entropy

MDPI

Article

Computation of Kullback–Leibler Divergence in Bayesian Networks

Serafín Moral, Andrés Cano and Manuel Gómez-Olmedo *

Computer Science and Artificial Intelligent Department, University of Granada, 18071 Granada, Spain; smc@decsai.ugr.es (S.M.); acu@decsai.ugr.es (A.C.)
* Correspondence: mgomez@decsai.ugr.es

Abstract: Kullback–Leibler divergence $KL(p,q)$ is the standard measure of error when we have a true probability distribution p which is approximate with probability distribution q. Its efficient computation is essential in many tasks, as in approximate computation or as a measure of error when learning a probability. In high dimensional probabilities, as the ones associated with Bayesian networks, a direct computation can be unfeasible. This paper considers the case of efficiently computing the Kullback–Leibler divergence of two probability distributions, each one of them coming from a different Bayesian network, which might have different structures. The paper is based on an auxiliary deletion algorithm to compute the necessary marginal distributions, but using a cache of operations with potentials in order to reuse past computations whenever they are necessary. The algorithms are tested with Bayesian networks from the *bnlearn* repository. Computer code in *Python* is provided taking as basis *pgmpy*, a library for working with probabilistic graphical models.

Keywords: probabilistic graphical models; learning algorithms; Kullback–Leibler divergence

Citation: Moral, S.; Cano, A.; Gómez-Olmedo, M. Computation of Kullback–Leibler Divergence in Bayesian Networks. *Entropy* **2021**, *23*, 1122. https://doi.org/10.3390/e23091122

Academic Editor: Raúl Alcaraz

Received: 29 July 2021
Accepted: 25 August 2021
Published: 28 August 2021

1. Introduction

When experimentally testing Bayesian network learning algorithms, in most of the cases, the performance is evaluated looking at structural differences between the graphs of the original Bayesian network and the learned one [1], as in the case of using the structural Hamming distance. This measure is used in recent contributions as [2–4]. A study and comparison of the different metrics used to measure the structural differences between two Bayesian networks can be found in [1].

However, in most cases the aim of learning a Bayesian network is to estimate a joint probability for the variables in the problem. In that situation the error of a learning procedure should be computed by measuring the difference between the probability associated with the learned network and the original joint probability distribution. Therefore, it can be useful to estimate a network that is less dense than the original one, but in which parameters can have a more accurate estimation. This is the case of the *Naive Bayes* classifier, which obtains very good results in classification problems, despite the fact that the structure is not the correct one. So, in this situation, structural graphical differences are not a good measure of performance.

The basic measure to determine the divergence between an estimated distribution and a true one is the so-called Kullback–Leibler divergence [5]. Some papers use this way of asserting the quality of a learning procedure as in [6–8]. A direct computation of the divergence is unfeasible if the number of variables is high. However, some basic decomposition properties [9] (Theorem 8.5) can be applied to reduce the cost of computation of the divergence. This is the basis of the procedure implemented in the Elvira system [10] which is the one used in [6]. Methods in [7,8] are also based on the same basic decomposition. Kullback–Leibler divergence is not only meaningful for measuring divergence between a learned network and a true one, but also for other tasks, as for example the approximation of a Bayesian network by a simpler one [11–13] by removing some of the existing links.

The aim of this work is to improve existing methods for computing Kullback–Leibler divergence in Bayesian networks and to provide a basic algorithm for this task using *Python* and integrated into the *pgmpy* [14] environment. The algorithm implemented in the *Elvira system* [10] is based on carrying out a number of propagation computations in the original true network. The hypothesis underlying our approach is that there are a lot of computations that are repeated in these propagation algorithms, so what it is done is to determine which are the operations with potentials that are repeated and then storing the results in a cache of operations in order to allow reuse them. The experimental work will show that this is an effective method to improve the efficiency of algorithms, especially in large networks.

The paper is organized as follows: Section 2 is devoted to set the basic framework and to present fundamental results for the Kullback–Leibler divergence computation; Section 3 describes the method implemented in the Elvira system for computing Kullback–Leibler divergence; Section 4 is devoted to describing our proposal based on the cache of operations with potentials; Section 5 contains the experimental setting and the obtained results; finally the conclusions are shown in Section 6.

2. Kullback–Leibler Divergence

Let N be a Bayesian network defined on a set of variables $\mathbf{X} = \{X_1 \dots X_n\}$. The family of a variable X_i in $\mathbf{X}$ is termed $f(X_i) = \{X_i\} \cup pa(X_i)$, where $pa(X_i)$ is the set of parents of X_i in the directed acyclic graph (DAG) defined by N. $\mathbf{F} = \{f(X_1) \dots f(X_n)\}$ denotes the complete set of families, one for each one of the variables. Sometimes simplified notations for families and parent sets will be used: f_i (for $f(X_i)$) and pa_i (for $pa(X_i)$), respectively. As a running example, assume a network with three variables, X_1, X_2, X_3 and the following structure: $X_1 \to X_2 \to X_3$ (see right part of Figure 1). Then the set of families for this network is given by $\{f_1, f_2, f_3\}$, where $f_1 = \{X_1\}, f_2 = \{X_2, X_1\}, f_3 = \{X_3, X_2\}$.

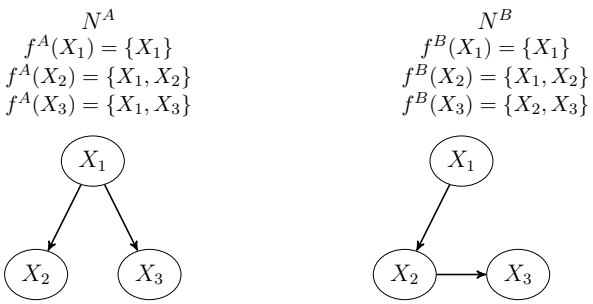

Figure 1. Bayesian networks to compare.

A configuration or assignment of values to a set of variables $\mathbf{X}$, $\{X_1 = x_1 \dots X_n = x_n\}$, can be abbreviated with $(x_1 \dots x_n)$ and is denoted as $\mathbf{x}$. If the set of possible values for each variable in the previous example is $\{0, 1\}$, then a configuration can be $\mathbf{x} = (0, 0, 1)$, representing the assignment $\{X_1 = 0, X_2 = 0, X_3 = 1\}$.

A partial configuration involving a subset of variables $\mathbf{Y} \subseteq \mathbf{X}$ is denoted as $\mathbf{y}$. If the set of variables is f_i or pa_i, then the partial configuration will be denoted by $\mathbf{x}_{f_i}$ or $\mathbf{x}_{pa_i}$, respectively. In our example, if $f_2 = \{X_2, X_1\}$ an example of partial configuration about these variables will be $\mathbf{x}_{f_2} = (0, 0)$.

