Fast factorisation of probabilistic potentials and its application to approximate inference in Bayesian networks

Andrés Cano, Manuel Gómez-Olmedo, Cora B. Pérez-Ariza
Dept. of Computer Science and Artificial Intelligence, University of Granada, C/ Daniel Saucedo Aranda s/n, 18071 Granada, Spain
{*acu,mgomez,cora*}*@decsai.ugr.es*

Antonio Salmerón
Dept. Statistics and Applied Mathematics, University of Almería, La Cañada de San Urbano s/n, 04120 Almería, Spain
antonio.salmeron@ual.es

Received (received date)
Revised (revised date)

We present an efficient procedure for factorising probabilistic potentials represented as probability trees. This new procedure is able to detect some regularities that cannot be captured by existing methods. In cases where an exact decomposition is not achievable, we propose a heuristic way to carry out approximate factorisations guided by a parameter called factorisation degree, which is fast to compute. We show how this parameter can be used to control the tradeoff between complexity and accuracy in approximate inference algorithms for Bayesian networks.

Keywords: Bayesian networks; Probability trees; Factorisation; Probabilistic inference

1. Introduction

Bayesian networks[1] constitute an efficient representation of multivariate probability distributions. During the last decades, they have been successfully used as a tool for knowledge representation and reasoning under uncertainty. The reasoning process consists of the computation of the posterior distribution for some variables of interest given that the value of other variables is known. This task is called *probabilistic inference* or inference for short.

Several exact[2,3,4,5,6,7,8] and approximate[9,10,11,12,13] inference algorithms can be found in the literature, due to the fact that it is an NP-hard problem,[14,15] which justifies the study of new techniques and algorithms with the aim of widening the class of affordable problems. Some of the most relevant inference algorithms incorporate the ability of dealing with factorised representations of the potentials that represent the probabilistic information. Examples of these algorithms are Lazy propagation,[6] Variable elimination,[16,17] Lazy-penniless propagation[10] and Importance Sampling based on approximate pre-computation.[12,13]

The use of factorised representations of probabilistic potentials has been proven to be an appropriate way of controlling the tradeoff between complexity and accuracy of propagation algorithms, by means of the concept of *approximate factorisation* of the probability trees used to represent the probabilistic potentials.[18,19] In this paper we propose a method for computing factorisations of probabilistic potentials which improves the existing one in terms of efficiency. Furthermore the new procedure is able to detect regularities that would not be captured by previous methods.[20,18] We illustrate that the new factorisation procedure can be used for controlling the tradeoff between efficiency and accuracy in approximate inference algorithms, through a modified version of the Variable Elimination scheme.[16,17]

The rest of the paper is organised as follows. We establish the notation used throughout the manuscript and give some basic definitions in Sec. 2. The fast factorisation algorithm is described and analysed in Sec. 3. Its application to the design of an approximate propagation algorithm is given in Sec 4. The paper ends with conclusions in Sec. 5.

2. Preliminaries and notation

Throughout this paper, we will use uppercase letters to denote random variables, and boldfaced uppercase letters to denote random vectors, e.g. $\mathbf{X} = \{X_1, \ldots, X_n\}$. We will use the term *potential* to refer to any probabilistic information, including 'a priori', conditional and 'a posteriori' distributions and intermediate results of operations between them. Formally, a *potential* ϕ for a random vector $\mathbf{X}$ is a mapping $\phi : \Omega_{\mathbf{X}} \to \mathbb{R}_0^+$, where $\Omega_{\mathbf{X}}$ is the set of possible cases of $\mathbf{X}$ and $\mathbb{R}_0^+$ is the set on non-negative real numbers. The set of variables for which a potential ϕ is defined will be denoted by $\text{dom}(\phi)$. We will consider only discrete variables with a finite number of cases. By lowercase letters x (or $\mathbf{x}$) we denote some element of Ω_X (or $\Omega_{\mathbf{X}}$).

A *Bayesian network* is a directed acyclic graph where each node represents a random variable, and the topology of the graph encodes the independence relations among the variables, according to the *d*-separation criterion.[1] Given the independencies attached to the graph, the joint distribution is determined giving a probability distribution for each node conditioned on its parents, so that for a Bayesian network with variables $X_1, \ldots, X_n$, the joint distribution factorises as

$$p(x_1, \ldots, x_n) = \prod_{i=1}^{n} p_i(x_i | pa(x_i)), \tag{1}$$

where $pa(x_i)$ denotes a configuration of values of the parents of variable X_i in the network.

Several data structures have been used to represent probabilistic potentials, being probability tables and probability trees the most commonly used. In this paper we will concentrate on probability trees, as previous factorisation methods operate over them.

A *probability tree*[21,13] is a directed labeled tree, where each internal node represents a variable and each leaf node represents a piece of probabilistic information (a probability value or, in general, a non-negative real number with some probabilistic meaning as, for instance, a normalisation factor). Each internal node has one outgoing arc for each state of the variable associated with that node. The *size* of a tree $\mathcal{T}$, denoted as size($\mathcal{T}$), is defined as its number of leaves. A probability tree $\mathcal{T}$ on variables $\mathbf{X}$ represents a potential $\phi : \Omega_{\mathbf{X}} \rightarrow \mathbb{R}_0^+$ if for each $\mathbf{x} \in \Omega_{\mathbf{X}}$ the value $\phi(\mathbf{x})$ is the number stored in the leaf node that is reached by starting from the root node and selecting the child corresponding to coordinate x_i for each internal node labeled with X_i.

Probability trees are in general more compact representations of probabilistic potentials than tables, as they are able to capture asymmetric independencies (also called context specific independencies), and furthermore, trees allow to obtain even more compact representations in exchange of losing accuracy. This compact representation is achieved by pruning some leaves and replacing them by the average value. This is illustrated in Fig. 1, where it is understood that, if a variable does not appear in a given branch, it means that the value of the potential corresponding to that branch is the same for all the possible values of the missing variable. Note that the location of the variables in each branch of the tree is relevant for capturing context specific independencies, and also to allow more accurate approximations when pruning leaves. This issue has been deeply studied in the literature.[9,13]

x_1	x_2	x_3	$\phi(x_1,x_2,x_3)$
1	1	1	0.2
1	1	2	0.5
1	2	1	0.7
1	2	2	0.7
2	1	1	0.3
2	1	2	0.5
2	2	1	0.3
2	2	2	0.3

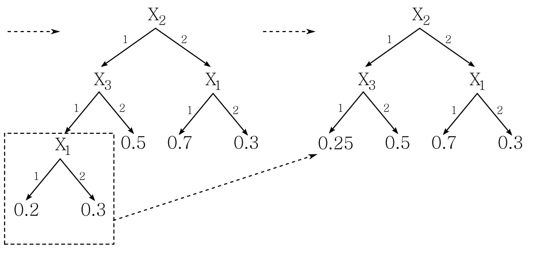

Fig. 1. A potential ϕ, a probability tree representing it and an approximation of it after pruning the branches underneath configuration $(X_2 = 1, X_3 = 1)$.

3. Fast factorisation of probability trees

In this section we describe our proposal for finding multiplicative decompositions of probabilistic potentials. Our method is in general valid for different representations of potentials, like probability tables and probability trees. However, we will focus on probability trees, as previous factorisation techniques operate over them. We will distinguish between the obtention of exact and approximate decompositions.

