# Learning Recursive Probability Trees from Probabilistic Potentials

Andrés Cano, Manuel Gómez-Olmedo, Serafín Moral, Cora B. Pérez-Ariza
Department of Computer Science and Artificial Intelligence
University of Granada, Spain
{acu,mgomez,smc,cora}@decsai.ugr.es

Antonio Salmerón
Department of Statistics and Applied Mathematics
University of Almería, Spain
antonio.salmeron@ual.es

**Abstract**

A recursive probability tree (RPT) is an incipient data structure for representing the distributions in a probabilistic graphical model. RPTs capture most of the types of independencies found in a probability distribution. The explicit representation of these features using RPTs simplifies computations during inference. This paper describes a learning algorithm that builds a RPT from a probability distribution. Experiments prove that this algorithm generates a good approximation of the original distribution, thus making available all the advantages provided by RPTs.

## 1 Introduction

The required size for representing probability distributions in a probabilistic graphical model (like a Bayesian network) is exponential in the number of variables. A Bayesian network is an efficient representation of a joint probability distribution, since it exploits independencies among the variables, but it cannot directly represent *context-specific independencies* (Boutilier et al., 1996) within the distributions. Probability trees have been previously used to represent this kind of independencies within probability potentials (Cano et al., 2000). Moreover, sometimes a probability distribution can be obtained through the multiplication (factorization) of a list of smaller distributions by detecting proportionalities within it (Martínez et al., 2002; Martínez et al., 2005; Martínez et al., 2006). Recently, a new data structure for representing potentials was introduced (Cano et al., 2009): *Recursive Probability Trees (RPTs).* This kind of tree is a generalization of a probability tree. It allows to represent context-specific independencies within distributions, while keeping potentials factorized.

Like probabilistic decision graphs (Jaeger, 2004) and chain event graphs (Smith and Anderson, 2008), RPTs can be used as a standalone representation of joint probability distributions, and probabilistic inference can be fully carried out using this single structure, as the necessary operations, namely product, marginalization and restriction, are well defined over this data structure (Cano et al., 2009).

This paper presents a learning algorithm able to decompose a probability distribution into smaller pieces, by detecting context-specific independencies and multiplicative decompositions. This decomposition will be represented as a RPT. The rest of the paper is organized as follows: Section 2 defines RPTs and describes their features; Section 3 presents the algorithm used for constructing a RPT from a probabilistic potential; Section 4 shows the experiments performed for testing the performance of the algorithm; and finally Section 5 presents conclusions as well as future research directions.

## 2 Recursive Probability Trees

A Recursive Probability Tree (Cano et al., 2009) (hereafter referred to as *RPT*) is a directed tree with two different kinds of inner nodes (Split nodes and List nodes), and two types of leaf nodes (Value nodes and Potential nodes). A Split node represents a discrete variable. A List node represents a multiplicative factorization by listing all the factors to which a potential is decomposed. It contains one outgoing arc for every factor in the decomposition. A Value node represents a non-negative real number. Finally, a Potential node stores a full potential internally using an arbitrary representation. Fig. 1 shows a *RPT* (left part) and the Bayesian network whose joint probability distribution is encoded in the tree (right part). Note how the potentials are enclosed in Potential nodes, and how the List node represents a multiplicative factorization. Using this structure, it is possible to represent context-specific independencies within a probability distribution, as shown in Fig. 2, as well as factorizations (involving the whole potential or parts of it). Proportionalities within the probability distribution are also easily represented using this structure (see Fig. 2).

In (Cano et al., 2009) we give a formal definition of a RPT and a method to obtain the value of the potential for each configuration of its variables.

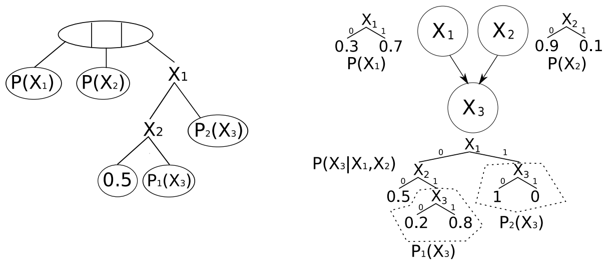

Figure 1: RPT (left) encoding of a Bayesian network distribution (right)

## 3 Constructing a RPT from a probabilistic potential

In this section we shall describe our proposal for transforming any given probabilistic potential (for instance, a probability table) into a RPT. Our proposal is aimed at detecting context specific independencies and multiplicative factorizations present in a probabilistic potential. In order to detect context specific independencies, we follow an approach similar to the procedure used for constructing probability trees (Salmerón et al., 2000), which is based on selecting variables for Split nodes according to their information gain, in a similar way as variables are selected when constructing decision trees (Quinlan, 1986). Regarding the multiplicative decomposition, the basic idea is to make groups of variables using their mutual information as a selection criterion. The groups are later used to obtain the potentials that make up the multiplicative decomposition.

