph of our given Bayesian Network.

In figure 7 we see the moral graph on the left and the reduced graph on the right. Since the term $P(f_1|W)$ only depends on W, we add the factor to the node W, without building the product yet.

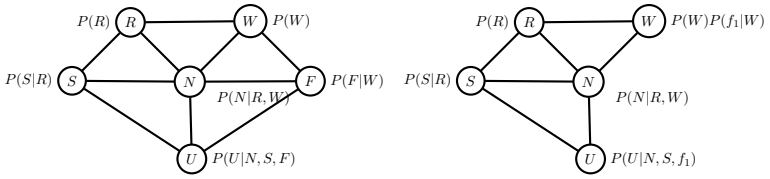


Figure 7: Moral and Reduced Graph.

In figure 8 we continue to visualize the first product step generating $\Psi_1(R, W, N) = P(W)P(N|R, W)P(f_1|W)$ and the first elimination step generating $\tau_1(R, N) = \sum_W \Psi_1(R, W, N)$.

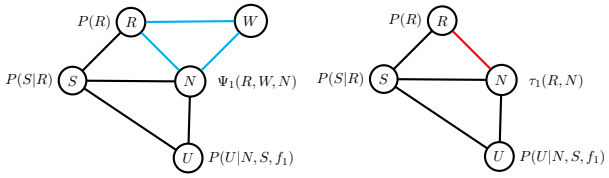


Figure 8: 1st product and elimination step.

Figure 9 shows the second product step generating $\Psi_2(R, S, N) = P(R)P(S|R)\tau_1(R, N)$, the second elimination step generating $\tau_2(S, N) = \sum_R \Psi_2(R, S, N)$ and finally the last product step generating $\Psi_3(S, N, U) = P(U|S, N, f_1)\tau_2(S, N)$ from which we can derive the queried CPD.

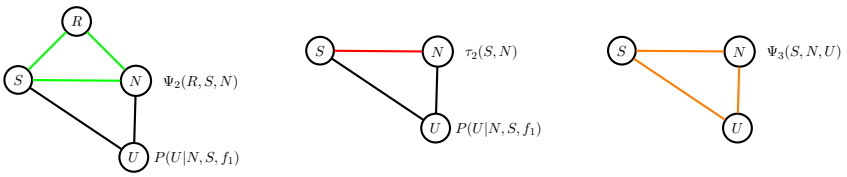


Figure 9: 2nd and last product and elimination step.

4.3 Elimination Ordering

Starting the elimination with variable N instead of variable W, we have to generate the following factor product:

$$\Phi_1(N, R, W, U, S, N) = P(N|R, W)P(U|S, N, f_1) \tag{4.3}$$

The scope of this factor product includes all non given variables. Like in the preceding chapter we visualize the factor product in figure 10 on the right site.

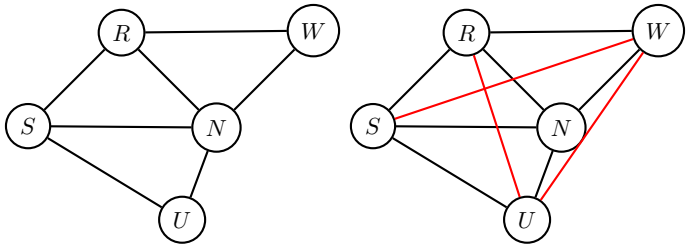


Figure 10: Original and Induced Graph by eliminating N first.

