International Journal of Uncertainty,
Fuzziness and Knowledge-Based Systems
Vol. 24, No. 1 (2016) 59–89

DOI: 10.1142/S0218488516500045

World Scientific
www.worldscientific.com

Using Binary Trees for the Evaluation of Influence Diagrams*

Rafael Cabañas, Manuel Gómez-Olmedo and Andrés Cano

Department of Computer Science and Artificial Intelligence, University of Granada, CITIC-UGR, C/Daniel Saucedo Aranda s/n, Granada, 18071, Spain

*rcabanas@decsai.ugr.es
†mgomez@decsai.ugr.es
‡acu@decsai.ugr.es

Received 14 June 2015
Revised 8 January 2016

This paper proposes the use of binary trees for representing and managing the potentials involved in Influence Diagrams. This kind of tree allows representing context-specific independencies that are finer-grained compared to those encoded using other representations. This enhanced capability can be used to improve the efficiency of the inference algorithms used for Influence Diagrams. Moreover, binary trees allow computing approximate solutions when exact inference is not feasible. In this work we describe how binary trees can be used to perform this approximate evaluation and we compare them with other structures present in the literature.

Keywords: Probabilistic graphical models; influence diagrams; approximate computation; variable elimination; context-specific independencies.

1. Introduction

Influence Diagrams (IDs)[2,3] provide a framework to model decision problems with uncertainty for a single decision maker. The goal of evaluating an ID is to obtain the best option for the decision maker (*optimal policy*) and its utility. The evaluation of IDs modelling complex decision problems becomes unfeasible due to its computational cost. It is thus necessary to use alternative methods for IDs evaluation such as *LIMIDs*,[4] or simulation techniques.[5,6] Other solutions propose using alternative representations for potentials such as *numerical trees* (NTs)[7–9] trying to offer efficient data structures for storing and managing quantitative information (probabilities and utilities). This data structure takes advantage of *context-specific independencies*[10] so that identical values can be grouped into a single one offering

*A preliminary version of this paper was presented at PGM'12 Workshop.[1]

a compact storage. Moreover, when NTs are too large they can be pruned and converted into smaller trees leading to approximate encodings.

Other alternative representations for potentials are *binary trees* (BTs),[11] where the internal nodes always have two children. The objective of this paper is to describe and to test the use of BTs for evaluating IDs. These trees allow representing finer-grained context-specific independencies than NTs, and should lead to more efficient algorithms. In addition, approximate solutions obtained with BTs should be more accurate than those obtained with NTs. In a previous paper,[11] BTs were already used for making inference in Bayesian Networks (BNs). By contrast, here we detail how this data structure can be used for representing and managing the potentials involved in IDs.

In the experimental work, the behaviour of BTs for evaluating IDs is analyzed. For that purpose, BTs, NTs and tables are compared in different aspects (computability, computation time, storage and error level) and using different evaluation algorithms (*Variable Elimination*,[12,13] *Lazy Evaluation*[14,15] and *Symbolic Probabilistic Inference*[16]). BTs are also compared with other approaches for representing potentials such as ADDs and AADDs.[17] However this comparison is performed using *Bayesian networks* (BNs) instead of IDs due to the lack of any proposal for using them when evaluating IDs.

In a preliminary version of this paper,[1] the evaluation of IDs using BTs was already considered, however there are some differences. In the previous approach only utility potentials were approximated while here probabilities are approximated as well. In addition, a more detailed description of the algorithms is given and the experimental work has been improved: here we analyze a larger number of IDs with three different evaluation algorithms.

The paper is organized as follows: Section 2 introduces basic concepts about IDs, *context-specific independencies* and NTs; Sec. 3 describes key issues about BTs and the operations for building and pruning them; Sec. 4 explains how BTs are used during the evaluation; Sec. 5 includes the experimental work; finally Sec. 6 details our conclusions and lines for future work.

2. Preliminaries

2.1. *Influence diagrams*

An ID[2,3] is a directed acyclic graph (DAG) used for representing and evaluating decision problems under uncertainty. An ID contains three types of nodes: *decision nodes* (squares) that correspond with the actions which the decision maker can control; *chance nodes* (ellipses) representing random variables; and *utility nodes* (diamonds) representing the decision maker preferences. Figure 1 shows an example of an ID used for the treatment of gastric NHL disease.[18]

We denote by $\mathcal{U}_C$ the set of chance nodes, by $\mathcal{U}_D$ the set of decision nodes, and by $\mathcal{U}_V$ the set of utility nodes. The decision nodes have a temporal order, $D_1, \ldots, D_n$, and the chance nodes are partitioned into a collection of disjoint sets according to