The starting point is a potential $f$ defined over a set of variables $\mathbf{X}$, and the aim of the algorithm is therefore to find a representation of $f$ as a RPT. We denote $S = \sum_{\mathbf{x}} f(\mathbf{x})$ as the sum of all the values in $f$ and $S(Y = y) = \sum_{\mathbf{z}} f(\mathbf{z}, y)$ (sum of values consistent with $Y = y$). The procedure we propose operates over an auxiliary graph structure $G_f$ with vertex set $\mathbf{X}$ where two variables $Y, Z \in \mathbf{X}$ will be linked if there is a probabilistic dependence between them. More precisely, a link $Y - Z$ is present in $G_f$ if the dependence between $Y$ and $Z$ exceeds a given threshold $\varepsilon > 0$. We use the mutual information as a measure of dependence, and hence, a link $Y - Z$ will be included only if

$$I(Y, Z) = \sum_{y,z} p(y, z) \log \frac{p(y, z)}{p(z)p(y)} > \varepsilon, \qquad (1)$$

where $p(y, z) = \frac{f^{\downarrow YZ}(y,z)}{S}$ and $f^{\downarrow YZ}$ is the marginal of $f$ over variables $Y$ and $Z$. Each

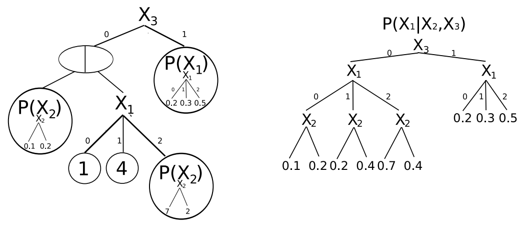

Figure 2: Distribution with context specific independencies and proportional values (right), and the corresponding *RPT* (left)

link $Y - Z$ is weighted with $I(Y, Z)$. After constructing graph $G_f$, a first factorization of $f$ is readily obtained if $G_f$ is disconnected in $n$ connected components $\mathbf{X}_1 \cup \cdots \cup \mathbf{X}_n = \mathbf{X}$. The factorization is given by

$$f(\mathbf{x}) = f_1(\mathbf{x}_1) \cdots f_n(\mathbf{x}_n) S_n, \tag{2}$$

where $f_i = f^{\downarrow \mathbf{X}_i}$, $i = 1, \ldots, n$, and $S_n$ is a normalization factor required to keep the sum of $f_1(\mathbf{x}_1) \cdots f_n(\mathbf{x}_n)$ equal to the sum of $f(\mathbf{x})$:

$$S_n = \frac{\sum_{\mathbf{x}} f(\mathbf{x})}{\sum_{\mathbf{x}} (f_1(\mathbf{x}_1) \cdots f_n(\mathbf{x}_n))}. \tag{3}$$

Hence, potential $f$ can be represented as a RPT where the root node would be a List node containing the factors in Eq. (2). On the contrary, if graph $G_f$ remains as a single connected component, it means that the potential is not decomposable as a list of factors with disjoint variables. However, conditional decompositions are possible. In order to seek for such context specific factorizations we must compute for each variable $Y \in \mathbf{X}$ the following value

$$V(Y) = \sum_{Z \text{ neighbour of } Y} I(Y, Z). \tag{4}$$

Next, we choose $Y_0$ such that $Y_0 = \arg\max_{Y \in \mathbf{X}} V(Y)$. This heuristic removes the variable more dependent on the remaining variables in the graph. This way we can split the graph into several connected components representing independent parts within the potential analyzed.