3.1. *Exact Decomposition*

By *exact decomposition* we mean the fact that a probability tree can be factorised as a product of two trees of smaller size. Previous factorisation methods[20,18] search for proportional sub-trees located below a given variable. Therefore, the performance is highly dependent on the order of the variables in the tree. As an example, consider the probability distribution for variables X, Y and Z represented as the probability tree shown in Fig. 2. It can be seen that the sub-trees below X, for branch $Y = 0$, are proportional. The classical factorisation procedure would decompose the tree in Fig. 2 as the product of the two trees in Fig. 3, where the $\times$ symbol indicates a point-wise product, i.e., the value of the decomposition for a given configuration of the involved variables, is equal to the result of multiplying the values obtained by evaluating that configuration in both trees. Notice that, in this case, the size of the resulting factorisation (which is the sum of the sizes of individual trees that comprise the factorisation) is higher than the one of the original tree, but applying the classical procedure to the tree in the left side of Fig. 3 it could be further decomposed, as the two sub-trees below X are also proportional when $Y = 0$.

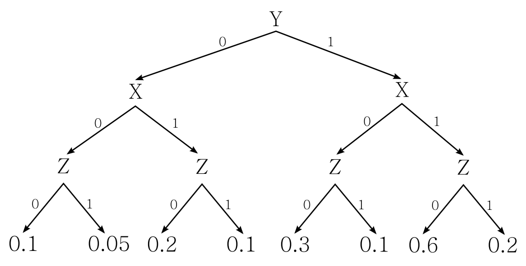

Fig. 2. A probability tree with proportional sub-trees.

But it can even be the case that the classical factorisation algorithm is not able to decompose a tree with respect to a given variable, if it is located away from the root of the tree. For instance, the tree in Fig. 4 can be expressed as the product of the two trees in Fig. 5, but such decomposition cannot be obtained using the classical factorisation technique, since variable X is located near to the leaves. In theory, classical factorisation could find the right decomposition, if the variables in the tree are re-arranged until the appropriate order is obtained. However, that would be too costly in terms of time.

In what follows we propose a factorisation technique able to deal with situations like the one described in Figs. 4 and 5, in a computationally efficient way. The key concept is the decomposability of trees, as stated in the next definition.

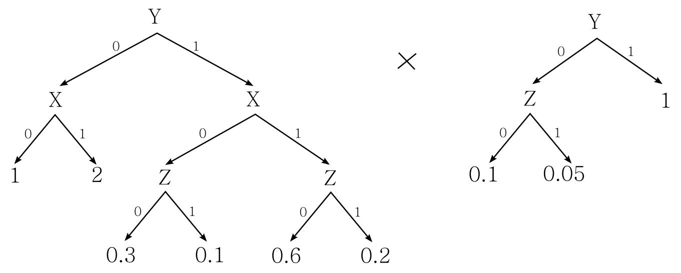

Fig. 3. Decompositon of the tree in Fig. 2 using classical decomposition.

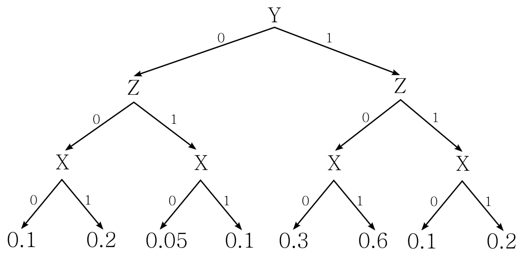

Fig. 4. Probability tree that cannot be factorised by variable X with classical factorisation.

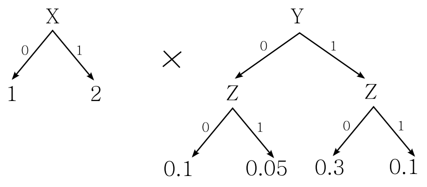

Fig. 5. Decomposition of the tree in Fig. 4, which cannot be obtained using classical factorisation.

Definition 1. A probability tree $\mathcal{T}$ defined for a set of variables $\mathbf{X}$ is said to be *decomposable with respect to* $\mathbf{Y} \subsetneq \mathbf{X}$ if there are two probability trees $\mathcal{T}_1$ and $\mathcal{T}_2$, defined for variables $\mathbf{Y}$ and $\mathbf{Z} \subsetneq \mathbf{X}$ respectively, such that

(1) $\mathbf{Y} \cap \mathbf{Z} = \emptyset$,
(2) $\mathbf{X} = \mathbf{Y} \cup \mathbf{Z}$ and
(3) $\mathcal{T} = \mathcal{T}_1 \times \mathcal{T}_2$.

Notice that the tree in Fig. 4 is decomposable with respect to X, as the three conditions in Def. 1 are met. A detailed procedure for finding a factorisation of a decomposable tree is described in Alg. 1. It makes use of the *restriction* operation over probability trees, defined as follows.

Definition 2. Let $\mathcal{T}$ be a probability tree over variables $\mathbf{X}$. Let $\mathbf{Y} \subsetneq \mathbf{X}$ be a set of variables in $\mathcal{T}$. The *restriction* of $\mathcal{T}$ to a value $\mathbf{y} \in \Omega_{\mathbf{Y}}$, denoted as $\mathcal{T}^{R(\mathbf{Y}=\mathbf{y})}$ is a tree defined on $\mathbf{X} \setminus \mathbf{Y}$, obtained from $\mathcal{T}$ by replacing each node labeled with $Y \in \mathbf{Y}$ by its child corresponding to value $y \in \mathbf{y}$.

Example 1. Let $\mathcal{T}$ be the probability tree in Fig. 4. The tree in the right handside of Fig.5 is actually the restriction of $\mathcal{T}$ to $X = 0$, that is, $\mathcal{T}^{R(X=0)}$.

The next proposition shows that, if a tree is decomposable, then Alg. 1 actually finds a decomposition consistent with Def. 1.

Proposition 1. *Let $\mathcal{T}$ be a probability tree defined for a set of variables $\mathbf{X}$. If $\mathcal{T}$ is* decomposable with respect to $\mathbf{Y} \subsetneq \mathbf{X}$, *then Alg. 1 returns two probability trees $\mathcal{T}_1$ and $\mathcal{T}_2$ that factorise $\mathcal{T}$ according to Def. 1.*

Proof. The sets of variables over which $\mathcal{T}_1$ and $\mathcal{T}_2$ are defined, are determined in Steps 9 and 10. It is clear that, according to the definition of restriction in Def. 2, conditions (1) and (2) in Def. 1 hold. Now we will show that condition (3) also holds after applying Alg. 1.

If $\mathcal{T}$ is decomposable, then for all $\mathbf{y}, \mathbf{z}$

$$\mathcal{T}(\mathbf{y}, \mathbf{z}) = \mathcal{T}_1(\mathbf{y}) \times \mathcal{T}_2(\mathbf{z}). \tag{2}$$

Also, according to Def. 2, for all $\mathbf{y}, \mathbf{z}$

$$\mathcal{T}(\mathbf{y}, \mathbf{z}) = \mathcal{T}^{R(\mathbf{Y}=\mathbf{y})}(\mathbf{z}). \tag{3}$$

Let $\mathbf{y}_0$ be the first configuration of $\mathbf{Y}$. It follows from Steps 7 and 9 that $\mathcal{T}_1(\mathbf{y}_0) = 1$. Therefore,

$$\mathcal{T}(\mathbf{y}_0, \mathbf{z}) = \mathcal{T}_1(\mathbf{y}_0) \times \mathcal{T}_2(\mathbf{z}) = \mathcal{T}_2(\mathbf{z}),$$

and according to Eq. (3), it means that

$$\mathcal{T}^{R(\mathbf{Y}=\mathbf{y}_0)}(\mathbf{z}) = \mathcal{T}_2(\mathbf{z}). \tag{4}$$