The figure shows that the generation of factor $\Phi_1(N, R, W, U, S, N)$ affords additional connections in the graph. The resulting graph is called the induced graph of the variable elimination process eliminating N first. The formal definition is:

Definition 6. Let $\oplus = \{\Phi_1, ..., \Phi_K\}$ be a set of factors over $\mathbf{X} = \{X_1, ..., X_n\}$, and α be an elimination ordering for some subset $\mathbf{W} \subset \mathbf{X}$. The induced graph $\mathcal{I}_{\oplus,\alpha}$ is an undirected graph over $\mathbf{X}$, where X_i and X_j are connected by en edge if they both appear in same intermediate factor Ψ generated by the VE algorithm using α as an elimination ordering. [1, ch. 9.4]

Moreover, the intermediate factors of a variable elimination process posses some important properties[1, ch. 9.4]:

Theorem 1. *Let $\mathcal{I}_{\oplus,\alpha}$ be an induced graph of a set of factors $\oplus$ and some elimination ordering α. Then:*

- *The scope of every generated factor Ψ is a clique in $\mathcal{I}_{\oplus,\alpha}$.*
- *Every maximal clique in $\mathcal{I}_{\oplus,\alpha}$ is the scope of some Ψ.*

The first statement comes due to the property that all variables in the scope of a factor must be a fully connected subgraph in the corresponding Markov Network. The generation of the factor product $\Phi_1(N, R, W, U, S, N)$ immediately required that all those

variables get fully connected in the induced graph. The second statement means that if a very large clique in the induced graph appears, then this clique will at some point be the scope of an intermediate factor during the variable elimination process. We already know that the large factors are the major cause for a blow-up of calculations. This leads us to the definition of the width of an induced graph [1, ch. 9.4]:

Definition 7. We define the **width** of a graph to be the number of nodes in the largest clique in the graph minus 1. The **induced width** $\omega_{\mathcal{K},\alpha}$ of an ordering α relative to a graph $\mathcal{K}$ is the width of the graph $\mathcal{I}_{\oplus,\alpha}$ induced by applying VE to $\mathcal{K}$ using elimination ordering α.[1, ch. 9.4]

A large induced width ω of an elimination ordering α, implies the generation of a large factor during variable elimination, and therefore indicates a blow-up of calculations.

4.4 Operations

In every variable elimination step we had to generate a factor product and eliminate a variable. Let Ψ_j be the generated factor and X_{i_j} the eliminated variable at elimination step j. We set K_j to the number of factors to generate the Ψ_j, N_j to the number of entries in Ψ_j and S_j to the number of variables in the scope of Ψ_j. Further we set $d_{i_j} = |Val(X_{i_j})|$ to be the domain cardinality of variable X_{i_j}. Using the equations 1.3 and 1.6 and assuming m elimination steps we conclude [1, ch. 9.1]:

$$N_{total} = \sum_{j=1}^{m} \left(\frac{d_{i_j} - 1}{d_{i_j}} N_j + (K_j - 1) N_j \right) \tag{4.4}$$

We continue to estimate an upper bound for this value. Setting $S = \max_j S_j$ to the largest number of variables in an intermediate factor (corresponds to width+1) and $d = \max_i d_i$ to the highest domain cardinality of all variables we derive $\forall j \in \{1, ..., m\}$: $N_j \leq d^S$ and conclude [1, ch. 9.1]:

$$N_{total} < d^S \sum_{j=1}^{m} K_j = d^S (K + m) < 2nd^S \rightarrow O(nd^S) \tag{4.5}$$

For the last estimation we assumed that the initial number of factors $K \leq n$ (always true for Bayesian Networks) and the number of elimination steps $m < n$.

The total number of operations increases exponential in d and S. Since S can be associated with the induced width $\omega_{\mathcal{K},\alpha}$ of an elimination ordering α, we associate the

computational complexity with an elimination ordering α. Finding good Eliminations Orderings is therefore very important for the inference performance, but also a *NP-complete* problem [1, ch. 9.4]. Nevertheless search algorithms using simple heuristics can find a good Elimination Ordering in practice. Those heuristics define a cost, which is evaluated for each node. The minimum node will be picked and eliminated and the next node will be searched. Common costs to reduce are[1, ch. 9.4]:

1. **Min-neighbors**: Cost is the number of neighbors.
2. **Min-weight**: Cost is the domain cardinality of its neighbors.
3. **Min-fill**: Cost is the number of edges that need to be added to the graph due to its elimination.
4. **Weighted-min-fill**: Cost is the sum of domain cardinalities of the edges that need to be added to the graph due to its elimination.