The procedure described above is repeated, but restricted to variable $Y_0$. That is, we construct a graph $G_f^{Y_0}$ by removing $Y_0$ and its links from $G_f$ and re-weighting each remaining link $Z - U$ in $G_f^{Y_0}$ with

$$I(Z, U|Y_0) = \sum_{z,u,y_0} p(z, u, y_0) \log \frac{p(z, u|y_0)}{p(z|y_0)p(u|y_0)},$$

where

$$p(z, u, y_0) = \frac{f^{\downarrow ZUY_0}(z, u, y_0)}{\sum_{z,u,y_0} f^{\downarrow ZUY_0}(z, u, y_0)}$$

and the conditional distributions $p(z, u|y_0)$, $p(z|y_0)$ and $p(u|y_0)$ are computed from $p(z, u, y_0)$. If the value of the marginal for some configuration of the conditioning variable is equal to zero, then the conditional mutual information reduces to a summation of terms of the form $0 log 0$, and assumed to be zero. Again, we consider a threshold $\varepsilon > 0$ so that only those links $Z - U$ where $I(Z, U \mid Y_0) > \varepsilon$ will be added to the graph, and weighted as $I(Z, U \mid Y_0)$.

As a previous step to the creation of $G_f^{Y_i}$, we must sort $Y_i$ into set $\mathbf{Y}_1$ if the variable was connected to all the other variables in the connected component before being removed; otherwise, the variable will be appended to set $\mathbf{Y}_2$. These two sets will help us to discern between scenarios with context-specific independencies and factorizations.

In general, if $Y_0, \ldots, Y_m$ have been chosen, everything must be turned conditional on $Y_1, \ldots, Y_m$ before selecting $Y_{m+1}$. The process stops when a division of the graph is found or when there is no variable left to choose, i.e. the graph has two variables and both are connected.

In the second case (the graph has only two connected variables) there is no further possible decomposition. In the first case, suppose that after choosing $\mathbf{Y} = \{Y_0, ..., Y_m\}$ the graph is decomposed into $n$ connected components $\mathbf{Z} = \mathbf{Z}_1 \cup \cdots \cup \mathbf{Z}_n$, with $\mathbf{Z} = \mathbf{X} \setminus \mathbf{Y}$.

Next, a RPT is built with the variables in $\mathbf{Y}_1$ (containing Split nodes). For each leaf $h$ of this tree, suppose that $\mathbf{Y}_1 = \mathbf{y}_1$ is the assignment compatible with $h$, then potential $f^h$ is stored in leaf $h$, where $f^h$ is defined as $f^h = f^{R(\mathbf{Y}_1 = \mathbf{y}_1)}$, and $f^{R(\mathbf{Y}_1 = \mathbf{y}_1)}$ denotes the potential $f$ restricted to assignment $(\mathbf{Y}_1 = \mathbf{y}_1)$. Potential $f^h$ is decomposed as

$$f^h(\mathbf{z}_1, \ldots, \mathbf{z}_n, \mathbf{y}_2) := S^h \prod_{i=1}^{n} f^h(\mathbf{z}_i, \mathbf{y}_2),$$

$$S^h = \frac{\sum_{\mathbf{z},\mathbf{y}_2} f^h(\mathbf{z}_1, \ldots, \mathbf{z}_n, \mathbf{y}_2)}{\sum_{\mathbf{z},\mathbf{y}_2} \prod_{i=1}^{n} f^h(\mathbf{z}_i, \mathbf{y}_2)} \tag{5}$$

with $\mathbf{z} = (\mathbf{z}_1, \ldots, \mathbf{z}_n)$. This scenario is explained in Fig. 3. Once a decomposition is performed, the algorithm is recursively applied to each and every potential obtained successively, until no further decomposition can be computed.

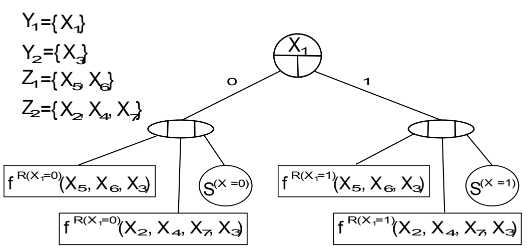
$Y_1=\{X_1\}$
$Y_2=\{X_3\}$
$Z_1=\{X_5, X_6\}$
$Z_2=\{X_2, X_4, X_7\}$
$X_1$
0
1
$f^{R(X_1=0)}(X_5, X_6, X_3)$
$S^{(X=0)}$
$f^{R(X_1=1)}(X_5, X_6, X_3)$
$S^{(X=1)}$
$f^{R(X_1=0)}(X_2, X_4, X_7, X_3)$
$f^{R(X_1=1)}(X_2, X_4, X_7, X_3)$

Figure 3: Example where a division of the graph has been found.