The set of configurations for variables $\mathbf{Y}$ is denoted by $\Omega_{\mathbf{Y}}$. If $\mathbf{x}$ is an assignment and $\mathbf{Y} \subseteq \mathbf{X}$, then the configuration $\mathbf{y}$ obtained by deleting the values of the variables in $\mathbf{X} \setminus \mathbf{Y}$ is denoted by $\mathbf{x}^{\downarrow \mathbf{Y}}$. If $\mathbf{x}_{f_2} = (0, 0)$ is a partial configuration about variables $\{X_2, X_1\}$ and we consider $\mathbf{Y} = \{X_2\}$, then $\mathbf{x}_{f_2}^{\downarrow \mathbf{Y}}$ is the configuration obtained by removing the value of X_1, i.e., (0).

If $\mathbf{w}$ and $\mathbf{z}$ are configurations for $\mathbf{W} \subseteq \mathbf{X}$ and $\mathbf{Z} \subseteq \mathbf{X}$ respectively, and $\mathbf{W} \cap \mathbf{Z} = \emptyset$, then $(\mathbf{w}, \mathbf{z})$ is a configuration for $\mathbf{W} \cup \mathbf{Z}$, and will be called the composition of $\mathbf{w}$ and $\mathbf{z}$. For example, if $\mathbf{w}$ is the configuration (0) about variable X_1 and $\mathbf{y}$ is the configuration

$(0,1)$ defined on X_2, X_3, then its composition will be the configuration $(0,0,1)$ for variables $\{X_1, X_2, X_3\}$.

The conditional probability distribution for X_i given its parents will be denoted as ϕ_i which is a potential defined on the set of variables $f(X_i)$. In general, a potential ϕ for variables $\mathbf{Y} \subseteq \mathbf{X}$ is a mapping defined on $\Omega_{\mathbf{Y}}$ into the set of real numbers: $\phi : \Omega_{\mathbf{Y}} \to \mathbb{R}$. The set of variables of potential ϕ will be denoted as $v(\phi)$. If $\mathbf{\Phi}$ is a set of potentials, $v(\mathbf{\Phi})$ will denote $\bigcup_{\phi \in \mathbf{\Phi}} v(\phi)$.

In our example, there are three potentials and $\mathbf{\Phi} = \{\phi_1(X_1), \phi_2(X_2, X_1), \phi_3(X_3, X_2)\}$ (that is, a probability distribution about X_1, and two conditional probability distributions: one for X_2 given X_1 and the other for X_3 given X_2, respectively).

There are three basic operations that can be performed on potentials:

- Multiplication. If ϕ, ϕ' are potentials, then their multiplication is the potential $\phi \cdot \phi'$, with set of variables $v(\phi \cdot \phi') = v(\phi) \cup v(\phi')$ and obtained by pointwise multiplication:

$$\phi \cdot \phi'(\mathbf{y}) = \phi(\mathbf{y}^{\downarrow v(\phi)}) \cdot \phi'(\mathbf{y}^{\downarrow v(\phi')}).$$

  In our example, the combination of ϕ_2 and ϕ_3 will be the potential $\phi_2 \cdot \phi_3$ defined on $\{X_1, X_2, X_3\}$ and given by $\phi_2 \cdot \phi_3(x_1, x_2, x_3) = \phi_2(x_2, x_1) \cdot \phi_3(x_3, x_2)$.
- Marginalization. If ϕ is a potential defined for variables $\mathbf{Y}$ and $\mathbf{Z} \subseteq \mathbf{Y}$, then the marginalization of ϕ on $\mathbf{Z}$ is denoted by $\phi^{\downarrow \mathbf{Z}}$ and it is obtained by summing in the variables in $\mathbf{Y} \setminus \mathbf{Z}$:

$$\phi^{\downarrow \mathbf{Z}}(\mathbf{z}) = \sum_{\mathbf{y}^{\downarrow \mathbf{Z}} = \mathbf{z}} \phi(\mathbf{y}).$$

  When $\mathbf{Z}$ is equal to $\mathbf{Y}$ minus a variable W, then $\phi^{\downarrow \mathbf{Z}}$ will be called the result of removing W in ϕ and also denoted as ϕ^{-W}. In the example, a marginalization of ϕ_3 is obtained by removing X_3 producing $\phi_3^{-X_3}$ defined on X_2 and given by $\phi_3^{-X_3}(x_2) = \phi_3(0, x_2) + \phi_3(1, x_2)$. If $\phi_3(x_3, x_2)$ represents the conditional probability of $X_3 = x_3$ given $X_2 = x_2$, then it can be obtained that $\phi_3^{-X_3}(x_2)$ is always equal to 1 ($\forall x_2 \in v(\phi_3)$).
- Selection. If ϕ is a potential defined for variables $\mathbf{Y}$ and $\mathbf{z}$ is a configuration for variables $\mathbf{Z}$, then the selection of ϕ for this configuration $\mathbf{z}$ is the potential $\phi_{\mathbf{Z}=\mathbf{z}}$ defined on variables $\mathbf{W} = \mathbf{Y} \setminus \mathbf{Z}$ and given by

$$\phi_{\mathbf{Z}=\mathbf{z}}(\mathbf{w}) = \phi(\mathbf{w}, \mathbf{z}^{\downarrow \mathbf{Y}}).$$

  In this expression $(\mathbf{w}, \mathbf{z}^{\downarrow \mathbf{Y}})$ is the composition of configurations $\mathbf{w}$ and $\mathbf{z}^{\downarrow \mathbf{Y}}$ which is a configuration for variables $v(\phi)$. Going back to the example, assume that we want to perform the selection of ϕ_3 to configuration $\mathbf{z} = (0,1)$ for variables $\{X_1, X_2\}$, then $\phi_{3\mathbf{Z}=\mathbf{z}}$ will be a potential defined for variables $\{X_2, X_3\} \setminus \{X_1, X_2\} = \{X_3\}$ given by $\phi_{3\mathbf{Z}=\mathbf{z}}(x_3) = \phi_3(x_3, 1)$, as we are reducing $\phi_3(X_3, X_2)$ to a configuration $\mathbf{z}$ in which $X_2 = 1$.