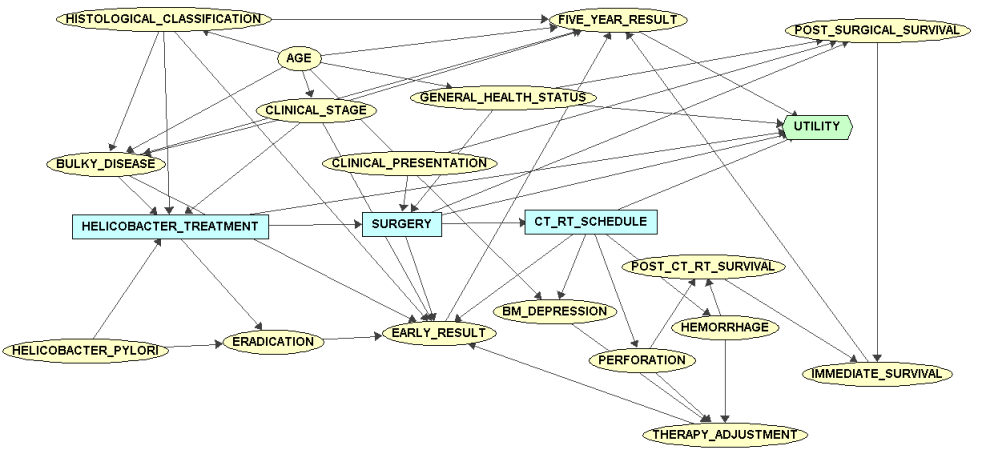


Fig. 1. A real world ID used for the treatment of gastric NHL disease.

when they are observed: $\mathcal{I}_0$ is the set of chance nodes observed before D_1, and $\mathcal{I}_i$ is the set of chance nodes observed after D_i is and before D_{i+1}. Finally, $\mathcal{I}_n$ is the set of chance nodes observed after D_n. That is, there is a partial order:

$$\mathcal{I}_0 \prec D_1 \prec \mathcal{I}_1 \prec \cdots \prec D_n \prec \mathcal{I}_n \,.$$

The *universe* of the ID is $\mathcal{U} = \mathcal{U}_C \cup \mathcal{U}_D = \{X_1, \ldots, X_m\}$. Let us suppose that each variable X_i takes values on a finite set $\Omega_{X_i} = \{x_1, \ldots, x_k\}$. If I is a set of indexes, we shall write $\mathbf{X}_I$ for the set of variables $\{X_i | i \in I\}$, defined on $\Omega_{\mathbf{X}_I} = \times_{i \in I} \Omega_{X_i}$. The elements of Ω_{X_I} are called configurations of $\mathbf{X}_I$ and will be represented as $\mathbf{x}_I$. We denote by $\mathbf{x}_I^{\downarrow \mathbf{X}_J}$ the *projection* of the configuration $\mathbf{x}_I$ onto the set of variables $\mathbf{X}_J$, $\mathbf{X}_J \subseteq \mathbf{X}_I$.

In the present paper we will consider IDs satisfying *non-forgetting assumption*: previous decisions and observations are known at each decision. Arcs that satisfy this condition (*non-forgetting arcs*) are usually assumed implicit to reduce complexity of the graphical display. In this paper we assume non-forgetting arcs to be present. The set of direct predecessors (parents) of a chance or value node X_i are called *conditional predecessors* and denoted $pa(X_i)$. Similarly, the set of direct predecessors of a decision node D_i are designated *informational predecessors* and denoted $pa(D_i)$. That is, the informational predecessors are the variables known to the decision maker when deciding on D_i. Non-forgetting assumption implies that informational predecessors of each decision D_i must include previous decisions and their informational predecessors.

In an ID, each chance node X_i has a conditional probability distribution $P(X_i|pa(X_i))$ associated. Similarly, each utility node V_i has a utility function $U(pa(V_i))$ associated. In general, we will talk about potentials (not necessarily normalized). Let $\mathbf{X}_I$ be the set of all variables involved in a potential, then a *probability potential* denoted by ϕ is a mapping $\phi : \Omega_{\mathbf{X}_I} \to [0, 1]$. A *utility potential*

denoted by ψ is a mapping $\psi : \Omega_{\mathbf{X}_I} \rightarrow \mathbb{R}$. The set of all probability and utility potentials are denoted by Φ and Ψ respectively.

A *policy* δ_{D_i} for a decision D_i is a mapping from past observations and decisions to the possible decision options at D_i such that $\delta_{D_i} : \Omega_{pa(D_i)} \rightarrow \Omega_{D_i}$. A strategy Δ is an ordered set of policies $\Delta = \{\delta_{D_1}, \delta_{D_2} \ldots, \delta_{D_n}\}$, including a policy for each decision variable. When evaluating and ID, we must identify an optimal strategy, denoted $\widehat{\Delta}$, maximizing the expected utility for the decision maker and to compute the *maximum expected utility* $MEU(\widehat{\Delta})$. For each D_i, this optimal strategy includes an *optimal policy* $\widehat{\delta}_{D_i}$, which is a mapping that specifies the best action for the decision maker for each configuration in $\Omega_{pa(D_i)}$.