### 3.1 Computing the threshold $\varepsilon$

The value of $\varepsilon$ should not be kept constant throughout the entire algorithm, as the range of possible values of the mutual information varies depending on the variables over which it is computed, and also on the computation of the mutual information conditioned to other variables. Thus, we propose assigning a rate $\delta$, with $0 \leq \delta \leq 1$, and then compute $\varepsilon$ as the fraction of the maximum mutual information determined by $\delta$. In the case of unconditional mutual information between two variables $X$ and $Y$, notice that

$$\begin{aligned} I(X,Y) &= H(X) - H(X|Y) \\ &= H(X) - (H(X,Y) - H(Y)) \\ &= H(X) + H(Y) - H(X,Y) \end{aligned} \tag{6}$$

where $H(\cdot)$ denotes Shannon's entropy. Therefore, as $H(X) \leq \log|X|$, where $|X|$ is the number of possible values of $X$, it follows that
$I(X,Y) \leq \log|X| + \log|Y|$,
and thus, we compute $\varepsilon$ as

$$\varepsilon := \delta \times (\log|X| + \log|Y|). \tag{7}$$

In the case of conditional mutual information, if we have $X, Y$ and $\mathbf{Z}$, it follows that $I(X,Y|\mathbf{Z}) = H(X|\mathbf{Z}) - H(X,Y,\mathbf{Z})$. Since $H(X|\mathbf{Z}) = H(X,\mathbf{Z}) - H(\mathbf{Z})$ and $H(X|Y,\mathbf{Z}) = H(X,Y,\mathbf{Z}) - H(Y,\mathbf{Z})$, it holds that

$$\begin{aligned} I(X,Y|\mathbf{Z}) = H(X,\mathbf{Z}) - H(\mathbf{Z}) - H(X,Y,\mathbf{Z}) + \\ H(Y,\mathbf{Z}) \leq \log|X \cup \mathbf{Z}| + \log|Y \cup \mathbf{Z}| \end{aligned} \tag{8}$$

So, $\varepsilon$ can be computed as:

$$\varepsilon := \delta \times (\log|X \cup \mathbf{Z}| + \log|Y \cup \mathbf{Z}|). \tag{9}$$

### 3.2 Detecting context specific independencies

Besides factorizations, RPTs can efficiently represent context specific independencies, in a similar way as they are represented by probability trees (Boutilier et al., 1996). A possible improvement of the described algorithm would be to widen the search by choosing at some point a variable $Y_j$ connected to the rest of the variables and splitting the tree by it. In this case, the tree would grow with a Split node for such variable, and the process would continue for every branch, but now with the distributions restricted to the branch configuration. This way, the algorithm may follow different paths during factorization for each branch. In (Salmerón et al., 2000), a methodology of variable selection for labeling the internal nodes of probability trees is proposed. This methodology involves the calculation of the information gain of a given variable $Y_j$ according to Eq. 9. Here we propose a similar approach to discern if splitting by a certain variable increases the quality of the learned model.

Let's assume that we are working with a potential $f$ defined for a set of variables $\mathbf{X} = \{X_1, \ldots, X_n\}$. Let's consider any variable $Y \in \mathbf{X}$ and define $\mathbf{Z} = \mathbf{X} \setminus \{Y\}$. The information gain resulting of splitting potential $f$ by variable $Y$ is computed as

$$\begin{aligned} I(Y,f) = S \cdot (\log|Y| - \log S) + \\ \sum_y S(Y=y) \log S(Y=y). \end{aligned} \tag{10}$$

The maximum possible value for $I(Y,f)$ can be obtained using the properties of Shannon's entropy. We define $N_Y = \sum_y S(Y=y)$. Then, it holds that

$$\frac{-1}{N_Y} \sum_{y \in \Omega_Y} S(Y=y) \log S(Y=y) \geq 0,$$

and, therefore

$$-\sum_{y\in\Omega_Y} S(Y=y)\log S(Y=y) \geq 0 \Rightarrow$$
$$\sum_{y\in\Omega_Y} S(Y=y)\log S(Y=y) \leq 0. \quad (11)$$

Hence, replacing (11) in (10), we obtain $I(Y,f) \leq S\cdot(\log|Y| - \log S)$. Using this result, the criterion for choosing a variable to split from a parameter $0 \leq \delta \leq 1$ would be to select a variable $Y$ if $I(Y,f) \geq \delta\cdot S\cdot(\log|Y| - \log S)$.