The family of all the conditional distributions is denoted as $\mathbf{\Phi} = \{\phi_1, \ldots, \phi_n\}$. It is well known that given N the joint probability distribution of the variables in N, p, is a potential that decomposes as the product of the potentials included in $\mathbf{\Phi}$:

$$p = \prod_{\phi_i \in \mathbf{\Phi}} \phi_i \tag{1}$$

Considering the example, $\mathbf{\Phi} = \{\phi_1, \phi_2, \phi_3\}$ and $p = \phi_1 \cdot \phi_2 \cdot \phi_3$. The marginal distribution of p for a set of variables $\mathbf{Y} \subseteq \mathbf{X}$ is equal to $p^{\downarrow \mathbf{Y}}$. When $\mathbf{Y}$ contains only one variable X_i, then to simplify the notation, $p^{\downarrow \mathbf{Y}}$ will be denoted as p_i. Sometimes it will be needed to make reference to the Bayesian network containing a potential or family. In these cases we will use a superscript. For example, $f^A(X_i)$ and $pa^A(X_i)$ refer to the family and parents set of X_i in a Bayesian network N^A respectively.

The aim of this paper is to compute the Kullback–Leibler divergence (termed KL) between the joint probability distributions, p^A and p^B, of two different Bayesian networks N^A and N^B defined on the same set of variables $\mathbf{X}$ but possibly having different structures. This divergence, denoted as $KL(N^A, N^B)$ can be computed considering the probabilities for each configuration $\mathbf{x}$ in both distributions as follows:

$$KL(N^A, N^B) = \sum_{\mathbf{x}} p^A(\mathbf{x}) \log\left(\frac{p^A(\mathbf{x})}{p^B(\mathbf{x})}\right) \tag{2}$$

However, the computation of the joint probability distribution may be unfeasible for complex models as the number of configurations $\mathbf{x}$ is exponential in the number of variables. If p, q are probability distributions on $\mathbf{X}$ then the expected *log likelihood* (LL) of q with respect to p is:

$$LL(p, q) = \sum_{\mathbf{x}} p(\mathbf{x}) \log(q(\mathbf{x})),$$

then, from Equation (2) it is immediate that:

$$\begin{gathered} KL(N^A, N^B) = \sum_{\mathbf{x}} p^A(\mathbf{x}) \log(p^A(\mathbf{x})) - \sum_{\mathbf{x}} p^A(\mathbf{x}) \log(p^B(\mathbf{x})) = \\ LL(p^A, p^A) - LL(p^A, p^B) = LL(N^A, N^A) - LL(N^A, N^B) \end{gathered} \tag{3}$$

The probability distribution p^B can be decomposed as well as considered in Equation (1). Therefore, the term $LL(N^A, N^B)$ in Equation (3) can be obtained as follows considering the families of variables in N^B and their corresponding configurations, $\mathbf{x}^{\downarrow f_i^B}$:

$$\begin{gathered} LL(N^A, N^B) = \sum_{\mathbf{x}} p^A(\mathbf{x}) \log(p^B(\mathbf{x})) = \sum_{\mathbf{x}} p^A(\mathbf{x}) \log(\prod_{X_i \in \mathbf{X}} \phi_i^B(\mathbf{x}^{\downarrow f_i^B})) = \\ \sum_{\mathbf{x}} p^A(\mathbf{x}) \sum_{X_i \in \mathbf{X}} \log(\phi_i^B(\mathbf{x}^{\downarrow f_i^B})) \end{gathered} \tag{4}$$

Interchanging additions and reorganizing the terms in Equation (4):

$$\begin{gathered} LL(N^A, N^B) = \sum_{X_i \in \mathbf{X}} \sum_{\mathbf{x}} p^A(\mathbf{x}) \log(\phi_i^B(\mathbf{x}^{\downarrow f_i^B})) = \\ \sum_{X_i \in \mathbf{X}} \sum_{\mathbf{x}_{f_i^B}} \log(\phi_i^B(\mathbf{x}_{f_i^B})) \left(\sum_{\mathbf{x}^{\downarrow f_i^B} = \mathbf{x}_{f_i^B}} p^A(\mathbf{x}) \right) = \\ \sum_{X_i \in \mathbf{X}} \sum_{\mathbf{x}_{f_i^B}} \log(\phi_i^B(\mathbf{x}_{f_i^B}))(p^A)^{\downarrow f_i^B}(\mathbf{x}_{f_i^B}) \end{gathered} \tag{5}$$

Equation (5) implies a decomposition of the term $LL(N^A, N^B)$ and, as a consequence of $KL(N^A, N^B)$ computation as well. Observe that $\phi_i^B(\mathbf{x}_{f_i^B})$ is the value of the potential ϕ_i^B for a configuration $\mathbf{x}_{f_i^B}$ and can be obtained directly from the potential ϕ_i^B of the Bayesian network N^B. The main difficulty in Equation (5) consists of the computation of $(p^A)^{\downarrow f_i^B}(\mathbf{x}_{f_i^B})$ values, as it is necessary to compute the marginal probability distribution for variables in f_i^B, the family of X_i in Bayesian network N^B, but using the joint probability distribution p^A associated with the Bayesian network N^A.

3. Computation with Propagation Algorithms

In this section we introduce the category of inference algorithms based on deletion of variables and then we show how these algorithms can be applied to compute the Kullback–Leibler divergence using Equation (5).

3.1. Variable Elimination Algorithms

To compute $(p^A)^{\downarrow f_i^B}$ we consider $\mathbf{\Phi}^A$ the set of potentials associated to network N^A: the multiplication of all the potentials in $\mathbf{\Phi}^A$ is equal to p^A. Deletion algorithms [15,16], can be applied to $\mathbf{\Phi}^A$ to determine the required marginalizations. The basic step of these algorithms is the deletion of a variable from a set $\mathbf{\Phi}^A$:

- **Variable Deletion.** If $\mathbf{\Phi}$ is a set of potentials, the deletion of X_i consists of the following operations:
  - Compute $\mathbf{\Phi}_i = \{\phi : X_i \in v(\phi)\}$, i.e., the set of potentials containing variable X_i.
  - Compute $\phi^{-i} = \left(\prod_{\phi \in \mathbf{\Phi_i}} \phi\right)^{-X_i}$, i.e., combine all the potentials in $\mathbf{\Phi}_i$ and remove variable X_i by marginalization.
  - Update $\mathbf{\Phi} \leftarrow (\mathbf{\Phi} \setminus \mathbf{\Phi}_i) \cup \{\phi^{-i}\}$, i.e., remove from $\mathbf{\Phi}$ the potentials containing X_i and add the new potential ϕ^{-i} which does not contain X_i.

The main property of the deletion step is the following: starting with $\prod_{\phi \in \mathbf{\Phi}} \phi = q$, then after the deletion of X_i from $\mathbf{\Phi}$, $\prod_{\phi \in \mathbf{\Phi}} \phi = q^{-X_i}$. It is easy to see that the deletion of a variable X_i can be computed just operating with the elements of $\mathbf{\Phi}$ defined on X_i.