4.5 Variable Elimination Algorithm

As a summary we write the Variable Elimination Algorithm for CPD queries in algorithmic form [1, ch. 9.3]:

Algorithm 1 Variable Elimination for CPD

reduce initial factors by evidence $\mathbf{E} = \mathbf{e} \rightarrow \oplus := \{\Phi_1, ..., \Phi_m\}$ set of reduced factors
search elimination ordering using heuristics $\rightarrow \alpha$
for all $X_{\alpha(k)} \in \mathbf{W}$ **do**
    product $\Psi_k = \prod_{\Phi_i \in \oplus : X_{\alpha(k)} \in Scope[\Phi_i]} \Phi_i$
    sum $\tau_k = \sum_{X_{\alpha(k)}} \Psi_k$
    update $\tau_k := \Phi_{m+k} \in \oplus$
end for
renormalize the final factor $\tau_K(\mathbf{Y}) \rightarrow$ proper CDP $P(\mathbf{Y}|\mathbf{E} = \mathbf{e})$

5 Message Passing

5.1 Cluster Graphs

The fundamental structure used for Message Passing algorithms is the cluster graph. Beyond that certain properties need to hold to ease the message passing process. We start stating the basic definitions and explain them in advance with an example:

Definition 8. A **Cluster Graph** $\mathcal{T}$ is an undirected graph.
Each node is associated with a *cluster* $\mathbf{C_i} \subseteq \mathbf{X}$.
Each edge is associated with a *sepset* $\mathbf{S_{i,j}} \supseteq \mathbf{C_i} \cap \mathbf{C_j}$.
Family-preserving: Each factor $\Phi_k \in \oplus$ must be associated with a cluster $\mathbf{C_k}$, such that $Scope[\Phi_k] \supseteq \mathbf{C_k}$.[1, ch. 10.1]

Definition 9. Running Intersection Property (RIP): If any $X \in \mathbf{C_i}$ and $X \in \mathbf{C_j}$, then there exists an unique path from $\mathbf{C_i}$ to $\mathbf{C_j}$ containing X in every sepset and cluster on this path in the graph.[1, ch. 10.1]

Definition 10. Clique Tree: A Cluster Graph without loops, that satisfies the *RIP*.[1, ch. 10.1]

Theorem 2. *A Variable Elimination process induces a clique tree.[1, ch. 10.1]*

Fortunately, we can use our well-known example out of the variable elimination chapter, since the above theorem states that the VE process in chapter 4 induces a clique tree. Figure 11 shows how we can extract the clique tree out of the VE process example.

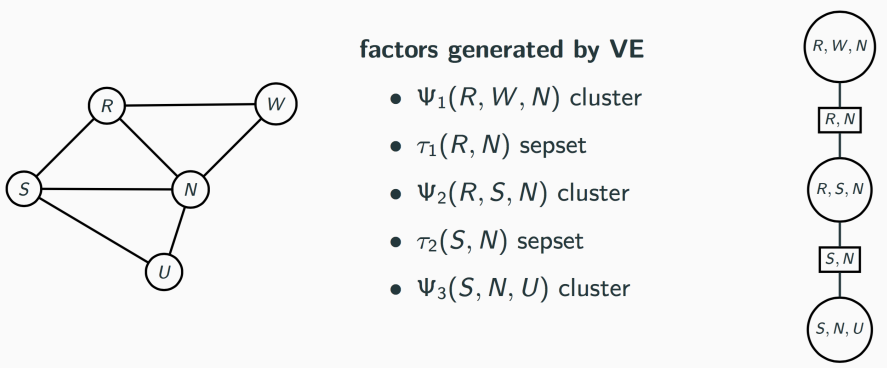


Figure 11: From Variable Elimination to Cluster Graphs.