## 3.3 The algorithm

This section describes the pseudocode of the algorithm outlined above. The algorithm has been decomposed into different modules for the sake of readability. Algorithm 1 is the main body of the procedure, and the others are divided according to the different scenarios that can appear during the learning process. Algorithm 2 describes the required steps if the graph representing a potential is disconnected. In addition, Algorithm 3, gives the details of the process followed by a connected graph. Algorithm 3 is divided into several parts: Algorithm 4 shows what to do in case an Information Gain is detected for a variable. Algorithms 5 and 6 cover the case in which the resulting graph is disconnected within Algorithm 3.

```
Input: A potential $f$
Output: An $RPT$, $recTree$
begin
    Let $G_f$ be the graph built for $f$
    if $G_f$ is connected then
        if there are more than two variables in $G_f$ then
            $recTree$ = dealWithConnectedGraph()
        else
            Set $recTree = PotentialTreeNode(f)$
        end
    else
        $recTree$ = dealWithDisconnectedGraph()
    end
end
```

**Algorithm 1**: Body of potential factorization algorithm

**Example 1.** We shall illustrate the algorithm with a simple example consisting of decomposing a potential $f$ defined over four binary variables $X, Y, Z$, and $W$ with values $f(x,y,z,w) = \{0.03, 0.04, 0.07, 0.06, 0.18, 0.24, 0.42, 0.36, 0.27, 0.36, 0.63, 0.54, 0.12, 0.16, 0.27, 0.24\}$. It can be seen that potential $f$ is decomposable as $f(x,y,z,w) = f_1(x,y)f_2(z,w)$ with $f(x,y) = \{0.1, 0.6, 0.9, 0.4\}$ and $f_2(z,w) = \{0.3, 0.4, 0.7, 0.6\}$. Alg. 1 is called with potential $f$ as argument. The first step is the construction of graph $G_f$. This is achieved by computing the entropy between each pair of variables and inserting the links for which the mutual information is greater than a given threshold. Let's assume we consider a threshold $\varepsilon = 1E-6$, that is, approximately equal to 0. We do this because independent variables can have a slightly positive mutual information due to rounding errors. The mutual information between each pair of variables, computed according to Eq. (1) is $I(X,Y) = 0.1484$, $I(X,Z) =$ 5.607E-17, $I(X,W) =$ 3.886E-17, $I(Y,Z) =$ 7.772E-17, $I(Y,W) = 0$, and $I(W,Z) = 0.0055$. Therefore, graph $G_f$ has only two arcs, $X-Y$ and $W-Z$. Since the graph is disconnected, Alg.2 is called. As the two connected components of $G_f$ only have two variables each, a list node is returned, in which the first factor is the marginal of $f$ over $(X,Y)$, the second is the marginal of $f$ over $(Z,W)$, and the third is a normalizing constant. More precisely, the three factors are: $g_1(x,y) = \{0.2, 1.2, 1.8, 0.8\}$, $g_2(z,w) = \{0.6, 0.8, 1.4, 1.2\}$, and $g_3(x,y,z,w) = 0.25$. It can be proved that $f(x,y,z,w) = g_1(x,y)g_2(z,w)g_3(x,y,z,w)$.

```
Input: A list $C$, which is the list of connected components
       of $G_f$, the potential $f$
Output: A List Tree Node, $L$
begin
    Let $L$ be a List Tree Node;
    for each $C_i$ in $C$ do
        Let $\mathbf{X}_C$ be the variables in $C_i$;
        if $C_i$ contains only 1 or 2 variables then
            Set $recTree = f^{\downarrow \mathbf{X}_{C_i}}$;
        else
            $recTree = PotentialFactorization(f^{\downarrow \mathbf{X}_{C_i}})$
            (Alg.1);
        end
    end
    Add $recTree$ to $L$;
    Let $factor$ be a ValueTreeNode, computed as in
    Eq. (3);
    Add $factor$ to $L$;
end
```