If $\mathbf{\Phi}$ is the initial set of potentials of a network N, then $p = \prod_{\phi \in \mathbf{\Phi}} \phi$. In order to compute the marginalization of p on a set of variables $\mathbf{Y} \subseteq \mathbf{X}$, the deletion procedure should be repeated for each variable X_i in $\mathbf{X} \setminus \mathbf{Y}$. If the marginal probability distribution for variable X_k is to be calculated, any variable in $\mathbf{X}$ different from X_k should be deleted. The order of variable deletion is not meaningful for the final result, but the efficiency may depend on it.

When there are observed variables, $\mathbf{Z} = \mathbf{z}$, then a previous step of selection should be carried out: any potential $\phi \in \mathbf{\Phi}$ is transformed into $\phi_{\mathbf{Z}=\mathbf{z}}$. This step will be called *evidence restriction*. After it q, the product of the potentials in $\mathbf{\Phi}$ is defined for variables in $\mathbf{Y} = \mathbf{X} \setminus \mathbf{Z}$ and its value is $q(\mathbf{y}) = p(\mathbf{y}, \mathbf{z})$, i.e., the joint probability of obtaining this value and the observations. If a deletion of variables in $\mathbf{W}$ is carried out, then the product of the potentials in $\mathbf{\Phi}$ is the potential defined for variables $\mathbf{Y} = (\mathbf{X} \setminus \mathbf{Z}) \setminus \mathbf{W}$, and satisfies $q(\mathbf{y}) = p^{\downarrow \mathbf{Y} \cup \mathbf{Z}}(\mathbf{y}, \mathbf{z})$.

When we have observations and we want to compute the marginal on a variable X_i, it is well known that not all the initial potentials in $\mathbf{\Phi}$ are relevant. A previous pruning step can be done using the *Bayes-ball* algorithm [17] in order to remove the irrelevant potentials from $\mathbf{\Phi}$ before restricting to the observations and carrying out the deletion of variables.

3.2. Computation of Kullback–Leibler Divergence Using Deletion Algorithms

Our first alternative to compute $LL(N^A, N^B)$ is based on using a simple deletion algorithm to compute the values $(p^A)^{\downarrow f_i^B}(\mathbf{x}_{f_i^B})$ in Equation (5). The basic steps are:

- Given a specific variable X_i, we have that $f_i^B = \{X_i\} \cup pa_i^B$. Then for each possible configuration $\mathbf{x}_{pa_i^B}$ of the parent variables, we include the observation $pa_i^B = \mathbf{x}_{pa_i^B}$ and we apply a selection operation to the list of potentials associated with Bayesian network N^A by means of evidence restriction. We also apply a pruning of irrelevant variables using the *Bayes-ball* algorithm.
- Then all the variables are deleted except the target variable X_i. The potentials in $\mathbf{\Phi}$ will be all defined for variable X_i and their product will be a potential q defined for variable X_i such that $q(x_i) = (p^A)^{\downarrow f_i^B}(x_i, \mathbf{x}_{pa_i^B})$.
- The deletion algorithm is repeated for each variable X_i and each configuration of the parent variables $\mathbf{x}_{pa_i^B}$ in Bayesian network N^B. So, the number of executions of the

propagation algorithm in Bayesian network N^A is equal to $\sum_{i=1}^{n} \prod_{X_j \in pa^B(X_i)} n_j$, where n_j is the number of possible values of X_j. This is immediate taking into account that $\prod_{X_j \in pa^B(X_i)} n_j$ is the number of possible configurations $\mathbf{x}_{pa^B(X_i)}$ of variables in $pa^B(X_i)$.

Though this method can take advantage of propagation algorithms to compute marginals in a Bayesian network, and it avoids a brute force computation associated with the use of Equation (2), it is quite time consuming when the structure of the involved Bayesian network is complex.

Algorithm 1 details the basic steps of the initial proposal for computing the Kullback–Leibler divergence. This algorithm is the one used in [8,10]. Observe that this algorithm computes $LL(N^A, N^B)$. It allows the computation of the KL divergence by using Equation (3).

Algorithm 1 Computation of LL using an evidence propagation algorithm

```
1:  function LL(N^A, N^B)
2:      sum ← 0.0                                   ▷ sets initial value to sum
3:      for each X_i in N^B do
4:          for each x_{pa_i^B} do                  ▷ configuration of X_i parents
5:              Let Φ' the set of relevant potentials from Φ   ▷ Applying Bayes-ball
6:              Restrict the potentials in Φ' to evidence pa_i^B = x_{pa_i^B}
7:              Delete in Φ' all the variables in v(Φ) \ {X_i}
8:              Let q the product of all the potential in Φ'
9:              for each x_i in Ω_{X_i} do
10:                 sum ← sum + q(x_i) log(φ_i^B(x_i, x_{pa_i^B}))
11:             end for
12:         end for
13:     end for
14:     return sum
15: end function
```

As an example, let us suppose we wish to compute the KL divergence between two Bayesian networks N^A and N^B defined on $\mathbf{X} = \{X_1, X_2, X_3\}$ (see Figure 1). Let us assume N^A is the reference model. The families of variables in both models are presented in Figure 1. We have to compute $LL(N^A, N^A)$ and $LL(N^A, N^B)$. To compute $LL(N^A, N^B)$ Algorithm 1 is applied. Initially, $\mathbf{\Phi} = \{\phi_1^A, \phi_2^A, \phi_3^A\}$ where ϕ_i^A is defined for variables $f^A(X_i)$. The algorithm works as follows:

- The parent set for X_1 is empty. The set of relevant potentials in network N^A to compute the marginal for X_1 is given by $\mathbf{\Phi}' = \{\phi_1^A\}$, which is the desired marginal q.
- The parents set for X_2 in N^B is $\{X_1\}$. So, for each value $X_1 = x_1$ we have to introduce this evidence in network N^A and compute the marginal on X_2. The set relevant potentials is $\mathbf{\Phi}' = \{\phi_1^A, \phi_2^A\}$. These potentials are reduced by selection on configuration $X_1 = x_1$. If we call ϕ_4, ϕ_5 the results of reducing ϕ_1^A, ϕ_2^A, respectively, then ϕ_4 is a potential defined for the empty set of variables and determined by its value $\phi_4()$ for the empty configuration. ϕ_5 is a potential defined for variable X_2. The desired marginal q is the multiplication of these potentials: $q(x_2) = \phi_4() \cdot \phi_5(x_2)$.
- The parents set for X_3 in N^B is $\{X_2\}$. So, for each value $X_2 = x_2$ we have to introduce this evidence in network N^A and compute the marginal on X_3. In this case, all the potentials are relevant $\mathbf{\Phi}' = \{\phi_1^A, \phi_2^A, \phi_3^A\}$. The first step introduces the evidence $X_2 = x_2$ in all the potentials containing this variable. Only ϕ_2^A contains X_2; therefore the selection $\phi_2^A{}_{X_2=x_2}$ is a potential defined on variable X_1 which we will denote as ϕ_6. So, after that $\mathbf{\Phi}' = \{\phi_1^A, , \phi_3^A, \phi_6\}$. To compute the marginal on X_3, we have to delete variable X_1. As all the potentials in $\mathbf{\Phi}'$ contains this variable they must be combined for removing X_1 afterwards, i.e., computing $(\phi_1^A \cdot \phi_3^A \cdot \phi_6)^{-X_1}$. After this operation, this will be the only potential in $\mathbf{\Phi}'$ and it is the desired marginal q.