For any other configuration $\mathbf{y}_j \in \Omega_{\mathbf{Y}}$, we can write

$$\mathcal{T}(\mathbf{y}_j, \mathbf{z}) = \mathcal{T}_1(\mathbf{y}_j) \times \mathcal{T}_2(\mathbf{z}) = \mathcal{T}^{R(\mathbf{Y}=\mathbf{y}_j)}(\mathbf{z}),$$

and using Eq. (4),

$$\mathcal{T}^{R(\mathbf{Y}=\mathbf{y}_j)}(\mathbf{z}) = \mathcal{T}_1(\mathbf{y}_j) \times \mathcal{T}^{R(\mathbf{Y}=\mathbf{y}_0)}(\mathbf{z}) \Rightarrow \mathcal{T}_1(\mathbf{y}_j) = \frac{\mathcal{T}^{R(\mathbf{Y}=\mathbf{y}_j)}(\mathbf{z})}{\mathcal{T}^{R(\mathbf{Y}=\mathbf{y}_0)}(\mathbf{z})},$$

which corresponds to the calculation in Step 7. Finally, notice that the re-scaling in Steps 12 and 13 do not affect this result, as

$$\begin{aligned}
\mathcal{T}_1 \times \frac{s_{\mathcal{T}}}{s_{\mathcal{T}_1}} \times \mathcal{T}_2 \times \frac{1}{s_{\mathcal{T}_2}} &= \mathcal{T}_1 \times \mathcal{T}_2 \times \frac{\sum_{\mathbf{yz}} t(\mathbf{y},\mathbf{z})}{(\sum_{\mathbf{y}} t_1(\mathbf{y}))(\sum_{\mathbf{z}} t_2(\mathbf{z}))} \\
&= \mathcal{T}_1 \times \mathcal{T}_2 \times \frac{\sum_{\mathbf{y},\mathbf{z}} t(\mathbf{y},\mathbf{z})}{\sum_{\mathbf{y}} t_1(\mathbf{y})(\sum_{\mathbf{z}} t_2(\mathbf{z}))} \\
&= \mathcal{T}_1 \times \mathcal{T}_2 \times \frac{\sum_{\mathbf{y},\mathbf{z}} t(\mathbf{y},\mathbf{z})}{\sum_{\mathbf{y}} \sum_{\mathbf{z}} t_1(\mathbf{y}) t_2(\mathbf{z})} \\
&= \mathcal{T}_1 \times \mathcal{T}_2 \times \frac{\sum_{\mathbf{y},\mathbf{z}} t(\mathbf{y},\mathbf{z})}{\sum_{\mathbf{y}} \sum_{\mathbf{z}} t(\mathbf{y},\mathbf{z})} \\
&= \mathcal{T}_1 \times \mathcal{T}_2 .
\end{aligned}$$

Therefore, it follows that $\mathcal{T} = \mathcal{T}_1 \times \mathcal{T}_2$ and thus condition (3) in Def. 1 holds. $\square$

```
   Factorise($\mathcal{T}$,$\mathbf{Y}$)
   Input: A probability tree $\mathcal{T}$ on $\mathbf{X}$ and set of variables $\mathbf{Y} \subset \mathbf{X}$.
   Output: A decomposition of $\mathcal{T}$ with respect to $\mathbf{Y}$, given as two probability
           trees $\{T_1, T_2\}$.
 1 begin
 2     Let $\mathbf{y}_0$,...,$\mathbf{y}_{r-1}$ be the elements of $\Omega_{\mathbf{Y}}$.
 3     $\mathbf{Z} \leftarrow \mathbf{X} \setminus \mathbf{Y}$.
 4     Let $(\mathbf{Z} = \mathbf{z})$ be any configuration for all variables in $\mathbf{Z}$.
 5     Let $\alpha_0, \ldots, \alpha_{r-1}$ be the leaves of tree $\mathcal{T}^{R(\mathbf{Z}=\mathbf{z})}$.
 6     for $i \leftarrow 0$ to $r-1$ do
 7         Let $\beta_i = \frac{\alpha_i}{\alpha_0}$.
 8     end
 9     Let $\mathcal{T}_1$ be a tree with the variables in $\mathbf{Y}$ as inner nodes and $\beta_0, \ldots, \beta_{r-1}$
       as leaves.
10     $\mathcal{T}_2 \leftarrow \mathcal{T}^{R(\mathbf{Y}=\mathbf{y}_0)}$.
11     Let $s_{\mathcal{T}}, s_{\mathcal{T}_1}$ and $s_{\mathcal{T}_2}$ be the sum of all the values in $\mathcal{T}, \mathcal{T}_1$ and $\mathcal{T}_2$
       respectively.
12     $\mathcal{T}_1 \leftarrow \mathcal{T}_1 \times \frac{s_{\mathcal{T}}}{s_{\mathcal{T}_1}}$.
13     $\mathcal{T}_2 \leftarrow \mathcal{T}_2 \times \frac{1}{s_{\mathcal{T}_2}}$.
14     return $\{\mathcal{T}_1, \mathcal{T}_2\}$
15 end
```

Algorithm 1: Fast factorisation of probability trees.

The next example illustrates the way in which Alg. 1 carries out the decomposition.

Example 2. Let $\mathcal{T}$ be the tree in Fig. 4. We will use Alg. 1 to decompose it with respect to variable X. The first action is actually performed in Step 4, where a configuration is selected for those variables in the tree, different from the one with respect to which we are going to decompose. In this case, a configuration for (Y, Z) has to be selected. Assume we choose configuration $(Y = 0, Z = 0)$. Step 5 requires the computation of tree $\mathcal{T}^{R(Y=0,Z=0)}$, which is a tree that only contains variable X, and whose leaves are $\alpha_0 = 0.1$ for $X = 0$ and $\alpha_1 = 0.2$ for $X = 1$. In Step 7, the β coefficients are computed:

$$\beta_0 = \frac{\alpha_0}{\alpha_0} = \frac{0.1}{0.1} = 1, \quad \beta_1 = \frac{\alpha_1}{\alpha_0} = \frac{0.2}{0.1} = 2.$$

Next, a tree is constructed in Step 9 with X as unique variable and β_0 and β_1 as leaves. It corresponds to the leftmost tree in Fig. 5. Step 10 computes the second tree in the decomposition, by restricting $\mathcal{T}$ to $(X = 0)$. This tree is shown in the right side of Fig. 5. Finally, Steps 11, 12 and 13 re-scale the values in the decomposition in order to guarantee that the total mass is the same as in the original tree. The re-scaled decomposition is shown in Fig. 6.

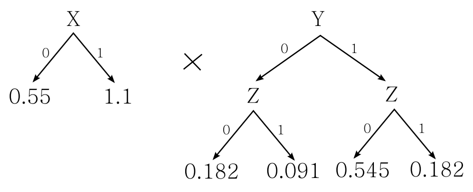

Fig. 6. Modified decomposition obtained from Fig. 5 after making the total mass be equal to the one in the original tree in Fig. 4.

An important feature of this decomposition scheme is its low complexity. This is specially interesting because it allows fast factorisation to be included in other algorithms (for instance, inference algorithms) without increasing the complexity order.

Lemma 1. *The complexity of Alg. 1 is linear in the size of the input tree, in the worst case.*

Proof. The complexity is determined by Steps 5, 10 and 11, which compute the restriction of the input tree and the sum of the input and output trees. The restriction operation is, in the worst case, linear in the size of the input tree as it can be obtained just by visiting all the leaves in the tree and keeping those consistent with the restricting configuration. The sum is also linear as it requires to visit all the leaves in the tree. □

3.2. *Approximate Decomposition*

As shown in Prop. 1, Alg. 1 finds the correct factorisation of a tree that is actually decomposable. However, it may happen that a tree is not decomposable with respect to a set of variables, but perhaps it is possible to factorise it in such a way that the product of the resulting trees is not far away from the original one. The next example illustrates this fact.