**Algorithm 2**: DealWithDisconnectedGraph()

**Example 2.** Now consider a potential $f$ defined for three binary variables $X, Y$, and $Z$, with values $f(x,y,z) = \{0.3, 0.3, 0.3, 0.3, 0.1, 0.2, 0.75, 0.25\}$. Let's assume we consider a threshold $\varepsilon = 0.001$. Then, Alg. 1 generates a complete graph $G_f$, as the mutual information values are $I(X,Y) = 0.0398$, $I(X,Z) = 0.0122$, and $I(Y,Z) = 0.0212$. Next, Alg. 3 is called with $G_f$ as an argument. This algorithm selects a variable according to the connectivity values defined in Eq.(4). These values are $V(X) = 0.052$, $V(Y) = 0.061$, and $V(Z) = 0.0334$. Therefore, the chosen variable is $Y$. Since $Y$ is connected to the rest of the variables, it is inserted into $\mathbf{Y}_1$ and $G_f^Y$, consisting of a graph with variables $X$ and $Z$, and a link between them, is generated. In the next step, the information gain is computed according to Eq.(10) and, since it is equal to 0.0993, Alg.4 is called. This last procedure constructs a split node with variable $Y$ and two children, one for each possible value of $Y$. The two children are $g_1(x,z) = f^{R(Y=0)}$ and $g_2(x,z) = f^{R(Y=1)}$. Their values are $g_1(x,z) = \{0.3, 0.3, 0.1, 0.2\}$ and $g_2(x,z) = \{0.3, 0.3, 0.75, 0.25\}$.

```
Input: The potential $f$, graph $G$
Output: A Tree Node, $recTree$
begin
    Let $\mathbf{Y}_1$ and $\mathbf{Y}_2$ be empty vectors of variables;
    while $G$ remains connected and $f$ has more than 2
    variables and there is no information gain do
        Choose variable to remove, $Y$ (Eq.4);
        if $Y$ was connected to all other variables in $G_f$
        then
            Add $Y$ to $\mathbf{Y}_1$;
        else
            Add $Y$ to $\mathbf{Y}_2$;
        end
        Build $G_f^Y$;
        if $G_f^Y$ is connected then
            if There is Information Gain then
                $treeNode =$
                $dealWithIndependentSplit()$
            end
        else
            if $\mathbf{Y}_1$ is empty then
                $treeNode = dealWithoutSplitChain$
            else
                $treeNode = dealWithSplitChain$
            end
        end
    end
end
```

**Algorithm 3**: DealWithConnectedGraph()

## 4 Experiments

### 4.1 Learning from a probability table

The first experiment consisted of 30 runs of the algorithm over randomly generated probability tables, defined over 6 binary variables. For each

```
Input: The potential $f$, Vector $\mathbf{Y}_1$
Output: A Split Tree Node, $recTree$
begin
    Let $recTree$ be a Split Chain of variables from $\mathbf{Y}_1$;
    for each possible value $\mathbf{y}_1$ of $Y_1$ do
        $treeNode = potentialFactorization(f^{R(\mathbf{Y}_1=\mathbf{y}_1)})$;
        Update $recTree$'s current leaf with $f^{R(\mathbf{Y}_1=\mathbf{y}_1)}$;
    end
end
```

**Algorithm 4**: DealWithIndependentSplit()

```
Input: A list $C$ which is the list of connected components
       of $G_f$, the potential $f$
Output: A Tree Node, $recTree$
begin
    if $C$ has more than one element then
        Let $recTree$ be a List Tree Node;
        for each $C_i$ of $C$ do
            Set $f_1 = potentialFactorization(f^{\downarrow \mathbf{X}_{C_i}})$;
            Add $f_1$ as children of $recTree$;
        end
        Let $factor$ be a ValueTreeNode, computed as in
        Eq. (3);
        Add $factor$ to $recTree$;
    else
        Set $recTree = PotentialTreeNode(f)$;
    end
end
```