The scope of every generated factor product builds a cluster and the scope of every marginal factor after elimination builds a sepset. Note that the scopes and not the

generated factors themselves are associated with a cluster! The other properties are easy to see: the graph is undirected, without loop and satisfying the RIP. An edge between the top and the bottom node, containing a sepset $\{N\}$, would violate the RIP, since there would exist two paths containing N between each cluster. It would also establish a loop, which violates the tree structure. Family Preservation is shown in figure 12, where each factor of the original factorized joint is associated with a cluster containing all the variables of this factor's scope. Do not confuse the associated factor products with the ones generated during VE!

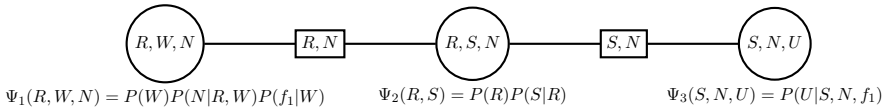


Figure 12: Family Preservation.

The products of the associated factors $\Psi_1(R, W, N)$, $\Psi_2(R, S)$ and $\Psi_3(S, N, U)$ are called initial potentials:

Definition 11. Recall that each factor $\Phi \in \oplus$ is assigned to some clique $\alpha(\Phi)$. We define the initial potential of $\mathbf{C_j}$ to be [1, ch. 10.2]:
$\Psi_j(\mathbf{C_j}) = \prod_{\Phi:\alpha(\Phi)=j} \Phi$.

5.2 Message Passing

The initial potentials know something about the variables in the cluster. To formalize how two neighboring initial potentials communicate their knowledge over the variables they share in the sepset we define messages:

Definition 12. The message from $\mathbf{C_i}$ to another cluster $\mathbf{C_j}$ is computed using the following sum-product message passing computation [1, ch. 10.2]:

$$\delta_{i \to j}(\mathbf{S_{i,j}}) = \sum_{\mathbf{C_i} - \mathbf{S_{i,j}}} \Psi_i \prod_{k \in (\mathcal{N}_i - j)} \delta_{k \to i} \tag{5.1}$$

Cluster C_i sends a message to cluster C_j concerning the variables in the sepset $S_{i,j}$, by combining the information of all incoming messages, except the one coming from C_j, and his initial potential. Since they can only talk about variables in their sepset all other variables are summed out. At this point it is very important to recognize that messages are defined through messages. Therefore, it is obvious that loops in the cluster graph structure, lead to loops in the message definitions, which means that they can only

converge to approximate values but never to exact ones. A clique tree on the other hand has endings where messages converge directly, since the sole incoming message comes from the cluster it is sending to and is therefore ignored.

In figure 13 we see all messages for our example clique tree. The messages starting from the tails $\delta_{1\to2} = \sum_W \Psi_1$ and $\delta_{3\to2} = \sum_U \Psi_3$ converge immediately, since they are independent from incoming messages. In the next message passing step the messages $\delta_{2\to1}$ and $\delta_{2\to3}$ converge also, since the only messages they depend on converged as well.

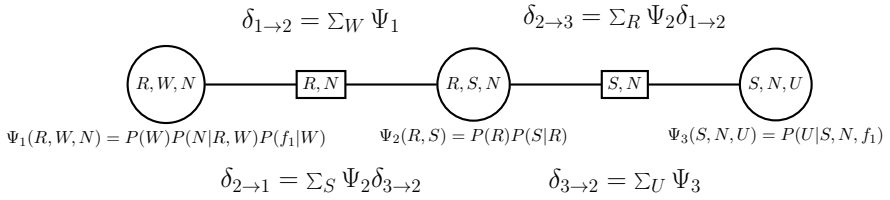


Figure 13: Message Passing.