Optimal Policy and Expected Utility:[13] *Let ID be an influence diagram over the universe* $\mathcal{U} = \mathcal{U}_C \cup \mathcal{U}_D$ *and let* $\mathcal{U}_V$ *the set of utility nodes. Let us suppose that non-forgetting arcs are present in the ID. Let the temporal order of the variables be described as* $\mathcal{I}_0 \prec D_1 \prec \mathcal{I}_1 \prec \cdots \prec D_n \prec \mathcal{I}_n$*. Then:*

(a) *An optimal policy for* D_i *is*

$$\widehat{\delta}_{D_i}(\mathcal{I}_0, D_1, \ldots, \mathcal{I}_{i-1}) = \arg\max_{D_i} \sum_{\mathcal{I}_i} \max_{D_{i+1}} \cdots \max_{D_n} \sum_{\mathcal{I}_n} \prod_{X \in \mathcal{U}_C} P(X|pa(X)) \left(\sum_{V \in \mathcal{U}_V} U(pa(V)) \right). \quad (1)$$

(b) *If the decision maker follows the optimal policy* $\widehat{\delta}_{D_i}$ *(and acts optimally in the future), the expected utility is:*

$$EU(\mathcal{I}_0, D_1, \ldots, \mathcal{I}_{i-1}) = \frac{1}{P(\mathcal{I}_0, \ldots, \mathcal{I}_{i-1}|D_1, \ldots, D_{i-1})} \max_{D_i} \sum_{\mathcal{I}_i} \max_{D_{i+1}} \cdots \max_{D_n} \sum_{\mathcal{I}_n} \prod_{X \in \mathcal{U}_C} P(X|pa(X)) \left(\sum_{V \in \mathcal{U}_V} U(pa(V)) \right) \quad (2)$$

and the optimal strategy $\widehat{\Delta}$ *yields the maximum expected utility:*

$$MEU(\widehat{\Delta}) = \sum_{\mathcal{I}_0} \max_{D_1} \cdots \max_{D_n} \sum_{\mathcal{I}_n} \prod_{X \in \mathcal{U}_C} P(X|pa(X)) \left(\sum_{V \in \mathcal{U}_V} U(pa(V)) \right). \quad (3)$$

2.2. *Context-specific independencies*

The power of IDs lies in the representation of conditional independencies (CIs) which are exploited in order to provide savings in the representation of the joint probability distribution and computational savings during the inference process. However, independencies that only hold for certain contexts cannot be captured with the ID structure, i.e. given a specific assignment of values to some variables. This kind of independence is called *context-specific independence (CSI)*, and was first introduced by Boutilier.[10] Let $\mathbf{X}, \mathbf{Y}, \mathbf{C}, \mathbf{Z}$ be pairwise disjoint sets of variables.

We say $\mathbf{X}$ and $\mathbf{Y}$ are *contextually independent* given $\mathbf{Z}$ and the *context* $\mathbf{C} = \mathbf{c}$, denoted by $I_{\mathbf{c}}(\mathbf{X}; \mathbf{Y}|\mathbf{Z}, \mathbf{c})$ if:

$$P(\mathbf{X}|\mathbf{Z}, \mathbf{c}, \mathbf{Y}) = P(\mathbf{X}|\mathbf{Z}, \mathbf{c}) \tag{4}$$

whenever $P(\mathbf{Z}, \mathbf{c}, \mathbf{Y}) > 0$. Clearly, a CI is a special case of CSI, namely, a CSI becomes a CI when the CSI holds for all $\mathbf{c} \in \Omega_{\mathbf{C}}$. Similarly, CSIs can also exist in utility functions if:

$$U(\mathbf{X}, \mathbf{Z}, \mathbf{c}, \mathbf{Y}) = U(\mathbf{X}, \mathbf{Z}, \mathbf{c})\,. \tag{5}$$

2.3. *Numerical trees*

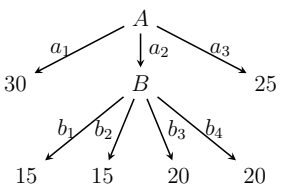
Traditionally, potentials have been represented using tables. However, alternative representations can be used to reduce the storage size and improve the efficiency of the evaluation. For example, *numerical trees* (NTs) have been used to represent potentials in BNs[7,8] and IDs.[9] NTs can be used to encode probability and utility functions, and therefore will be called *numerical probability trees* (NPTs) and *numerical utility trees* (NUTs) respectively. The basic operations for evaluating IDs can be directly performed on NTs.

A NT defined over the set of variables $\mathbf{X}_I$ is a directed tree, where each internal node is labelled with a variable (random or decision), and each leaf node is labelled with a number (a probability or a utility value). We use L_t to denote the *label of node* t. Each internal node has an outgoing arc for each state of the variable associated with that node. Outgoing arcs from node X_i are labelled with the name of a state ($x_i \in \Omega_{X_i}$) of X_i. Calligraphic letters will denote concrete trees. The *size* of a tree $\mathcal{NT}$, denoted *size*($\mathcal{NT}$), is defined as its number of nodes (internal nodes and leaves). A sub-tree of $\mathcal{NT}$ is a *terminal tree of* $\mathcal{NT}$ if it contains one node labelled with a variable and all its children are leaf nodes. Figure 2 shows three different representations for the same utility potential:

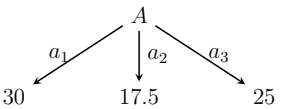
The main advantage of NTs is that they allow the specification of CSIs. For example, the table in Fig. 2 requires 12 numbers for representing the potential. When $A = a_1$, the potential will always take the value 30, regardless of the value of B. Similarly happens when $A = a_3$. In that case, a tree representing the same utility potential requires only 8 nodes. Moreover, a NT can be pruned[7–9] in order