Example 3. Consider the probability tree in Fig. 4, but with the first two leaves equal to 0.11 and 0.19 instead of 0.1 and 0.2 respectively. After such modification, the tree is no longer decomposable. If we apply Alg. 1 to factorise such tree with respect to variable X, we obtain the decomposition in Fig. 7. Notice that this decomposition is actually an approximation. In fact, if we multiply again the two trees, the result is the tree in Fig. 8, which is not exactly the same as the original one.

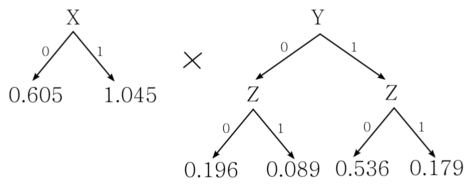

Fig. 7. Decomposition of the tree in Fig. 8, using Alg. 1.

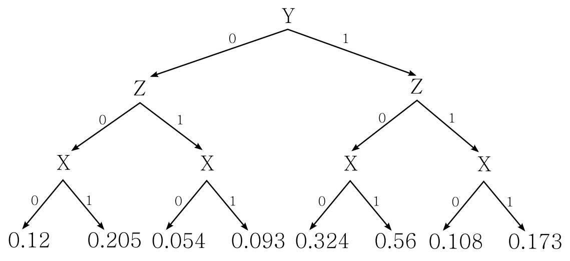

Fig. 8. Result of multiplying the two trees in Fig. 7.

The concept of approximate factorisation has been studied previously,[18] but the limitation of that approach is the same as the one described in the case of exact classical factorisation. Here we analyze how to extend the factorisation procedure explained above to the case in which a tree is not exactly decomposable.

If a tree is not exactly decomposable with respect to a given set of variables, we propose to use the *extended Kullback-Leibler divergence*[22] as a basis to determine how far from exact factorisation a given decomposition is. This divergence measure is an extension of Kullback-Leibler divergence[23] to unnormalised potentials. It is an important feature, as in general we deal with unnormalised potentials (i.e., potentials that do not sum up to one), especially after applying Alg. 1. For two unnormalised potentials p and q, it is defined as

$$eKL(p,q) = \sum_x p(x)\log\frac{p(x)}{q(x)} + \sum_x (q(x) - p(x)). \tag{5}$$

From now on, whenever we mention the eKL divergence between probability trees we understand the eKL divergence between the potentials represented by those trees. The key result is given in the next theorem, which provides an upper bound of the eKL divergence for a given decomposition.

Theorem 1. *Let $\mathcal{T}$ be a probability tree to be decomposed with respect to a set of variables $\mathbf{Y}$. Let $\mathbf{X}$ be the set of variables for which $\mathcal{T}$ is defined. Let be $\mathbf{Z} = \mathbf{X} \setminus \mathbf{Y}$. Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be the output of function **Factorise($\mathcal{T}$,$\mathbf{Y}$)**, described in Alg. 1. Then it holds that*

$$eKL(\mathcal{T}, \mathcal{T}_1 \otimes \mathcal{T}_2) \leq \sum_{\mathbf{x}} t(\mathbf{x})\log t(\mathbf{x}) - \left(\sum_{\mathbf{x}} t(\mathbf{x})\right)\left(\sum_{\mathbf{y}:t_1(\mathbf{y})\leq 1} \log t_1(\mathbf{y}) + \sum_{\mathbf{z}} \log t_2(\mathbf{z})\right), \tag{6}$$

where t, t_1 and t_2 are the potentials represented by trees $\mathcal{T}, \mathcal{T}_1$ and $\mathcal{T}_2$ respectively.

Proof.

$$\begin{aligned}
eKL(\mathcal{T}, \mathcal{T}_1 \otimes \mathcal{T}_2) &= \sum_{\mathbf{x}} t(\mathbf{x}) \log\frac{t(\mathbf{x})}{t_1(\mathbf{y})t_2(\mathbf{z})} + \sum_{\mathbf{x}} (t_1(\mathbf{y})t_2(\mathbf{z}) - t(\mathbf{x})) \\
&= \sum_{\mathbf{y},\mathbf{z}} t(\mathbf{y},\mathbf{z}) \log\frac{t(\mathbf{y},\mathbf{z})}{t_1(\mathbf{y})t_2(\mathbf{z})} + \sum_{\mathbf{y},\mathbf{z}} (t_1(\mathbf{y})t_2(\mathbf{z}) - t(\mathbf{y},\mathbf{z})) \\
&= \sum_{\mathbf{y},\mathbf{z}} t(\mathbf{y},\mathbf{z}) \left(\log t(\mathbf{y},\mathbf{z}) - \log t_1(\mathbf{y}) - \log t_2(\mathbf{z})\right) \\
&\quad + \sum_{\mathbf{y},\mathbf{z}} (t_1(\mathbf{y})t_2(\mathbf{z}) - t(\mathbf{y},\mathbf{z})) \\
&= \sum_{\mathbf{y},\mathbf{z}} t(\mathbf{y},\mathbf{z}) \log t(\mathbf{y},\mathbf{z}) - \sum_{\mathbf{y},\mathbf{z}} t(\mathbf{y},\mathbf{z}) \log t_1(\mathbf{y}) - \sum_{\mathbf{y},\mathbf{z}} t(\mathbf{y},\mathbf{z}) \log t_2(\mathbf{z}) \\
&\quad + \sum_{\mathbf{y},\mathbf{z}} t_1(\mathbf{y})t_2(\mathbf{z}) - \sum_{\mathbf{y},\mathbf{z}} t(\mathbf{y},\mathbf{z}).
\end{aligned}$$

Now, let us denote by t_1^* and t_2^* the potentials corresponding to the trees $\mathcal{T}_1$ and $\mathcal{T}_2$ in Steps 9 and 10 of Alg. 1, i.e., before re-scaling the trees. Then,

$$\begin{aligned}\sum_{\mathbf{y},\mathbf{z}} t_1(\mathbf{y})t_2(\mathbf{z}) &= \sum_{\mathbf{y},\mathbf{z}} t_1^*(\mathbf{y})\frac{s_{\mathcal{T}}}{s_{\mathcal{T}_1}} t_2^*(\mathbf{z})\frac{1}{s_{\mathcal{T}_2}}\\ &= \frac{s_{\mathcal{T}}}{s_{\mathcal{T}_1}s_{\mathcal{T}_2}}\sum_{\mathbf{y},\mathbf{z}} t_1^*(\mathbf{y})t_2^*(\mathbf{z})\\ &= \frac{s_{\mathcal{T}}}{s_{\mathcal{T}_1}s_{\mathcal{T}_2}}\left(\sum_{\mathbf{y}} t_1^*(\mathbf{y})\right)\left(\sum_{\mathbf{z}} t_2^*(\mathbf{z})\right)\\ &= \frac{s_{\mathcal{T}}s_{\mathcal{T}_1}s_{\mathcal{T}_2}}{s_{\mathcal{T}_1}s_{\mathcal{T}_2}} = s_{\mathcal{T}} = \sum_{\mathbf{y},\mathbf{z}} t(\mathbf{y},\mathbf{z}),\end{aligned}$$

where $s_{\mathcal{T}}, s_{\mathcal{T}_1}$ and $s_{\mathcal{T}_2}$ are defined in Step 11 of Alg. 1. Hence,

$$\begin{aligned}eKL(\mathcal{T},\mathcal{T}_1\otimes\mathcal{T}_2) &= \sum_{\mathbf{y},\mathbf{z}} t(\mathbf{y},\mathbf{z})\log t(\mathbf{y},\mathbf{z}) - \sum_{\mathbf{y},\mathbf{z}} t(\mathbf{y},\mathbf{z})\log t_1(\mathbf{y}) - \sum_{\mathbf{y},\mathbf{z}} t(\mathbf{y},\mathbf{z})\log t_2(\mathbf{z})\\ &= \sum_{\mathbf{y},\mathbf{z}} t(\mathbf{y},\mathbf{z})\log t(\mathbf{y},\mathbf{z}) - \sum_{\mathbf{y}}\left((\log t_1(\mathbf{y}))\sum_{\mathbf{z}} t(\mathbf{y},\mathbf{z})\right)\\ &\quad -\sum_{\mathbf{z}}\left((\log t_2(\mathbf{z}))\sum_{\mathbf{y}} t(\mathbf{y},\mathbf{z})\right).\end{aligned}$$