**Algorithm 5**: DealWithoutSplitChain()

```
Input: The potential $f$, Vector $\mathbf{Y}_1$, Vector $\mathbf{Y}_2$, A list $C$
       which is the list of connected components of $G_f$
Output: A Tree Node, $recTree$
begin
    Let $recTree$ be a Split Chain of variables from $\mathbf{Y}_1$;
    for each possible value $\mathbf{y}_1$ of $\mathbf{Y}_1$ do
        Let $f_1$ be $f^{R(\mathbf{Y}_1=\mathbf{y}_1)}$;
        if $C$ has more than one element then
            Let $recTree$ be a List Tree Node;
            for each element $C_i$ of $C$ do
                Set $f^R = f^{\downarrow \mathbf{X}_{C_i \cup Y_2}}$;
                Set $f_1 = potentialFactorization(f^R)$;
                Add $f_1$ as children of $recTree$;
            end
            Let $factor$ be a ValueTreeNode, computed
            as in Eq. (3);
        else
            Set $treeNode = potentialFactorization(f_1)$
        end
        Update $recTree$'s current leaf with $treeNode$;
    end
end
```

**Algorithm 6**: DealWithSplitChain()

run, $\epsilon$ was set between 0 and 0.01 at intervals of 0.001, and the root mean squared error (MSE) between the original probability table and the learned RPT was computed. The results are shown in Fig. 4, where it can be seen that for higher values of $\epsilon$, the approximations obtained are worse. But there is a point where the error suddenly increases, fact that can be used as a stopping criterion when searching for a solution. It was also observed that for higher values of $\epsilon$, the decompositions obtained are more factorized and we get close to a model with a list node containing one factor per variable, which is not accurate and gives high error rates.

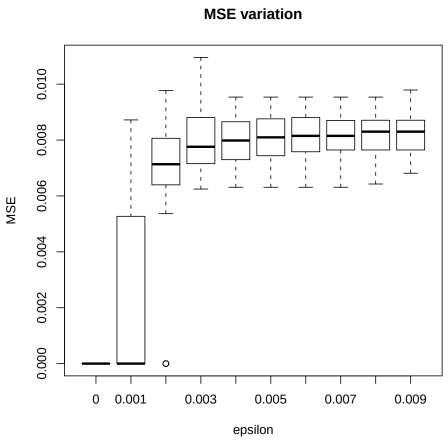

Figure 4: Results for the first experiment

### 4.2 Capturing repeated values

The second experiment consisted of 30 runs of the algorithm over probability tables for 6 binary variables generated from pruned probability trees, in the first case with a prune value (Salmerón et al., 2000) of 0.001 (light pruning) and in the second case, of 0.01 (severe pruning). Pruning the tree consists of replacing subtrees with the average value in the leaves, which means that the value for every configuration of variables in the pruned sub-tree becomes constant. Therefore, if we construct a table from such a tree, as a result all the cells in the table corresponding to the configurations in the pruned sub-tree will contain the same value. Thus, a severe pruning will generate more repeated values in the equivalent probability table than a light pruning. For each probability table generated, the algorithm is applied ten times, corresponding to $\epsilon$ values ranging from 0.0 to 0.01 at intervals of 0.001. Fig. 5 shows the MSE variation for each case. The upper panel of Fig. 5, which corresponds to light prune, shows that higher error values are reached in this case. So, it seems likely that the algorithm is able to detect this kind of regularity within a potential.

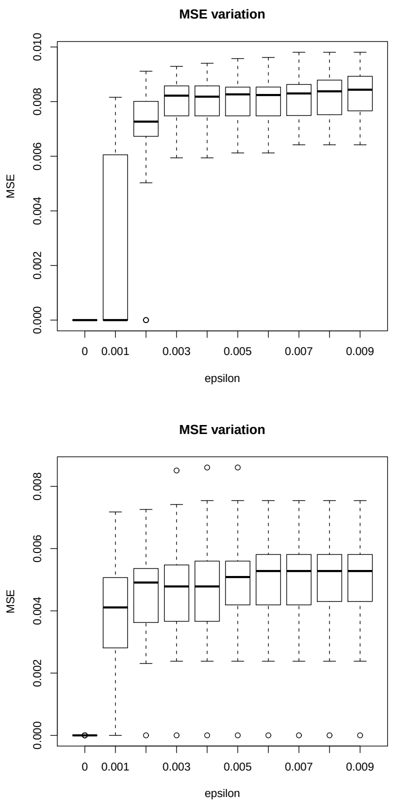

Figure 5: MSE variation learning from the same tree with light and severe prune

Other experiments have been performed to check the relation between accuracy and size of the learned model. The results show that the less accurate representations are those with smaller size. This reflects mainly the fact that smaller representations are expected to be a list of small factors (most likely, one per variable in the distribution), which is a not very accurate representation.