$U(A,B)$	b_1	b_2	b_3	b_4
a_1	30	30	30	30
a_2	15	15	20	20
a_3	25	25	25	25

(*a*)



(*b*)



(*c*)

Fig. 2. Utility potential represented as (a) a table, (b) an exact NUT, and (c) an approximate NUT.

to reduce its storage size and the execution time in propagation algorithms. Thus approximate versions of the potentials will be obtained. The tree approximating the potential (part *c* of Fig. 2) has been obtained by replacing a terminal tree by the average of their leaves. This tree requires only 4 nodes. However, this last representation introduces error during computing. But it must be considered that an exact solution will be unfeasible for some complex IDs.

3. Binary Trees

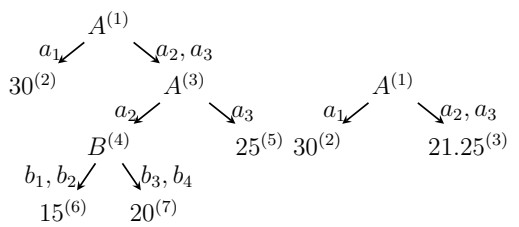
A *binary tree* (BT)[11] is similar to a NT. It is also a directed labelled tree, where each internal node is labelled with a variable (L_t denotes the *label of node* t), and each leaf is labelled with a real number. It also represents a potential for a set of variables $\mathbf{X}_I$. But in this case each internal node has always two outgoing arcs, and as a consequence, a variable can appear more than once labelling the nodes in the path from the root to a leaf node. Another difference is that, for an internal node t labelled with X_i, the outgoing arcs can usually be labelled with more than one state of Ω_{X_i}. We denote by $L_{lb(t)}$ and $L_{rb(t)}$ the labels (two subsets of $\Omega^t_{X_i}$) of the left and right branches of node t. Then, we denote by t_l and t_r the two children of t (t_r for the right child and t_l for the left one). Trees encoding probability potentials will be called *binary probability trees* (BPTs) and those for utility potentials will be called *binary utility trees* (BUTs). Calligraphic letters will denote concrete trees. The *size* of a tree $\mathcal{BT}$, denoted *size($\mathcal{BT}$)*, is defined as its number of nodes (internal nodes and leaves).

An advantage of BTs is that they allow representing CSIs which are finer-grained (also called *contextual-weak independencies*[19]) than those represented using NTs. As an example, Fig. 3 shows the same utility potential presented in Fig. 2 represented as an exact BUT (b) and an approximate BUT (c).

As it happens in the representation of this approximate potential as a NT (see Fig. 2), when $A = a_1$, the potential will always take the value 30, regardless of the value of B. A similar situation happens when $A = a_3$. In these cases the BT has been pruned and only one value has been necessary to represent four configurations. The use of BTs allows pruning more nodes: nodes with values 15 and 20 can also

$U(A,B)$	b_1	b_2	b_3	b_4
a_1	30	30	30	30
a_2	15	15	20	20
a_3	25	25	25	25

(a)



(b) (c)

Fig. 3. Utility potential represented as (a) a table, (b) an exact BUT, and (c) an approximate BUT.

be represented by two nodes in Fig. 3, part (b). Therefore, the exact BUT requires 7 nodes for representing the potential. A sub-tree of $\mathcal{BT}$ is a *terminal tree of* $\mathcal{BT}$ if it contains one node labelled with a variable and its both children are leaf nodes. Terminal trees can be pruned and replaced by the average of their leaves obtaining a tree which approximates the potential using only three, see Fig. 3, part (c).

In order to define the operations with BTs, it is necessary to extend the definition of configuration of a set of variables. An *extended configuration* of a set of variables $\mathbf{X}_I$, denoted by $\mathbf{A}_{\mathbf{X}_I}$ or $\mathbf{A}$, is defined as $\{A_i \subseteq \Omega_{X_i} | X_i \in \mathbf{X}_I\}$. It can also be seen as a mapping assigning to each $X_i \in \mathbf{X}_I$ a subset $A_i \subseteq \Omega_{X_i}$. An extended configuration of a set of variables defines a set of configurations of that set of variables. That is, the set of configurations defined by $\mathbf{A}_{\mathbf{X}_I}$ is denoted by $S_{A_{\mathbf{X}_I}}$ and it is obtained with the Cartesian product of the subsets of sates in $\mathbf{A}_{\mathbf{X}_I}$. For example, let us consider the set of variables $\mathbf{X}_I = \{A, B\}$ whose domains are $\Omega_A = \{a_1, a_2, a_3\}$ and $\Omega_B = \{b_1, b_2, b_3, b_4\}$ respectively. Then, an extended configuration could be $\mathbf{A}_{\mathbf{X}_I} = \{\{a_3\}, \{b_1, b_2\}\}$ which defines the set of configurations $S_{A_{\mathbf{X}_I}} = \{\{a_3, b_1\}, \{a_3, b_2\}\}$.