Notice that, since the values in t are not negative, it holds that

$$\sum_{\mathbf{z}} t(\mathbf{y},\mathbf{z}) \le \sum_{\mathbf{y}',\mathbf{z}} t(\mathbf{y}',\mathbf{z}) \tag{7}$$

and

$$\sum_{\mathbf{y}} t(\mathbf{y},\mathbf{z}) \le \sum_{\mathbf{y},\mathbf{z}'} t(\mathbf{y},\mathbf{z}'). \tag{8}$$

Furthermore, the values in t_2 are guaranteed to be lower than 1, and therefore their log is a negative number. Thus, we can write

$$\begin{aligned}eKL(\mathcal{T},\mathcal{T}_1\otimes\mathcal{T}_2) &\le \sum_{\mathbf{y},\mathbf{z}} t(\mathbf{y},\mathbf{z})\log t(\mathbf{y},\mathbf{z}) - \sum_{\mathbf{y}:t_1(\mathbf{y})\le 1}\left((\log t_1(\mathbf{y}))\sum_{\mathbf{y}',\mathbf{z}} t(\mathbf{y}',\mathbf{z})\right)\\ &\quad -\sum_{\mathbf{z}}\left((\log t_2(\mathbf{z}))\sum_{\mathbf{y},\mathbf{z}'} t(\mathbf{y},\mathbf{z}')\right)\\ &= \sum_{\mathbf{x}} t(\mathbf{x})\log t(\mathbf{x}) - \left(\sum_{\mathbf{x}} t(\mathbf{x})\right)\left(\sum_{\mathbf{y}:t_1(\mathbf{y})\le 1}\log t_1(\mathbf{y}) + \sum_{\mathbf{z}}\log t_2(\mathbf{z})\right),\end{aligned}$$

which completes the proof. □

Note that, if a decomposition is exact, then $eKL(\mathcal{T}, \mathcal{T}_1 \otimes \mathcal{T}_2)$ is equal to 0, and the further a decomposition is to the exact one, the higher the value of the divergence reaches.[22] Observe also that the upper bound given in Eq. (6), actually depends on the specific decomposition through the term

$$S = \sum_{\mathbf{y}:t_1(\mathbf{y})\leq 1} \log t_1(\mathbf{y}) + \sum_{\mathbf{z}} \log t_2(\mathbf{z}),$$

which suggests that S could be used as a measure of the degree of decomposability of a tree $\mathcal{T}$ with respect to $\mathbf{Y}$. The computation of S is rather fast, as it requires a time linear on the size of $\mathcal{T}_1$ and $\mathcal{T}_2$. Hence, we use the reasoning above to formally define the degree of decomposability of a tree with respect to a given set of variables, called the *factorisation degree*, as follows.

Definition 3. Let $\mathcal{T}$ be a probability tree. Let $\mathbf{X}$ be the set of variables for which $\mathcal{T}$ is defined, and $\mathbf{Y} \subset \mathbf{X}$. Let $\mathbf{Z} = \mathbf{X} \setminus \mathbf{Y}$. Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be the output of function **Factorise**($\mathcal{T}$,$\mathbf{Y}$), described in Alg. 1. We define the *factorisation degree* of $\mathcal{T}$ *with respect to* $\mathbf{Y}$ as

$$\mathrm{fd}(\mathcal{T}, \mathbf{Y}) = \sum_{\mathbf{y}:t_1(\mathbf{y})\leq 1} \log t_1(\mathbf{y}) + \sum_{\mathbf{z}} \log t_2(\mathbf{z}), \tag{9}$$

where t_1 and t_2 are the potentials represented by $\mathcal{T}_1$ and $\mathcal{T}_2$ respectively.

The factorisation degree defined above provides a heuristic way to choose the variable or set of variables with respect to which a tree can be decomposed, producing the lowest error in terms of eKL divergence. It is only heuristic since what is minimised is not the eKL divergence itself, but an upper bound, as given in Th. 1. This heuristic suggests a way to control the tradeoff between accuracy and complexity in approximate inference algorithms for Bayesian networks, namely by establishing a threshold of factorisation degree, and decomposing those trees for which there is a set of variables whose factorisation degree surpasses such threshold. Notice that, according to Def. 3 and Th. 1, the higher the factorisation degree, the lower the bound above the divergence between the original and decomposed representation of the tree. Therefore, by setting a lower threshold, more trees are potentially decomposed, and therefore the complexity of the inference problem is reduced as it has to deal with smaller trees, in exchange of losing accuracy.

The effectiveness of this approach depends on whether or not the kind of regularity characterised in Def. 1 is actually found in real-world problems. In order to investigate this fact, we have analysed four real-world networks commonly used as a benchmark for approximate inference algorithms. These networks are called Munin,[24] Andes,[25] Barley[26] and Water.[27] The analysis consisted of decomposing their conditional distributions according to the variable with highest factorisation degree, and measuring the root mean squared error between the original probability tree and its decomposition. The results are shown in Fig. 9, which is a beanplot[28]

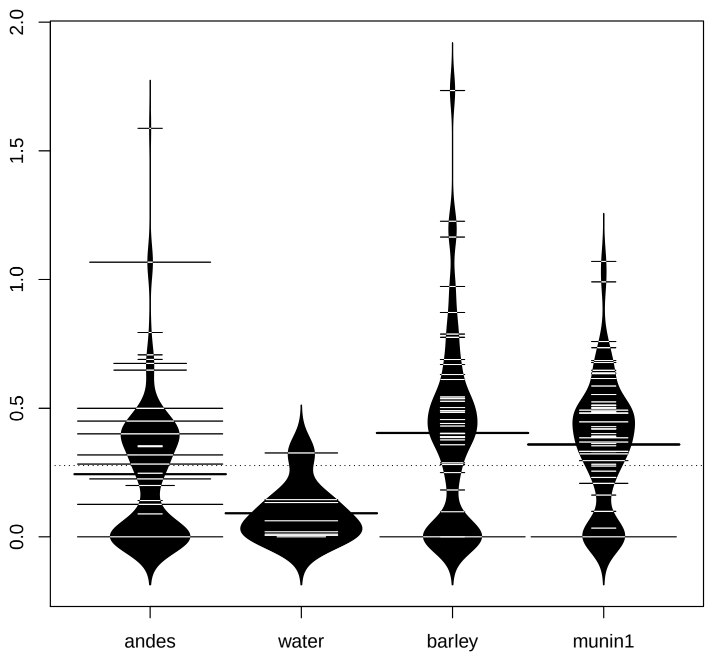

Fig. 9. Decomposability in four real-world examples. The plot shows the distribution of the potentials that, after decomposing, attained different levels of error.

that displays the empirical distribution of the errors obtained for the four networks. It can be seen how a high amount of distributions can be factorised introducing a very low error, close to zero, specially in the case of the Water network. Of course there are many potentials for which it is not possible to carry out the decomposition without introducing a large error, but this is a common fact in methods for approximating probability trees. For instance, consider the tree in the rightmost part of Fig. 1. It is clear that it can no longer be approximated using tree pruning, unless a high error in the approximation is admitted.