4. Inference with Operations Cache

The approach proposed in this paper is based on the following fact: the computation of KL divergence using Equations (3) and (5) requires us to obtain the following families of marginal distributions:

- $(p^A)^{\downarrow f_i^B}$, for each X_i in N^B, for computing $LL(N^A, N^B)$
- $(p^A)^{\downarrow f_i^A}$, for each X_i in N^A, for obtaining $LL(N^A, N^A)$

We have designed a procedure to compute each one of the required marginals $(p^A)^{\downarrow \mathbf{Y}}$ for each $\mathbf{Y} \in \{f_i^A : X_i \in N^A\} \cup \{f_i^B : X_i \in N^B\}$. Marginals are computed by deleting the variables not in $\mathbf{Y}$. The procedure uses a cache of computations which can be reused in the different marginalizations in order to avoid repeated computations.

We have implemented a general procedure to calculate the marginal for a family $\mathcal{Y}$ of subsets $\mathbf{Y}$ of $\mathbf{X}$ in a Bayesian network N. In our case the family $\mathcal{Y}$ is $(p^A)^{\downarrow \mathbf{Y}}$ for each $\mathbf{Y} \in \{f_i^A : X_i \in N^A\} \cup \{f_i^B : X_i \in N^B\}$ and the Bayesian network is N^A. A previous step consists of determining the relevant potentials for computing the marginal on a subset $\mathbf{Y}$, as not all the initial potentials are necessary. If $\mathbf{\Phi}$ is the list of potentials, then a conditional probability potential ϕ_i for variable X_i is relevant for $\mathbf{Y}$ when X_i is an ascendant for some of the variables in $\mathbf{Y}$. This is a consequence of known relevance properties in Bayesian networks [17]. Let us call $\mathbf{\Phi_Y}$ the family of relevant potentials for subset $\mathbf{Y}$.

Our algorithm assumes that the subsets in $\mathcal{Y}$ are numbered from 1 to K: $\{\mathbf{Y}_1, \ldots, \mathbf{Y}_K\}$. The algorithm first carries out the deletion algorithm symbolically, without actually doing numerical computations, in order to determine which of them can be reused. A symbolic combination of ϕ and ϕ' consists of determining a potential $\phi \cdot \phi'$ defined for variables $v(\phi) \cup v(\phi')$ but without computing its numerical values (only the scope of the resulting potential is actually computed). This procedure is analogously done in the case of marginalization.

In fact, two repositories are employed: one for potentials (R_Φ) and another for operations (R_O). The entry for each potential in R_Φ contains a value acting as its identifier (id); the potential itself; the identifier of the last operation for which this potential was required (this is denoted as potential $time$). Initially, R_Φ contains the potentials in Φ assigning $time = 0$ to all of them. When a potential is no longer required, then it is removed from R_Φ in order to alleviate memory space requirements. The potentials representing the required marginals (the results of the queries) are set with $time = -1$ in order to avoid their deletion.

The repository R_O contains an entry for each operation (combination or marginalization) with potentials performed during the running of the algorithm in order to compute the required marginals. This allows that if an operation is needed in the future, its result can be retrieved from R_O preventing repeated computations. Initially R_O will be empty. At the end of the analysis, it will include the description of the elementary operations carried out throughout the evaluation of all the queries. Two kinds of operations will be stored in R_O:

- combination of two potentials ϕ_1 and ϕ_2 producing a new one as result, ϕ_r.
- marginalization of a potential ϕ_1, in order to sum-out a variable and obtaining ϕ_r as result.

The operation description will be stored as registers $(id, type, \phi_1, \phi_2, \phi_r)$ with the following information:

- A unique identifier for the operation (id; an integer).
- The type of operation ($type$): marginalization or combination.
- Identifiers of the potentials involved as operands and result (identifiers allow to retrieve potentials from R_Φ). If the operation is a marginalization, then ϕ_2 will identify the index of the variable to remove.

The computation of a marginal for a set $\mathbf{Y}$ will also require a deletion order of variables in some cases. This order is always obtained with a fixed triangulation heuristic $min - weight$ [18]. However, the procedure described here does not depend on this heuristic and any one of them could be used.

Algorithm 2 depicts the basic structure of the procedure. The result is LR, an ordered list of K potentials containing the required marginals for $\mathbf{Y}_1, \ldots, \mathbf{Y}_K$. The algorithm is divided into two main parts.

In the first part (lines 2–26), the operations are planned (using symbolic propagation) and detecting repeated operations. It is assumed that there are two basic functions SCOMBINE(ϕ_1, ϕ_2) and SMARGINALIZE(ϕ, i), representing the symbolic operations: SCOMBINE(ϕ_1, ϕ_2) will create a new potential ϕ_r with $v(\phi_r) = v(\phi_1) \cup v(\phi_2)$ and SMARGINALIZE(ϕ, i) producing another potential ϕ_r with $v(\phi_r) = v(\phi) \setminus \{X_i\}$.

We will also consider that there are two conditional versions of these operations: if the operation already exists, only the *time* is updated, and if it does not exist it is symbolically carried out and added to the repository of operations. The conditional combination will be CONDSCOMBINE(ϕ_1, ϕ_2, t) and the conditional marginalization will be CONDSMARGINALIZE(ϕ, i) and are depicted in Algorithms 3 and 4, respectively. It is assumed that both repositories are global variables for all the procedures. The potentials representing the required marginals are never deleted. For that, a time equal to -1 is assigned: if $time = -1$, then the potential should not be removed and then this time is never updated. We will assume the function UPDATETIME(ϕ, t) which does nothing if the time of ϕ is equal to -1, and updates the time of ϕ to t otherwise in repository R_Φ.