Given a node t in a BT defined over the set of variables $\mathbf{X}_I$, the set of its *ancestors* is denoted by $\mathbf{X}_I^t$ and the set of *available states* of X_i at t is denoted by $\Omega_{X_i}^t$. If $X_i \in \mathbf{X}_I^t$, then $\Omega_{X_i}^t$ is the set of states labelling the outgoing branch of X_i in its last occurrence in the path from the root to t. Otherwise, $\Omega_{X_i}^t$ is equal to Ω_{X_i}. For example, in Fig. 3 part (b), the set of available states of A at the node (3) is $\{a_2, a_3\}$. The *associated extended configuration* for t, denoted by $\mathbf{A}^t$, is defined as $\{A_i \subseteq \Omega_{X_i}^t | X_i \in \mathbf{X}_I\}$. For example, the associated extended configuration for the node labelled with the value 15 in Fig. 3(b) is $\{\{a_2\}, \{b_1, b_2\}\}$.

3.1. *Building a BUT*

The method for building a BPT for a probability potential was described in a previous work.[11] It was inspired by the methods for learning classification trees from a set of examples.[20] Herein we describe how to extend such method for building a BUT given a utility potential ψ. It must be noticed that, in general, several BUTs can be built to represent the same potential. However, we are interested in finding the smaller one. Therefore, the task of building a BUT from a table can be seen as an optimization problem interested in choosing the labels for internal nodes (variables) and arcs (states). This requires an heuristic procedure for ordering the variables according to their information. The most informative variables will be located at the highest nodes of the tree. The motivation for such ordering is to obtain leaf nodes as similar as possible. This condition will minimize the error produced when pruning the trees collapsing several leaves into a single one.

The algorithm proposed here builds a BUT using a top-down approach, choosing at each step a variable and a partition of its states. The general scheme for building a BUT representing a utility potential ψ defined over the set of variables $\mathbf{X}_I$ is:

(1) Build an initial $\mathcal{BUT}$ with a single node labelled with the average of the values in the potential: $L_t = \sum_{\mathbf{x}_I \in \Omega_{\mathbf{X}_I}} \psi(\mathbf{x}_I)/|\Omega_{\mathbf{x}_I}|$

(2) While $\mathcal{BUT}$ contains any leaf node t that can be expanded:
   (a) Select a leaf node t.
   (b) According to any criteria, select a variable X_i and partition its available states at t into two subsets, $\Omega^{t_l}_{X_i}$ and $\Omega^{t_r}_{X_i}$.
   (c) Expand the leaf node t (with t rooting the terminal tree and being t_l and t_r its children).
   (d) Label t with X_i and the two outgoing arcs with $\Omega^{t_l}_{X_i}$ and $\Omega^{t_r}_{X_i}$. The two leaf nodes t_l and t_r will be labelled with the average of values consistent with the states labelling the path from the root to t_l and t_r respectively.

(3) Return $\mathcal{BUT}$

The process begins with an initial $\mathcal{BUT}_0$ which has only one node labelled with the average of the values in the potential (step 1). Then, a greedy algorithm (step 2) is applied until an exact BUT is obtained (there is not any leaf node that can be expanded). At each iteration, a new $\mathcal{BUT}_{j+1}$ is generated from the previous one, $\mathcal{BUT}_j$. This new tree is the result of expanding one of the leaf nodes t in $\mathcal{BUT}_j$ with a terminal tree (with t rooting the terminal tree and being t_l and t_r its children). The label of the node t will be replaced with one of the *candidate variables*. A variable X_i is candidate if $\mathbf{A}^t$ contains more than one state for X_i. The set of available states $\Omega^t_{X_i}$ of the chosen candidate variable X_i will be partitioned into two subsets, $\Omega^{t_l}_{X_i}$ and $\Omega^{t_r}_{X_i}$. Each subset labels one of the two outgoing arcs (left and right) of t. The two leaf nodes t_l and t_r in the new terminal tree will be labelled with the average of values consistent with the states labelling the path from the root to t_l and t_r respectively.

The candidate variable and the partition of its states must be chosen using any criteria or heuristic (step 2(b)). Any partition of $\Omega^t_{X_i}$ would be possible, but checking all of them would be a very time-consuming task. In order to reduce the complexity, we assume that the set of available states for X_i at node t is ordered and we only check partitions into subsets with consecutive states. For example, given a variable X with $\Omega^t_X = \{x_1, x_2, x_3\}$, we will only check the partitions $\{\{x_1\}, \{x_2, x_3\}\}$ and $\{\{x_1, x_2\}, \{x_3\}\}$. In our approach we propose choosing the variable and partition that maximizes the *information gain*, defined in the following paragraph.