4. Applying fast factorisation to approximate inference in Bayesian networks

In this section we show how fast factorisation can be used for controlling the trade-off between complexity and accuracy in approximate inference algorithms. As a background, we have chosen the variable elimination algorithm,[16,17] described in Alg. 2, that is actually an exact inference algorithm, which computes the posterior distribution for all the unobserved variables in a given Bayesian network, given a set of variables **E** for which their values are known to be **e**. We assume that any probabilistic potential is represented as a probability tree, and therefore the restriction operation in Step 2 corresponds to Def. 2. We have considered two variations over the basic variable elimination algorithm in order to carry out approximate inference.

```
Variable_Elimination(X,$W$,E,e,$P$)
Input: The variables in the network ($\mathbf{X}$), an observation $\mathbf{E}=\mathbf{e}$, the target
       variable ($W$) and a set of probability trees, $P$, over the variables in $\mathbf{X}$.
Output: The posterior distribution of $W$ given $\mathbf{E}=\mathbf{e}$.
1  Let $t_i$, $i=1,\ldots,k$, be the probability trees in $P$.
2  $T := \{t_i^{R(\mathbf{E}=\mathbf{e})},\ i=1,\ldots,k\}$.
3  foreach $U \in \mathbf{X} \setminus \mathbf{E} \setminus \{W\}$ do
4      $T_U := \{t \in T | U \in \mathrm{dom}(t)\}$.
5      Let $g_U$ be the product of the trees in $T_U$.
6      Let $r_U$ be the result of marginalising out variable $U$ from $g_U$.
7      $T := (T \setminus T_U) \cup \{r_U\}$.
8  end
9  Let $t$ be the product of the trees in $T$.
10 Normalise $t$ in order to make it sum to 1.
11 return $t$.
```

Algorithm 2: Pseudo-code of the variable elimination algorithm.

The setting we have established is simple, in order to facilitate the evaluation of the real impact of using factorisation of probability trees as a means to control the level of approximation, and also to be able to compare this approach with the well known approximation method based on tree pruning (see Sec. 2), successfully used as the fundamental of various approximate algorithms.[9,13] The idea is to simplify the inference problem by reducing the size of the initial probability distributions in the network, using either tree pruning or factorisation of the trees. Both schemes are described in Alg. 3 and 4 respectively.

Procedure **Prune_VE(B,E,e,α)** (Alg. 3) carries out the approximation by pruning the probability trees corresponding to the initial conditional distribution in the network. The pruning is controlled by parameter α. Intuitively, this parameter indicates that sub-trees whose entropy is higher than the entropy of the binary probability distribution $\{0.5-\alpha, 0.5+\alpha\}$ will be pruned, i.e., replaced by a single value equal to the average of all the values in the subtree.[13] On the other hand, algorithm **Factorise_VE(B,E,e,d)** (Alg. 4) carries out the approximation by factorising those initial distributions in the network for which the best factorisation degree obtained by any of its variables, surpasses a given threshold d.

```
   Prune_VE(B,E,e,$\alpha$)
   Input: A Bayesian network $\mathbf{B}$ and an observation $\mathbf{E} = \mathbf{e}$. A threshold $\alpha$ for
          pruning the initial distributions.
   Output: The posterior distribution of all the unobserved variables in the
           network, given $\mathbf{E} = \mathbf{e}$.
 1 Let $\mathbf{X}$ be the variables in $\mathbf{B}$.
 2 Let $P = \{t_i, i = 1, \ldots, n\}$, be the probability trees representing the
   conditional distributions in $\mathbf{B}$.
 3 foreach $t \in P$ do
 4     Let $t'$ be the result of pruning $t$ according to parameter $\alpha$.
 5     $P \leftarrow (P \setminus \{t\}) \cup \{t'\}$.
 6 end
 7 $R = \emptyset$.
 8 foreach $W \in \mathbf{X} \setminus \mathbf{E}$ do
 9     $t \leftarrow$ Variable_Elimination($\mathbf{X}$,$W$,$\mathbf{E}$,$\mathbf{e}$,$P$).
10     $R \leftarrow R \cup \{t\}$.
11 end
12 return $R$.
```

Algorithm 3: Pseudo-code of the variable elimination algorithm with tree pruning of the initial distributions.

Using both procedures, we carried out a series of tests in order to check that parameter d is appropriate for controlling the complexity of the inference process, in a similar way as parameter α. The tests consisted of running both algorithms over the four real-world networks mentioned in Sec. 3.2 (Munin, Andes, Barley and Water) with different values of α and d. For each run, we measured the execution time, error in the estimation of the posterior probabilities and average and maximum size of the probability trees handled during the inference process. The error was measured using Fertig and Mann's divergence,[29] which is defined as follows. For one variable X, the error is computed as

$$G(X) = \sqrt{\frac{1}{|\Omega_X|} \sum_{x \in \Omega_X} \frac{(\hat{p}(x|\mathbf{e}) - p(x|\mathbf{e}))^2}{p(x|\mathbf{e})(1 - p(x|\mathbf{e}))}} \ , \tag{10}$$

where $p(x|\mathbf{e})$ is the exact *posterior* probability, $\hat{p}(x|\mathbf{e})$ is the approximate value and $|\Omega_X|$ is the number of possible values of variable X. For a set of variables $\mathbf{X}$, the error is:

$$G(\mathbf{X}) = \sqrt{\sum_{X \in \mathbf{X}} G(X)^2} \ . \tag{11}$$

```
Factorise_VE(B,E,e,$d$)
Input: A Bayesian network ($\mathbf{B}$) and an observation ($\mathbf{E} = \mathbf{e}$). A factorisation
       degree threshold ($d$) for decomposing the initial distributions.
Output: The posterior distribution of all the unobserved variables in the
        network, given $\mathbf{E} = \mathbf{e}$.
1  Let $\mathbf{X}$ be the variables in $\mathbf{B}$.
2  Let $P = \{t_i, i = 1, \ldots, n\}$, be the probability trees representing the
   conditional distributions in $\mathbf{B}$.
3  $P' \leftarrow \emptyset$.
4  foreach $t \in P$ do
5      Let $Y_1, \ldots, Y_k$ be the variables in $t$.
6      Compute $\mathrm{fd}(t, Y_i)$, $i = 1, \ldots, k$ according to Eq. (9).
7      $Y \leftarrow \arg \max_{i=1,\ldots,k} \mathrm{fd}(t, Y_i)$.
8      if $\mathrm{fd}(t, Y) > d$ then
9          $F \leftarrow$ Factorise($t$,$Y$).
10     end
11     else
12         $F \leftarrow \emptyset$.
13     end
14     $P' \leftarrow P' \cup F$.
15 end
16 $R = \emptyset$.
17 foreach $W \in \mathbf{X} \setminus \mathbf{E}$ do
18     $t \leftarrow$ Variable_Elimination($\mathbf{X}$,$W$,$\mathbf{E}$,$\mathbf{e}$,$P'$).
19     $R \leftarrow R \cup \{t\}$.
20 end
21 return $R$.
```

Algorithm 4: Pseudo-code of the variable elimination algorithm with factorisation of the initial distributions.

Fertig and Mann's divergence is an appropriate measure for comparing approximate inference algorithms, as it takes into account the magnitude of the exact value when evaluating the approximation. More precisely, it gives more weight to errors made when estimating extreme probabilities (close to 0 or 1), as it can be easily checked that the denominator in Eq. (10) is maximised for $p(x|\mathbf{e}) = 0.5$.