We can generalize the **Message Passing for Clique Trees**:

1. pick an arbitrary root of the clique tree
2. starting from the tails pass all messages upwards to the root
3. continuing at the root to pass all messages back to the tails

After the upward and the downward pass all messages have converged. Note that this convergence depends on the tree like structure! Due to this special property of clique trees they are widely used, which lead to the particular name 'Clique Tree Algorithm'. [1, ch. 10.2]

The **Message Passing of Loopy Cluster Graphs** forms an iterative model, since a message passed through a loop will end up updating itself again and again:

1. initialize all messages to be 1
2. iterate over the message passing
3. stop when the approximate convergence is satisfying enough

In the next chapter, we will see that not only the messages get updated, but also the information a node has over the variables in its cluster, which is called belief. Approximate and iterative algorithms using Message Passing are also called 'Loopy Belief Propagation'. [1, ch. 10.2]

5.3 Beliefs

In contrast to the Variable Elimination Algorithm, Message Passing Algorithms don't answer queries directly, but try to build a basis for dynamic programming. The goal is to achieve intermediate results, which make it easier to answer many different types of queries. Those results include the hopefully converged messages and the so called beliefs, for which it can be shown that they correspond exactly (clique trees) or approximately (loopy) to the marginal distribution over the variables in the cluster.

Definition 13. A **belief** is defined as follows [1, ch. 10.2]:

$$\beta_i(\mathbf{C_i}) = \Psi_i \prod_{k \in \mathcal{N}_i} \delta_{k \to i} \tag{5.2}$$

We further define the term calibration, which presents the state of convergence [1, ch. 10.2]:

Definition 14. Two adjacent cliques $\mathbf{C_i}$ and $\mathbf{C_j}$ are said to be **calibrated** if they agree on their belief over their sepset:

$$\sum_{\mathbf{C_i} - \mathbf{S_{i,j}}} \beta_i(\mathbf{C_i}) = \sum_{\mathbf{C_j} - \mathbf{S_{i,j}}} \beta_j(\mathbf{C_j}) \tag{5.3}$$

Since changing messages cause changing beliefs the Message Passing process is also called Belief Propagation. Belief Propagation with clique trees is exact inference. After propagating the exactly calculated leaf messages thorough the tree the exact marginals are achieved everywhere. The cost for the upward and downward pass of messages in the Clique Tree Algorithm barely matches with the same computational performance necessary for two variable elimination algorithm runs. [1, ch. 10.2]

5.4 Dynamic Inference

The major advantage of Message Passing Algorithms is the dynamic programming concept. After achieving the approximate or exact marginals we can answer all kinds of queries. Thereby, we differentiate two cases:

Case 1: Evidence $E = e$ and Y are in the same cluster: Since E and Y are in the same cluster, we can just use the generated marginal distribution. We reduce the distribution by evidence E, sum out all needless variables and re-normalize.

Case 2: Evidence $E = e$ and Y are in different cliques. In this case we have to use **Incremental updates**! The new evidence changes the training potentials, namely the approximate or exact marginals. These changes in potentials lead to changes in the messages. All updated messages have to be passed to the node containing the query variables Y. In the end we achieve a new belief and therefore a new marginal distribution containing Y with respect to the given evidence $E = e$. Summing out all needless variables we can again find the queried CPD.

5.5 Difficulties

Example 2. As an example we take a n x n Markov Network grid which we want to break down into a clique tree with a simulated variable elimination procedure. Without using search methods to find a good elimination ordering, we see that every transformation into a clique tree requires to build a clique with at least n variables, to satisfy the running intersection property. A factor over n variables suffices to blowup the computational cost to exponential scale!