Observe that the first part of Algorithm 2 (lines 2–26) just determines the necessary operations for the deletion algorithm for for all the marginals, while the second part (lines 27–32) carries out the numerical computations in the order that was established in the first part. After each operation, the potentials that are no longer necessary are removed from R_Φ and their memory is deallocated. We will assume a function DELETEIF(ϕ, t) doing this (remove from R_Φ if *time* of ϕ is equal to t).

As mentioned above, the analysis of the operation sequence will be carried out using symbolic operations and taking into account the scopes of potentials but without numerical computations. This allows an efficient analysis. The result of the analysis will be used as an operation planning for the posterior numerical computation.

Assume the same example considered in previous sections for the networks in Figure 1. The marginals to compute on N^A (as reference model) will correspond to families $f^A(X_1) = \{X_1\}$, $f^A(X_2) = \{X_1, X_2\}$, $f^A(X_3) = \{X_1, X_3\}$ and $f^B(X_3) = \{X_2, X_3\}$ (observe that $f^A(X_1) = f^B(X_1)$, and $f^A(X_2) = f^B(X_2)$). Therefore, in this case $\mathcal{Y} = \{\{X_1\}, \{X_1, X_2\}, \{X_1, X_3\}, \{X_2, X_3\}\}$.

Initially, the potentials repository R_Φ contains the potentials of N^A (a conditional probability for each variable given its parents): $\phi_1^A(X_1)$, $\phi_2^A(X_2, X_1)$, and $\phi_3^A(X_3, X_1)$ with time 0. We indicate the variables involved in each potential. The operations repository, R_O, will be empty. Table 1 contains the initial repositories. Notice that the superscript A has been omitted in order to simplify the notation.

Table 1. Initial state for R_Φ (left part) and R_O (right part).

Rep. of Potentials (R_Φ)		**Rep. of Operations (R_O)**				
Potential	**Time**	**Id**	**Type**	**Arg. 1**	**Arg. 2**	**Result**
$\phi_1(X_1)$	0					
$\phi_2(X_1, X_2)$	0					
$\phi_3(X_1, X_3)$	0					

Algorithm 2 Computation of marginals of p for subsets $\mathbf{Y} \in \mathcal{Y}$

function MARGINAL($N, \mathcal{Y}$)
  $t \leftarrow 1$
  for each $k = 1, \ldots, K$ **do**
    Let $\mathbf{Y}$ the subset $\mathbf{Y}_k$ in $\mathcal{Y}$
    Let $\mathbf{\Phi_Y}$ the family of potentials from $\mathbf{\Phi}$ relevant to subset $\mathbf{Y}$
    for $X_i \in v(\mathbf{\Phi_Y}) \setminus \mathbf{Y}$ **do** ▷ determine operations for the query
      Let $\mathbf{\Phi}_i = \{\phi \in \mathbf{\Phi_Y} : X_i \in v(\phi)\}$
      Assume $\mathbf{\Phi}_i = \{\phi_1, \ldots, \phi_L\}$
      $\psi = \phi_1$
      for $l = 2, \ldots, L$ **do**
        $\psi \leftarrow$ CONDSCOMBINE(ψ, ϕ_l, t)
        $t \leftarrow t + 1$
      end for
      $\psi \leftarrow$ CONDSMARGINALIZE(ψ, i, t)
      $t \leftarrow t + 1$
      $\mathbf{\Phi_Y} \leftarrow (\mathbf{\Phi_Y} \setminus \mathbf{\Phi}_i) \cup \{\psi\}$
    end for
    Assume $\mathbf{\Phi_Y} = \{\phi_1, \ldots, \phi_J\}$ ▷ compute joint distribution
    $\psi_k \leftarrow \phi_1$
    for $j = 2, \ldots, J$ **do**
      $\psi_k \leftarrow$ CONDSCOMBINE(ψ_k, ϕ_j, t)
      $t \leftarrow t + 1$
    end for
    Append ψ_k to LR
    Set *time* of ψ_k to -1
  end for
  $T \leftarrow t - 1$
  for each $t = 1, \ldots, T$ **do** ▷ start numerical computation using operations planning
    Select register with time t from R_O: $(t, type, \phi_1, \phi_2, \phi_r)$
    Compute numerical values of ϕ_r ▷ Actual computation
    DELETEIF(ϕ_1, t), DELETEIF(ϕ_2, t), DELETEIF(ϕ_r, t)
  end for
  return LR
end function

Algorithm 3 Conditional symbolic combination

function CONDSCOMBINE(ϕ_1, ϕ_2, t)
  if register $(id, comb, \phi_1, \phi_2, \phi_r)$ is in R_O **then**
    UPDATETIME(ϕ_1, t), UPDATETIME(ϕ_2, t), UPDATETIME(ϕ_r, t)
  else
    $\phi_r =$ SCOMBINE(ϕ_1, ϕ_2)
    Add register $(id, comb, \phi_1, \phi_2, \phi_r)$ to R_O with id as identifier
    UPDATETIME(ϕ_1, t), UPDATETIME(ϕ_2, t)
    Add ϕ_r to R_ϕ with $time = t$
  end if
  return ϕ_r
end function

Algorithm 4 Conditional symbolic marginalization

```
1: function CondSMarginalize(φ, i, t)
2:     if register (id, marg, φ, i, φ_r) is in R_O then
3:         UpdateTime(φ, t), UpdateTime(φ_r, t)
4:     else
5:         φ_r = SMarginalize(φ, i)
6:         Add register (id, marg, φ, i, φ_r) to R_O with id as identifier
7:         UpdateTime(φ, t)
8:         Add φ_r to R_φ with t as time
9:     end if
10:    return φ_r
11: end function
```

The first marginal to compute is for $\mathbf{Y} = \{X_1\}$. In this case, the set of relevant potentials is $\mathbf{\Phi_Y} = \{\phi_1\}$ and there are not operations to carry out. Therefore the first marginal is $\Psi_1 = \phi_1$ which is appended to LR.

The second marginal to be computed is for $\mathbf{Y} = \{X_1, X_2\}$. In this case, the relevant potentials are $\mathbf{\Phi_Y} = \{\phi_1, \phi_2\}$ and there are no variables to remove, but it is necessary to carry out the symbolic combination of ϕ_1 and ϕ_2 in order to compute Ψ_2 (lines 19–23 of Algorithm 2). If we call ϕ_4 the result, then the repositories after this operation will be as shown in Table 2.

Table 2. Repositories after $k = 2$.