The results of the experiments are summarised in Figs. 10 to 15. In general, it can be said that parameter d can be used to control the approximation level in a similar way as parameter α (Figs. 10 and 11). The only anomaly is detected in the case of network Munin (see the right side of Fig. 10), where for the first point, execution time is higher than for others with lower errors. However, the same behaviour can be observed for the α parameter in this case. It can be seen

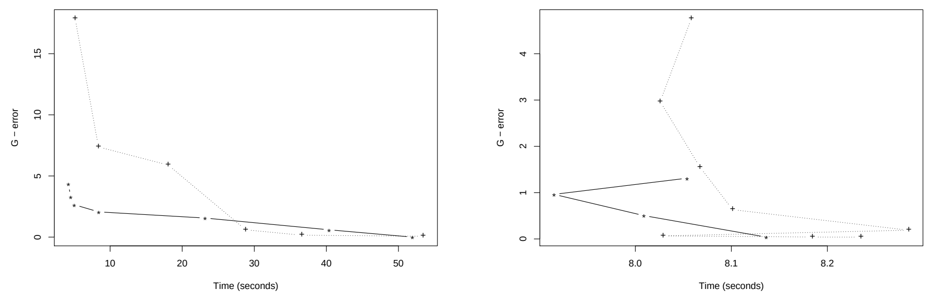

Fig. 10. Error vs. time for the Barley (left) and Munin (right) networks. The solid line corresponds to method **Factorise_VE** and the dotted one to **Prune_VE**.

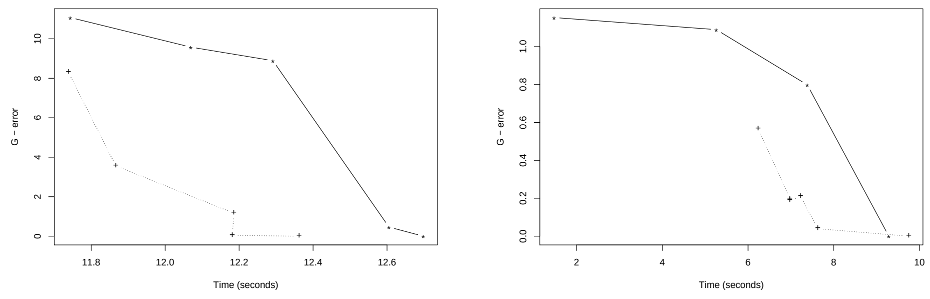

Fig. 11. Error vs. time for the Andes (left) and Water (right) networks. The solid line corresponds to method **Factorise_VE** and the dotted one to **Prune_VE**.

that for networks Barley and Munin, the convergence to low error values is reached more quickly by algorithm **Factorise_VE**, while the contrary happens for networks Andes and Water.

Regarding the size of the potentials involved in the calculations during the inference process, the experiments show how the average size is lower for algorithm **Factorise_VE** (see Figs. 12 and 13). However, the maximum size of such potentials is lower when using algorithm **Prune_VE** (see Figs. 14 and 15).

5. Conclusions

In this paper we have introduced a new and fast procedure for factorising probability trees. An important feature of the proposed algorithm is related to its capability for obtaining optimal decompositions in case that the tree is actually decomposable, in the sense that the decomposition is as compact as possible. We have also shown

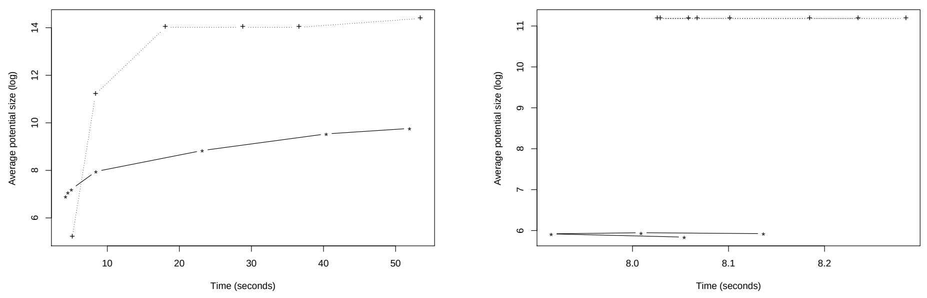

Fig. 12. Average potential size (in logarithmic scale) vs. time for the Barley (left) and Munin (right) networks. The solid line corresponds to method **Factorise_VE** and the dotted one to **Prune_VE**.

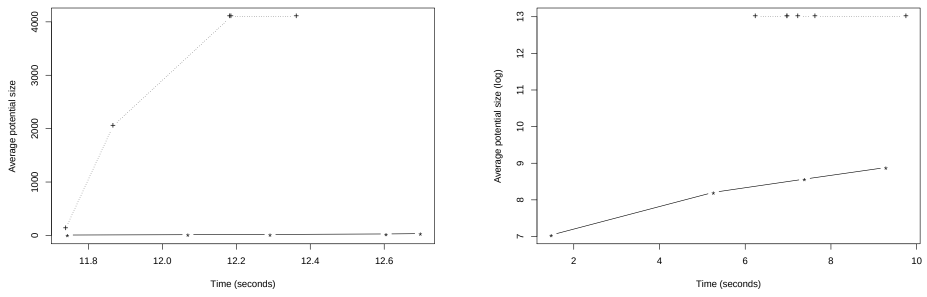

Fig. 13. Average potential size vs. time for the Andes (left) and Water (right) networks. The solid line corresponds to method **Factorise_VE** and the dotted one to **Prune_VE**.

that the decomposition can be carried out even if the tree does not really contain proportional subtrees, in which case the obtained factorisation will be approximate.

In order to deal with the degree of approximation of the possible decompositions of a potential, we have introduced a measure called the *factorisation degree*, that provides a heuristic to rank the variables in the domain of a potential according to the accuracy of the decompositions that they induce. The computation of such measure is fast enough as to be included in probabilistic inference algorithms, where computing time is a crucial issue.

We have analysed the behaviour of the fast factorisation algorithm as a means of controlling the tradeoff between accuracy and complexity. In the networks tested in the experiments, the factorisation degree performed in a similar way as tree pruning.

Possible applications of the concept of fast factorisation go beyond probabilistic inference algorithms. It could be used in algorithms for learning structured repre-

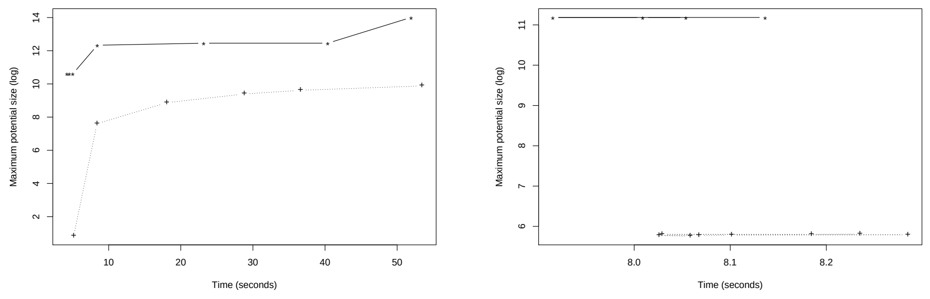

Fig. 14. Maximum potential size (in logarithmic scale) vs. time for the Barley (left) and Munin (right) networks. The solid line corresponds to method **Factorise_VE** and the dotted one to **Prune_VE**.