We conclude that in some cases we cannot find a clique tree without having to calculate large intractable factors.[1, ch. 10.4] That is when approximate methods like Loopy Belief Propagation come into play, since they don't require the generation of a tree structure. The problems for Loopy Belief Propagation are the mentioned uncertainty about convergence of the algorithm and the approximate results. Also, there are only heuristic methods to check whether convergence has not been achieved, but not if it has. Nevertheless, Loopy Belief Propagation is widely used for the so called turbo encoders, which try to decode and encode messages for better transition. The application of Loopy Belief Propagation was a big success in this area and shows that it can work pretty well in practice. [1, ch. 11.3]

5.6 Clique Tree Algorithm

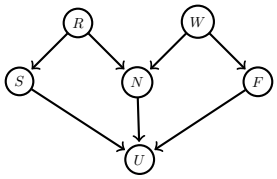
As a summary we write the Clique Tree Algorithm in algorithmic form [1, ch. 10.3]:

Algorithm 2 Clique Tree Algorithm

1: **simulate** VE and search for good $\alpha \rightarrow$ clique tree.
2: **product** initial potentials: $\Psi_i(\mathbf{C_i}) = \prod_{k:\alpha(k)=i} \Phi_k$.
3: **update** messages: $\delta_{i\rightarrow j}(\mathbf{S_{i,j}}) = \sum_{\mathbf{C_i}-\mathbf{S_{i,j}}} \Psi_i \prod_{k\in(\mathcal{N}_i-j)} \delta_{k\rightarrow i}$.
4: **upward pass** all messages from leafs to root.
5: **downward pass** all messages from root to leafs.
6: **calculate** beliefs: $\beta_i(\mathbf{C_i}) = \Psi_i \prod_{k\in\mathcal{N}_i} \delta_{k\rightarrow i}$.
7: **re-normalize** beliefs to get marginal distributions.
8: **store** tree structure and marginal distributions.

6 Sampling

6.1 Simple Sampling with Bayesian Networks

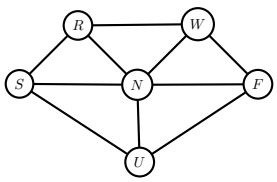


A Bayesian Network Graph $\mathcal{G}$ enables a simple sampling procedure. Since Bayesian Networks are directed we can start sampling the roots of the network with the marginal distributions. The parents of the subsequent descendants are now given, so we can sample those descendants as well using the CPDs. The so sampled descendants will be the parents for the next layer, and so on. Step by step we sample through those CPDs of which the parents have already been sampled. Notice that such a sampling procedure is not feasible with Markov Networks since we have no parental nodes as roots to start from! [1, ch. 12.1]

How can we use the generated samples to estimate e.g. $P(Y = y|\mathbf{E} = \mathbf{e})$? Just by throwing away all samples with $\mathbf{E} \neq \mathbf{e}$ and finally calculating the fraction of remaining samples with $Y = y$. This method is called **rejection sampling**. The computation of this method increases with the number of variables given in the evidence, since the accepted fraction gets smaller and smaller, it even decreases exponentially. [1, ch. 12.2]

6.2 Gibbs Sampling with Markov Networks

Gibbs Sampling is a special Markov Chain Monte Carlo method. Since we can't just sample from the roots to the leafs like in the Bayesian Network case, we start initializing random values for all variables. With those random values it is easy to calculate the full conditionals for every variable $P(X_i|\mathbf{X}_{-i})$. Note that except of one variable all other variables are given and therefore very few calculations are needed. We now use the calculated full CPDs to sample a next instantiation, and so on. Every sampled



instantiation updates the full CPDs and the updated full CPDs are used to generate a new sample. This loop can converge to the stationary distribution we are looking for and after convergence it generates samples from the real underlying distribution. But neither can we prove nor guarantee convergence. [1, ch. 12.3]

Other general problems of sampling are [1, ch. 12]:

- the lower the probabilities we want to estimate the more samples we need (Chernoff Bound)
- adjacent samples maybe are not i.i.d.

7 Outlook

The field of Probabilistic Graphical Models is still emerging. Further research tries:

- choosing more and better independence assumptions to achieve more sparse graphs
- finding better heuristics to find good elimination orderings and clique trees
- finding methods to improve and secure the convergence of approximate methods like Sampling and Loopy Belief Propagation

Also worth mentioning is the area of hybrid and continuous networks [4], which require and enable completely different inference methods, and the difficulty of inference with credal networks [5].