Information Gain: *Let ψ be the utility potential to be represented as a tree $\mathcal{BUT}_j$ and $\mathcal{BUT}_{j+1}(t, X_i, \Omega^{t_r}_{X_i}, \Omega^{t_l}_{X_i})$ the tree resulting of expanding the leaf node t with the candidate variable X_i and a partition of its available states into sets $\Omega^{t_l}_{X_i}$ and $\Omega^{t_r}_{X_i}$. Let $D(\psi, \mathcal{BUT}_j)$ be the distance between a potential and a tree. The information gain can be defined as:*

$$I(t, X_i, \Omega^{t_l}_{X_i}, \Omega^{t_r}_{X_i}) = D(\psi, \mathcal{BUT}_j) - D(\psi, \mathcal{BUT}_{j+1}(t, X_i, \Omega^{t_l}_{X_i}, \Omega^{t_l}_{X_i}))\,. \qquad (6)$$

For computing the information gain, we need to use a distance measure between a potential and a tree. We propose using the Euclidean distance (Eq. (7)). We performed some tests with other distances and divergences and the best results

were obtained with the Euclidean distance.

$$D(\psi(\mathbf{X}_I), \mathcal{BUT}(\mathbf{X}_I)) = \sqrt{\sum_{\mathbf{x}_I \in \Omega_{\mathbf{X}_I}} (\psi(\mathbf{x}_I) - \mathcal{BUT}(\mathbf{x}_I))^2} \tag{7}$$

where $\mathcal{BUT}(\mathbf{x}_I)$ is the value that the tree assigns to the configuration $\mathbf{x}_I$. A leaf node t can be expanded if there exists any candidate variable and partition of its states such that the information gain is greater than 0. That is, the values of ψ consistent with $\mathbf{A}^t$ are not equal.

Some of the computations performed to calculate the information gain when a node is expanded are also performed in posterior iterations when the children are expanded. Proposition 1 shows an alternative expression for computing the information gain that takes advantage of these repeated computations.

Proposition 1. *Let ψ be a utility potential with n values $v_1, v_2, \ldots, v_n$ represented by a tree $\mathcal{BUT}_j$ with a single leaf labelled with the mean $\sum_{i=1}^n v_i/n$. The result of expanding the leaf node with the candidate variable X_i and a partition of its available states into sets $\Omega_{X_i}^{t_l}$ and $\Omega_{X_i}^{t_r}$ node is a terminal tree. The utility values consistent with $\Omega_{X_i}^{t_l}$ are $vl_1, vl_2, \ldots, vl_{n_l}$ and those consistent with $\Omega_{X_i}^{t_r}$ are $vr_1, vr_2, \ldots, vr_{n_r}$. The left child is labelled with $\sum_{i=1}^{n_l} vl_i/n_l$ and the right child is labelled $\sum_{i=1}^{n_r} vr_i/n_r$. Then, the information gain (Eq. (6)) can be calculated in the following way:*

$$I(t, X_i, \Omega_{X_i}^{t_l}, \Omega_{X_i}^{t_r}) = \sqrt{\sum_{i=1}^{n} v_i^2 - \frac{\left(\sum_{i=1}^{n} v_i\right)^2}{n}} - \sqrt{\sum_{i=1}^{n_l} vl_i^2 - \frac{\left(\sum_{i=1}^{n_l} vl_i\right)^2}{n_l} + \sum_{i=1}^{n_r} vr_i^2 - \frac{\left(\sum_{i=1}^{n_r} vr_i\right)^2}{n_r}}\,. \tag{8}$$

It should be noticed that computing the information gain with this expression is more efficient than using Eq. (6), since the first term can be obtained from the information gain computed when the father node was expanded. The proof for this proposition is given in the Appendix.

In order to illustrate the process for building a BUT, let us consider the potential shown in Fig. 3. The intermediate trees obtained during the building process of this tree are shown in Fig. 4.

The initial tree $\mathcal{BUT}_0$ contains a single node labelled with the value 23.75, which is the average of all the values in the potential. Then, the information gain for each candidate variable and partition is calculated as follows:

$$\begin{aligned} I(t, A, \{a_1\}, \{a_2, a_3\}) &= 6.758 \\ I(t, A, \{a_1, a_2\}, \{a_3\}) &= 0.113 \\ I(t, B, \{b_1\}, \{b_2, b_3, b_4\}) &= 0.075 \\ I(t, B, \{b_1, b_2\}, \{b_3, b_4\}) &= 0.227 \\ I(t, B, \{b_1, b_2, b_3, \}, \{b_4\}) &= 0.075\,. \end{aligned}$$

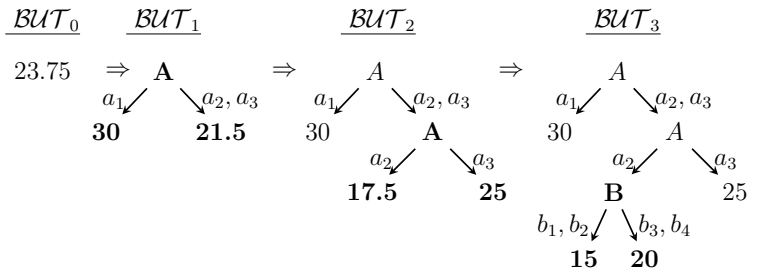


Fig. 4. Process for building a BUT from a potential in Fig. 3.

The highest value for the information gain is obtained if the variable A and the partition $\{\{a_1\}, \{a_2, a_3\}\}$ are chosen to expand the node. After that, the algorithm repeats the same process until the exact BUT is obtained. The node labelled with 30 at $\mathcal{BUT}_1$ will not be expanded any more: all configurations in the utility potential consistent with $A = a_1$ are equal to 30.