### 4.3 Learning from the same model

Due to the nature of the algorithm and the RPTs themselves, it is possible to obtain different RPTs representing the same distribution. The aim of this experiment was to check if those representations, while different, were accurate. A probability table was generated from a pruned probability tree. The algorithm was applied to it, with an $\epsilon$ value of 0.002, in order to get a slightly different distribution. The resulting RPT was called $RPT_1$. The algorithm used the probability table represented by $RPT_1$ as an argument, this time with an $\epsilon$ value of 0, and returned a new RPT called $RPT_2$. This procedure was repeated 30 times, and then we calculated the mean and the standard deviation of both the difference in sizes of the resultant trees, and the Kullback-Leibler divergence relative to the original distribution. For the tree size differences, we got a mean of 0.73333, and a standard deviation of 1.311312. For the KL divergence, the mean downs to 0.021955 and the standard deviation to 0.040008. These results seem to confirm that under these circumstances, $RPT_1$ and $RPT_2$ are similar representations of the same distribution. In other words, this experiment illustrates the ability of the algorithm to find a RPT representation of a probability distribution, close to the original distribution.

## 5 Conclusions

In this paper we have proposed an algorithm for transforming a probabilistic potential into a RPT. The experiments performed suggest that the proposed algorithm is able to capture most of the details of the original distribution. This proposal can be used as the basis for designing approximate algorithms for inference in Bayesian networks, using RPT-based representations of the potentials involved in the inference process. The applicability of this method is limited, in practice, by the size of the potential that is going to be transformed into a RPT. It can be seen in Eq. (1), where the distributions used are obtained by marginalizing the original potential $f$. Therefore, the availability of a representation of $f$ that allows an efficient computation of marginals would benefit the performance of the algorithm.

We are currently considering the possibility of extending the algorithm in order to learn directly from a database. Also, we are studying a way to detect a different kind of regularity, namely, the proportionality between different parts of the potential.

### Acknowledgments

This research was jointly supported by the Spanish Ministry of Education and Science under projects TIN2007-67418-C03-03,02, the European Regional Development Fund (FEDER), the FPI scholarship programme (BES-2008-002049) and the Andalusian Research Program under project P08-TIC-03717.

## References

C. Boutilier, N. Friedman, M. Goldszmidt, and D. Koller. 1996. Context-specific independence in Bayesian networks. In E. Horvitz and F.V. Jensen, editors, *Proceedings of the 12th Conference on Uncertainty in Artificial Intelligence*, pages 115–123. Morgan & Kaufmann.

A. Cano, S. Moral, and A. Salmerón. 2000. Penniless propagation in join trees. *International Journal of Intelligent Systems*, 15:1027–1059.

A. Cano, M. Gómez-Olmedo, S. Moral, and C.B. Pérez-Ariza. 2009. Recursive probability trees for Bayesian networks. *CAEPIA 2009. Lecture Notes in Artificial Intelligence.*, 5988:242–251.

M. Jaeger. 2004. Probabilistic decision graphs. combining verification and AI techniques for probabilistic inference. *International Journal of Uncertainty, Fuzziness and Knowledge Based Systems*, 12:19–42.

I. Martínez, S. Moral, C. Rodríguez, and A. Salmerón. 2002. Factorisation of probability trees and its application to inference in Bayesian networks. In J.A. Gámez and A. Salmerón, editors, *Proceedings of the First European Workshop on Probabilistic Graphical Models*, pages 127–134.

I. Martínez, S. Moral, C. Rodríguez, and A. Salmerón. 2005. Approximate factorisation of probability trees. *ECSQARU'05. Lecture Notes in Artificial Intelligence*, 3571:51–62.

I. Martínez, C. Rodríguez, and A. Salmerón. 2006. Dynamic importance sampling in Bayesian networks using factorisation of probability trees. In *Proceedings of the Third European Workshop on Probabilistic Graphical Models (PGM'06)*, pages 187–194.

J.R. Quinlan. 1986. Induction of decision trees. *Machine Learning*, 1:81–106.

A. Salmerón, A. Cano, and S. Moral. 2000. Importance sampling in Bayesian networks using probability trees. *Computational Statistics and Data Analysis*, 34:387–413.

J.Q. Smith and P.E. Anderson. 2008. Conditional independence and chain event graphs. *Artificial Intelligence*, 172:42–68.