Appendix

Graph Properties

Definition 15. A graph $\mathcal{K}$ is a data structure containing **nodes** $\mathcal{X}$ representing variables $X_1, ..., X_n$ and **edges** $\mathcal{E}$ containing relationships between variables with types like:

- **undirected** $X_i - X_j$ (X_i and X_j are neighbors)
- **directed** $X_i \to X_j$ (X_i is parent and X_j is child)
- **connected** $X_i \rightleftharpoons X_j$ (unspecified).[1, ch. 2.1]

Definition 16. We say that variables $X_1, ..., X_k$ in the graph $\mathcal{K}(\mathcal{X}, \mathcal{E})$ form a:

- **path** if, $\forall i \in \{1, ..., k-1\}$ we have either $X_i \to X_{i+1}$ or $X_i - X_{i+1}$
- **directed path** if they form a path for which $\exists i : X_i \to X_{i+1}$
- **cycle** if they form a directed path for which $X_1 = X_k$
- **trail** if, $\forall i \in \{1, ..., k-1\}$ we have $X_i \rightleftharpoons X_{i+1}$
- **loop** if they form a trail for which $X_1 = X_k$.[1, ch. 2.1]

Definition 17. In a **directed graph** $\mathcal{K} = \mathcal{G}$ all edges are directed.
A **directed acyclic graph** DAG contains no **cycles**.
In an **undirected graph** $\mathcal{K} = \mathcal{H}$ all edges are undirected.
An **undirected singly graph (tree)** contains no **loops**.
The **undirected version** of graph $\mathcal{K} = (\mathcal{X}, \xi)$ is a graph $\mathcal{H} = (\mathcal{X}, \xi')$ where $\xi' = \{X - Y : X \rightleftharpoons Y \in \xi\}$.
A graph $\mathcal{K}$ is **chordal** if the longest minimal loop in its undirected version $\mathcal{H}'$ is a triangle. [1, ch. 2.1]

Definition 18. A subgraph $\mathcal{K}[\mathbf{X}](\mathbf{X}, \mathcal{E}')$ of graph $\mathcal{K}(\mathcal{X}, \mathcal{E})$ with $\mathbf{X} \subset \mathcal{X}$, where the subgraph edges are defined $\mathcal{E}' = \{X \rightleftharpoons Y \in \mathcal{E} | X, Y \in \mathbf{X}\}$. A subgraph over $\mathbf{X} \subset \mathcal{X}$ is complete if every two nodes in $\mathbf{X}$ are connected by some edge. The set $\mathbf{X}$ is often called a clique; we say that a clique $\mathbf{X}$ is maximal if for any superset of nodes $\mathbf{Y} \supset \mathbf{X}$, $\mathbf{Y}$ is not a clique. [1, ch. 2.1]

Factor Properties

Factor products are commutative and associative:

$$\Phi_1\Phi_2 = \Phi_2\Phi_1 \tag{7.1}$$

$$(\Phi_1\Phi_2)\Phi_3 = \Phi_1(\Phi_2\Phi_3) \tag{7.2}$$

Marginalization rules:

$$\sum_X \sum_Y \Phi = \sum_Y \sum_X \Phi \tag{7.3}$$

$$X \notin Scope[\Phi_1] \rightarrow \sum_X (\Phi_1\Phi_2) = \Phi_1 \sum_X \Phi_2 \tag{7.4}$$

References

[1] D. Koller and N. Friedman, *Probabilistic Graphical Models: Principles and Techniques*, MIT Press 2009.

[2] D. Koller, *Probabilistic Graphical Models*, Coursera Course 2013.

[3] P. Schwaferts, *Einfuehrung in die diskreten Bayes-und Kredal-Netze*, LMU München PGM Seminar 2015-2016.

[4] J. Ficek, *Introduction to continuous and hybrid Bayesian networks*, LMU München PGM Seminar 2015-2016.

[5] T. Steinherr, *Inferenz in Credalnetzen*, LMU München PGM Seminar 2015-2016.