3.2. *Pruning a BUT*

Sometimes the size of a BUT can be too large making unfeasible its management during the propagation. In that case, it can be pruned in order to get an approximate one. Pruning a BUT consists of replacing a terminal tree by the average value of its leaves. We apply the following heuristic procedure to decide when to prune a terminal tree. This procedure is guided by a threshold ε.

Pruning a terminal tree. *Let $\mathcal{BUT}$ be a binary utility tree encoding a utility potential ψ, t the root of a terminal tree of $\mathcal{BUT}$ labelled with X_i, t_l and t_r its children, $\Omega_{X_i}^{t_l}$ and $\Omega_{X_i}^{t_r}$ the sets of states for the left and right child respectively,* $\max(\psi)$ *and* $\min(\psi)$ *the maximum and minimum values in ψ, and ε a given threshold such that $\varepsilon \geq 0$. Then, the terminal tree can be pruned if:*

$$I(t, X_i, \Omega_{X_i}^{t_l}, \Omega_{X_i}^{t_r}) \leq \varepsilon \cdot (\max(\psi) - \min(\psi)) \,. \tag{9}$$

In previous definition, I can be locally computed using Eq. (8). The goal of pruning involves detecting leaves that can be replaced by one value without a great increment in the euclidean distance between $\mathcal{BUT}$ and ψ. Here, I is considered as the *information loss* produced in the current binary tree rooted by t is pruned. The pruning process would finish when there are not more terminal trees in $\mathcal{BUT}$ verifying condition in Eq. (9). As utility potentials are not normalized, the highest information loss allowed when pruning depends on a threshold $\epsilon \geq 0$ but also on the maximum and minimum values in ψ. Low values of ε will produce large trees with low errors while low values of ε will lead to small trees and big errors. If $\epsilon = 0$, there is not approximation: a terminal tree will be pruned only if both children have the same value. Figure 5 shows an example of pruning, where the node consistent with the configuration $A = \{a_2\}$ in the left tree has been replaced by the average value of its children in the right tree.

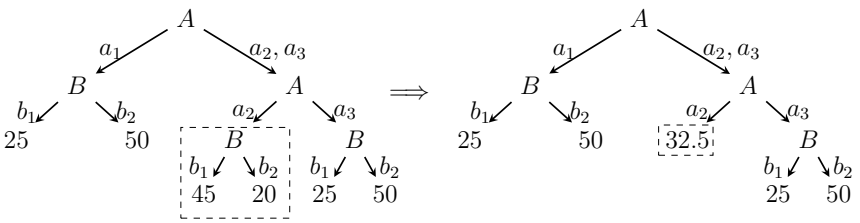


Fig. 5. Example of pruning a terminal node in a BT.

4. ID Evaluation

4.1. *Operations for ID evaluation*

Evaluating an ID requires performing several operations with the potentials. Herein we describe how these operations can be performed directly on BTs. In particular, evaluation algorithms require multiple types of combinations (*multiplication, division, addition* and *maximum*), multiple types of marginalizations (*sum-marginalization* and *max-marginalization*) and a *restriction* operation. Next, we define two generic operations that allow to define all the types of combinations and marginalizations:

- Let $\mathcal{BT}_1$ and $\mathcal{BT}_2$ be two binary trees representing two (probability or utility) potentials ϕ_1 and ϕ_2 defined on the sets of variables $\mathbf{X}_I$ and $\mathbf{X}_J$ respectively. Their generic combination $\mathcal{BT}_1 \otimes \mathcal{BT}_2$ is another binary tree over $\mathbf{X}_I \cup \mathbf{X}_J$ such as for each configuration $\mathbf{x} \in \Omega_{\mathbf{X}_I \cup \mathbf{X}_J}$, $(\mathcal{BT}_1 \otimes \mathcal{BT}_2)(\mathbf{x}) = f(\phi_1(\mathbf{x}^{\downarrow \mathbf{X}_I}), \phi_2(\mathbf{x}^{\downarrow \mathbf{X}_J}))$, where f is a function from $\mathbb{R}^2$ on $\mathbb{R}$ that depends on the particular operation we are carrying out.
- Let $\mathcal{BT}$ be a binary tree representing a (probability or utility) potential defined on the set of variables $\mathbf{X}_I$ and a variable $Y \in \mathbf{X}_I$ where $\Omega_Y = \{y_1, y_2, \ldots y_n\}$. The marginalization of Y out of $\mathcal{BT}$, denoted $\mathsf{M}_Y \mathcal{BT}$, is another binary tree defined over $\mathbf{X}_I \backslash \{Y\}$ such as for each configuration $\mathbf{x} \in \Omega_{\mathbf{X}_I \backslash \{Y\}}$, $(\mathsf{M}_Y \mathcal{BT})(\mathbf{x}) = g(\phi(\mathbf{x}, y_1), \phi(\mathbf{x}, y_2), \ldots, \phi(\mathbf{x}, y_n))$ where g is a function from $\mathbb{R}^n$ on $\mathbb{R}$ that depends on the particular operation we are carrying out.