Rep. of Potentials (R_Φ)		**Rep. of Operations (R_O)**				
Potential	**Time**	**Id**	**Type**	**Arg. 1**	**Arg. 2**	**Result**
$\phi_1(X_1)$	-1	1	Comb	ϕ_1	ϕ_2	ϕ_4
$\phi_2(X_1, X_2)$	1					
$\phi_3(X_1, X_3)$	0					
$\phi_4(X_1, X_2)$	-1					

The third marginal to compute is for set $\mathbf{Y} = \{X_1, X_3\}$. Now, the relevant potentials are $\mathbf{\Phi_Y} = \{\phi_1, \phi_3\}$. The situation is analogous to the computation of the previous marginal, with the difference that now the symbolic combination to carry out is $\phi_1 \cdot \phi_3$. The repositories status after $k = 3$ is shown in Table 3. We have that the third desired marginal is $\psi_3 = \phi_5$.

Table 3. Repositories after $k = 3$.

Rep. of Potentials (R_Φ)		**Rep. of Operations (R_O)**				
Potential	**Time**	**Id**	**Type**	**Arg. 1**	**Arg. 2**	**Result**
$\phi_1(X_1)$	-1	1	Comb	ϕ_1	ϕ_2	ϕ_4
$\phi_2(X_1, X_2)$	1	2	Comb	ϕ_1	ϕ_3	ϕ_5
$\phi_3(X_1, X_3)$	2					
$\phi_4(X_1, X_2)$	-1					
$\phi_5(X_1, X_3)$	-1					

Finally, for $k = 4$ we have to compute the marginal for $\mathbf{Y} = \{X_2, X_3\}$. The relevant potentials are now $\mathbf{\Phi_Y} = \{\phi_1, \phi_2, \phi_3\}$. Variable X_1 has to be deleted from this set of potentials. As all the potentials contain this variable, as a first step it is necessary to combine all of them, and afterwards to remove X_1 by marginalizing on $\{X_2, X_3\}$. Assume that the order of the symbolic operations is: first combine ϕ_1 and ϕ_2 and then its result is combined with ϕ_3; then this result is marginalized by removing X_1. Then the repositories after $k = 4$ are as presented in Table 4. The combination of ϕ_1 and ϕ_2 was previously carried out for $k = 2$ and therefore its result can be retrieved without new computations.

Table 4. Repositories after $k = 4$.

Rep. of Potentials (R_Φ)		Rep. of Operations (R_O)				
Potential	**Time**	**Id**	**Type**	**Arg. 1**	**Arg. 2**	**Result**
$\phi_1(X_1)$	-1	1	Comb	ϕ_1	ϕ_2	ϕ_4
$\phi_2(X_1, X_2)$	4	2	Comb	ϕ_1	ϕ_3	ϕ_5
$\phi_3(X_1, X_3)$	4	3	Comb	ϕ_1	ϕ_3	ϕ_5
$\phi_4(X_1, X_2)$	-1	4	Comb	ϕ_4	ϕ_3	ϕ_6
$\phi_5(X_1, X_3)$	-1	5	Marg	ϕ_6	1	ϕ_7
$\phi_6(X_1, X_2, X_3)$	5					
$\phi_7(X_2, X_3)$	-1					

After that, the numerical part of operations in Table 4 are carried out in the same order in which they are described in that table. In this process, after doing an operation with an identifier (*id*) equal to t, the potentials with *time* equal to t are removed from the R_Φ table. For example, in this case, potentials ϕ_2 and ϕ_3 can be removed from R_Φ after doing operation with $id = 4$ and potential ϕ_6 can be removed after operation with $id = 5$, leaving only in R_Φ the potentials containing the desired marginal potentials needed to compute the KL divergence between both networks (potentials with $time = -1$).

5. Experiments

In order to compare the computation approaches presented in the paper the experimentation uses a set of Bayesian networks available in the *bnlearn* [19] repository (https://www.bnlearn.com/bnrepository/, accessed on 24 August 2021). This library provides all the functions required for the process described below. Given a certain Bayesian network as defined in the repository:

- A dataset is generated using the *rbn* function. As explained in the library documentation, this function simulates random samples from a Bayesian network, using *forward/backward* sampling.
- The dataset is used for learning a Bayesian network. For this step, the *tabu* function is employed using the default setting (a dataset as unique argument). It is one of the structural learning methods available on *bnlearn*. Since the learned model could have unoriented links, the *cextend* function is required, which results in a Bayesian network consistent with the model passed as argument. Any other different learning algorithm could have been used, since the goal is to have an alternative Bayesian network that will be used later to calculate the Kullback–Leibler divergence with the methods described in Algorithms 1 and 2.

For each network, the Kullback–Leibler divergence is computed with the procedures presented using evidence propagation (see Algorithm 1) and using operations cache (described in Algorithm 2). The main purpose of the experiment is to get an estimation of the computation times required for both approaches. The obtained results are included in Table 5. It contains the following information:

- **Network** name.
- Number of **nodes**.
- Number of **arcs**.
- Number of **parameters** required for quantifying the uncertainty of the network.
- **time1**: Runtime using the algorithm without cache (Algorithm 1).
- **time2**: Runtime using the algorithm with cache (Algorithm 2).
- **ops**: Number of elementary operations stored in the operations repository R_O to compute all the necessary distributions for the calculation using Algorithm 2.
- **rep**: Number of operations that are repeated and that, thanks to the use of R_O and R_Φ, will be executed only once.
- **del**: Number of factors that were removed from the R_Φ with the consequent release of memory space for future calculations.

The experiments have been run in a desktop computer with an Intel(R) Xeon(R) Gold 6230 CPU working at 3.60 GHz (80 cores). It has 312 Gb of RAM memory. The operating system is Linux Fedora Core 34.

Table 5. Runtimes for KL computation without cache (time1) and with cache (time2).

Network	Nodes	Arcs	Parameters	Time1	Time2	Ops	Rep	Del
cancer	5	4	18	0.0313	**0.012**	48	21	13
earthquake	5	4	10	0.0316	**0.0091**	31	15	5
survey	5	6	21	0.0504	**0.0143**	49	23	12
asia	8	8	18	0.0787	**0.0173**	62	28	13
sachs	11	17	178	0.3484	**0.0409**	81	30	25
child	20	25	230	0.8796	**0.0726**	142	58	56
insurance	27	52	984	16.9990	**0.3788**	631	313	223
water	32	66	10,083	–	**8.6822**	640	299	170
mildew	35	46	540,150	–	**15.9393**	1278	955	150
alarm	37	46	509	4.0949	**0.3354**	638	415	132
barley	48	84	114,005	–	**205.6597**	2273	1695	328
hailfinder	56	66	2656	362.5262	**0.8498**	1197	921	127
hepar2	70	123	1453	23.1047	**1.4088**	1864	1459	242
win95pts	76	112	2656	404.4568	**1.0080**	960	458	314

It is observed that the calculation with the second method always offers shorter runtimes than the first one. The shortest runtimes are presented in the table with bold style. It is noteworthy that the case of three networks in which the method based on the use of evidence cannot be completed because the available memory capacity is exceeded: *water*, *mildew*, and *barley*. Moreover, the computational overhead required to manage operations and factor repositories is beneficial as it avoids the repetition of a significant number of operations and enables unnecessary potentials to be released, especially in the most complex networks.