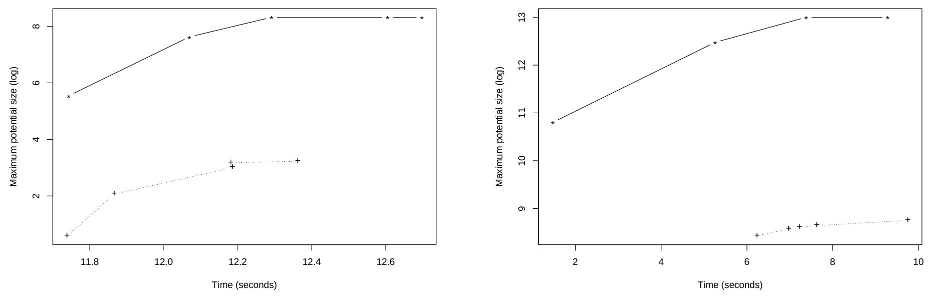

Fig. 15. Maximum potential size (in logarithmic scale) vs. time for the Andes (left) and Water (right) networks. The solid line corresponds to method **Factorise_VE** and the dotted one to **Prune_VE**.

sentations of probabilistic potentials, such as *recursive probability trees*.[30,31]

Acknowledgements

This work has been supported by the Spanish Ministry of Science and Innovation, through projects TIN2010-20900-C04-01,03, by the FPI scholarship programme (BES-2008-002049) and by EFRD (FEDER) funds. A preliminary version of this paper was presented in the SMPS'2010 Conference.[32]

References

1. J. Pearl, Probabilistic reasoning in intelligent systems, Morgan-Kaufmann (San Mateo), 1988.

2. C. Butz, S. Hua, K. Konkel, H. Yao, Join tree propagation with prioritized messages, Networks 55 (2010) 350–359.
3. C. Butz, K. Konkel, P. Lingras, Join tree propagation utilizing both arc reversal and variable elimination, International Journal of Approximate Reasoning 52 (2010) 948–959.
4. C. Butz, A.L. Madsen, K. Williams, Using four cost measures to determine arc reversal orderings, EQSCARU'2011, Lecture Notes in Artificial Intelligence 6717 (2011) 110–121.
5. F. Jensen, S. Lauritzen, K. Olesen, Bayesian updating in causal probabilistic networks by local computation, Computational Statistics Quarterly 4 (1990) 269–282.
6. A.L. Madsen, F. Jensen, Lazy propagation: a junction tree inference algorithm based on lazy evaluation, Artificial Intelligence 113 (1999) 203–245.
7. A.L. Madsen, Improvements to message computation in lazy propagation, International Journal of Approximate Reasoning 51 (2010) 499–514.
8. P. Shenoy, Binary join trees for computing marginals in the Shenoy-Shafer architecture, International Journal of Approximate Reasoning 17 (1997) 239–263.
9. A. Cano, S. Moral, A. Salmerón, Penniless propagation in join trees, International Journal of Intelligent Systems 15 (2000) 1027–1059.
10. A. Cano, S. Moral, A. Salmerón, Lazy evaluation in Penniless propagation over join trees, Networks 39 (2002) 175–185.
11. V. Gogate, R. Dechter, SampleSearch: Importance sampling in presence of determinism, Artificial Intelligence 175 (2011) 694–729.
12. S. Moral, A. Salmerón, Dynamic importance sampling in Bayesian networks based on probability trees, International Journal of Approximate Reasoning 38 (2005) 245–261.
13. A. Salmerón, A. Cano, S. Moral, Importance sampling in Bayesian networks using probability trees, Computational Statistics and Data Analysis 34 (2000) 387–413.
14. G. Cooper, The computational complexity of probabilistic inference using Bayesian belief networks, Artificial Intelligence 42 (1990) 393–405.
15. P. Dagum, M. Luby, Approximating probabilistic inference in Bayesian belief networks is NP-hard, Artificial Intelligence 60 (1993) 141–153.
16. N.L. Zhang, D. Poole, A simple approach to Bayesian network computations. in Proc. of the 10th Canadian Conference on Artificial Intelligence (1994) pp. 171–178.
17. N.L. Zhang, D. Poole, Exploiting causal independence in Bayesian network inference. Journal of Artificial Intelligence Research 5 (1996) 301–328.
18. I. Martínez, S. Moral, C. Rodríguez, A. Salmerón, Approximate factorisation of probability trees, in: ECSQARU'05. Lecture Notes in Artificial Intelligence, Vol. 3571, 2005, pp. 51–62.
19. I. Martínez, C. Rodríguez, A. Salmerón, Dynamic importance sampling in Bayesian networks using factorisation of probability trees, in: Proceedings of the Third European Workshop on Probabilistic Graphical Models, 2006, pp. 187–194.
20. I. Martínez, S. Moral, C. Rodríguez, A. Salmerón, Factorisation of probability trees and its application to inference in Bayesian networks, in: J. Gámez, A. Salmerón (Eds.), Proceedings of the First European Workshop on Probabilistic Graphical Models, 2002, pp. 127–134.
21. C. Boutilier, N. Friedman, M. Goldszmidt, D. Koller, Context-specific independence in Bayesian networks, in: E. Horvitz, F. Jensen (Eds.), Proceedings of the 12th Conference on Uncertainty in Artificial Intelligence, Morgan & Kaufmann, 1996, pp. 115–123.
22. A. Rényi, On measures of entropy and information, In Proc. 4th Berkeley Symp. Math. Stat. and Prob., volume 1, 1961, pp. 547?561.
23. S. Kullback, R. Leibler, On information and sufficiency, Annals of Mathematical

Statistics 22 (1951) 76–86.

24. K. Olesen, U. Kjærulff, F. Jensen, F. Jensen, B. Falck, S. Andreassen, S. Andersen, A munin network for the median nerve - a case study on loops, Applied Artificial Intelligence 3 (1989) 385–404.
25. C. Conati, A.S. Gertner, K. Van Lehn, M.J. Druzdzel, On-line Student Modeling for Coached Problem Solving Using Bayesian Networks, in: Proceedings of the Sixth International Conference on User Modeling (UM–96), Springer-Verlag, 1997, pp. 231–242.
26. K. Kristensen, I.A. Rasmussen, A decision support system for mechanical weed control in malting barley, Computers and Electronics in Agriculture 33 (2002) 197–217.
27. F. V. Jensen, U. Kjærulff, K. G. Olesen, and J. Pedersen, Et forprojekt til et ekspertsystem for drift af spildevandsrensning (An expert system for control of waste water treatment - - A pilot project), Technical Report, Judex Datasystemer A/S, Aalborg, Denmark, 1989 (in Danish).
28. P. Kampstra, Beanplot: A boxplot alternative for visual comparison of distributions, Journal of Statistical Software 28 (2008) 1–9.
29. K.W. Fertig, N.R. Mann, An accurate approximation to the sampling distribution of the studentized extreme-valued statistic, Technometrics 22 (1980) 83–90.
30. A. Cano, M. Gómez-Olmedo, S. Moral, C. Pérez-Ariza, Recursive probability trees for Bayesian networks, in: Proceedings of the XIII Conference of the Spanish Association for Artificial Intelligence (CAEPIA 2010), 2009.
31. A. Cano, M. Gómez-Olmedo, S. Moral, C. Pérez-Ariza, A. Salmerón, Learning recursive probability trees from probabilistic potentials, in: P. Myllymäki, T. Roos, T. Jaakkola (Eds.), Proceedings of the Fifth European Workshop on Probabilistic Graphical Models (PGM-2010), 2010, pp. 49–57.
32. A. Cano, M. Gómez-Olmedo, C. Pérez-Ariza, A. Salmerón, Fast factorization of probability trees and its application to recursive trees learning, in: C. Borgelt, G. González-Rodríguez, M. Lubiano, M. Gil, P. Grzegorzewski, O. Hryniewicz (Eds.), Combining Soft Computing and Statistical Methods in Data Analysis, 2010, pp. 65–72.