6. Conclusions

Computing the KL divergence between the joint probabilities associated with two Bayesian networks is an important task that is relevant for many problems, for example assessing the accuracy of Bayesian network learning algorithms. However, in general, it is not possible to find this function implemented in software packages for probabilistic graphical models. In this paper, we provide an algorithm that uses local computation to calculate the KL divergence between two Bayesian networks. The algorithm is based in a procedure with two stages. The first one plans the operations determining the repeated operations and the times in which potentials are no longer necessary, while the second one carries out the numerical operations, taking care to reuse the results of repeated operations instead of repeating them and deallocating the memory space associated with useless potentials. Experiments show that this strategy saves time and space, especially in complex networks.

The functions have been implemented in Python taking as basis *pgmpy* software package and are available in the *github* repository: https://github.com/mgomez-olmedo/KL-pgmpy, accessed on 24 August 2021. The *README* file of the project offers details about the implementation and the methods available for reproducing the experiments.

In the future, we plan to further improve the efficiency of the algorithms. The main line will be to invest more time in the planning stage looking for deletion orderings minimizing the total number of operations or optimizing the order of combinations, when several potentials have to be multiplied.

Author Contributions: Conceptualization, S.M., A.C. and M.G.-O.; methodology, S.M., A.C. and M.G.-O.; software, S.M., A.C. and M.G.-O.; validation, S.M., A.C. and M.G.-O.; formal analysis, S.M., A.C. and M.G.-O.; investigation, S.M., A.C. and M.G.-O.; writing–original draft preparation, S.M., A.C. and M.G.-O.; visualization, S.M., A.C. and M.G.-O.; supervision, S.M., A.C. and M.G.-O.; funding acquisition, S.M., A.C. and M.G.-O. All authors have read and agreed to the published version of the manuscript.

Funding: This research was jointly supported by the Spanish Ministry of Education and Science under project PID2019-106758GB-C31 and the European Regional Development Fund (FEDER).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: We are very grateful to the anonymous reviewers for their valuable comments and suggestions that have contributed to the improvement of the paper.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. de Jongh, M.; Druzdzel, M.J. A comparison of structural distance measures for causal Bayesian network models. In *Recent Advances in Intelligent Information Systems, Challenging Problems of Science, Computer Science Series*; Springer: Basel, Switzerland, 2009; pp. 443–456.
2. Scutari, M.; Vitolo, C.; Tucker, A. Learning Bayesian networks from big data with greedy search: Computational complexity and efficient implementation. *Stat. Comput.* **2019**, *29*, 1095–1108. [CrossRef]
3. Talvitie, T.; Eggeling, R.; Koivisto, M. Learning Bayesian networks with local structure, mixed variables, and exact algorithms. *Int. J. Approx. Reason.* **2019**, *115*, 69–95. [CrossRef]
4. Natori, K.; Uto, M.; Ueno, M. Consistent learning Bayesian networks with thousands of variables. In *Advanced Methodologies for Bayesian Networks*; PMLR; 2017; pp. 57–68. Available online: https://proceedings.mlr.press/v73/natori17a (accessed on 29 July 2021).
5. Kullback, S. *Information Theory and Statistics*; Dover Publication: Mineola, NY, USA, 1968.
6. de Campos, L.M. A scoring function for learning Bayesian networks based on mutual information and conditional independence tests. *J. Mach. Learn. Res.* **2006**, *7*, 2149–2187.
7. Liu, H.; Zhou, S.; Lam, W.; Guan, J. A new hybrid method for learning Bayesian networks: Separation and reunion. *Knowl.-Based Syst.* **2017**, *121*, 185–197. [CrossRef]
8. Cano, A.; Gómez-Olmedo, M.; Moral, S. Learning Sets of Bayesian Networks. In Proceedings of the International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems, Lisbon, Portugal, 15–19 June 2020; Springer: Berlin/Heidelberg, Germany, 2020; pp. 151–164.
9. Koller, D.; Friedman, N. *Probabilistic Graphical Models: Principles and Techniques*; MIT Press: Cambridge, MA, USA, 2009.
10. Elvira Consortium. Elvira: An Environment for Probabilistic Graphical Models. In Proceedings of the 1st European Workshop on Probabilistic Graphical Models, Cuenca, Spain, 6–8 November 2002; pp. 222–230.
11. Kjærulff, U. *Approximation of Bayesian Networks through Edge Removals*; Technical Report IR-93-2007; Department of Mathematics and Computer Science, Aalborg University: Aalborg, Denmark, 1993.
12. Choi, A.; Chan, H.; Darwiche, A. On Bayesian Network Approximation by Edge Deletion. *arXiv* **2012**, arXiv:1207.1370.
13. Kjærulff, U. Reduction of computational complexity in Bayesian networks through removal of weak dependences. In *Uncertainty Proceedings 1994*; Elsevier: Amsterdam, The Netherlands, 1994; pp. 374–382.
14. Ankan, A.; Panda, A. pgmpy: Probabilistic graphical models using Python. In Proceedings of the 14th Python in Science Conference (SCIPY 2015), Austin, TX, USA, 6–12 June 2015.
15. Shafer, G.R.; Shenoy, P.P. Probability propagation. *Ann. Math. Artif. Intell.* **1990**, *2*, 327–351. [CrossRef]
16. Dechter, R. Bucket elimination: A unifying framework for reasoning. *Artif. Intell.* **1999**, *113*, 41–85. [CrossRef]
17. Shachter, R. Bayes-Ball: The Rational Pasttime (for Determining Irrelevance and Requisite Information in Belief Networks and Influence Diagrams). In Proceedings of the Fourteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-98), Madison, WI, USA, 24–26 July 1998; Morgan Kaufmann Publishers: San Francisco, CA, USA, 1998; pp. 48–487.
18. Cano, A.; Moral, S. Heuristic algorithms for the triangulation of graphs. In Proceedings of the International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems, Paris, France, 4–8 July 1994; Springer: Berlin/Heidelberg, Germany, 1994; pp. 98–107.
19. Scutari, M. Learning Bayesian networks with the bnlearn R package. *J. Stat. Softw.* **2010**, *35*, 1–22. [CrossRef]