Each operation with potentials required for the evaluation of an ID can be now defined as a particularization of $\otimes$ and M by specifying the function f or g. Table 1 shows these functions for each particular combination of two binary trees $\mathcal{BT}_1$ and $\mathcal{BT}_1$ whose root nodes are t_1 and t_2. Similarly, Table 2 shows the functions g for carrying out each each particular marginalization of a variable Y out of a binary tree $\mathcal{BT}$ whose root node is t.

Algorithm 1 explains how to perform the *generic combination* operation $\otimes$ of two binary trees $\mathcal{BT}_1$ and $\mathcal{BT}_2$. The inputs for the algorithm are t_1 and t_2, the root nodes of both trees. If t_1 is not a leaf node (lines 13 to 18) a new node with the same label t_1 and the same labels $L_{lb(t_1)}$ and $L_{rb(t_1)}$ for the branches is built.

Table 1. Particularizations of the generic combination operation $\otimes$ and their corresponding functions f. For the division, convention $0/0 = 0$ is adopted.

Operation	Notation	f
$multiplication(t_1, t_2)$	$\mathcal{BT}_1 \cdot \mathcal{BT}_2$	$\phi_1(\mathbf{x}^{\downarrow \mathbf{X}_I}) \cdot \phi_2(\mathbf{x}^{\downarrow \mathbf{X}_J})$
$addition(t_1, t_2)$	$\mathcal{BT}_1 + \mathcal{BT}_2$	$\phi_1(\mathbf{x}^{\downarrow \mathbf{X}_I}) + \phi_2(\mathbf{x}^{\downarrow \mathbf{X}_J})$
$division(t_1, t_2)$	$\mathcal{BT}_1 / \mathcal{BT}_2$	$\phi_1(\mathbf{x}^{\downarrow \mathbf{X}_I}) / \phi_2(\mathbf{x}^{\downarrow \mathbf{X}_J})$
$maximum(t_1, t_2)$	$maximum(\mathcal{BT}_1, \mathcal{BT}_2)$	$\max(\phi_1(\mathbf{x}^{\downarrow \mathbf{X}_I}), \phi_2(\mathbf{x}^{\downarrow \mathbf{X}_J}))$

Table 2. Particularizations of the generic marginalization operation $\Mmarg$ and their corresponding functions g.

Operation	Notation	g
$sum\text{-}marginalization(t, Y, \lvert\Omega_Y\rvert)$	$\sum_Y \mathcal{BT}$	$\phi(\mathbf{x}, y_1) + \phi(\mathbf{x}, y_2) + \cdots + \phi(\mathbf{x}, y_n)$
$max\text{-}marginalization(t, Y)$	$\max_Y \mathcal{BT}$	$\max(\phi(\mathbf{x}, y_1), \phi(\mathbf{x}, y_2), \ldots, \phi(\mathbf{x}, y_n))$

Algorithm 1: Generic Combination algorithm

input : t_1 and t_2 (root nodes of $\mathcal{BT}_1$ and $\mathcal{BT}_2$);
output: the root of $\mathcal{BT} = \mathcal{BT}_1 \otimes \mathcal{BT}_2$

Build a new node t_n
if t_1 *is a leaf node* **then**
  if t_2 *is a leaf node* **then**
    // Sets the label of the leaf depending on the operation
    $L_{t_n} = f(L_{t_1}, L_{t_2})$
  else
    $L_{t_n} = L_{t_2}$ // Sets the label of t_n
    $L_{lb(t_n)} = L_{lb(t_2)}$ // Sets labels for both branches
    $L_{rb(t_n)} = L_{rb(t_2)}$
    t_{nl}=combination(t_1, t_{2_l}) // Sets children
    t_{nr}=combination(t_1, t_{2_r})
else
  Let X_i be the variable labelling t_1
  $L_{t_n} = L_{t_1}$ // Sets the label of t_n
  $L_{lb(t_n)} = L_{lb(t_1)}$ // Sets the labels of both branches
  $L_{rb(t_n)} = L_{rb(t1)}$
  t_{nl}=combination$(t_{1_l}, \mathcal{BT}_2^{R(X_i, L_{lb(t_1)})})$ // Sets children
  t_{nr} =combination$(t_{1_r}, \mathcal{BT}_2^{R(X_i, L_{rb(t_1)})})$
return t_n

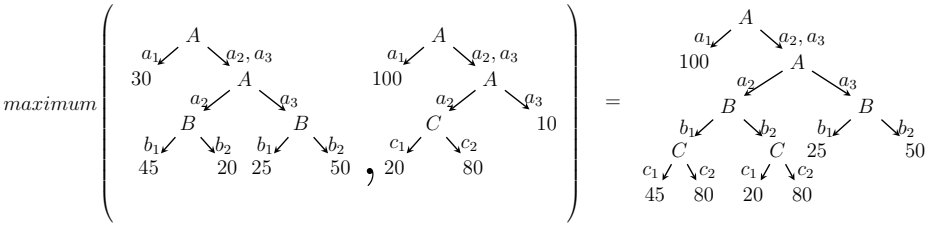


Fig. 6. Maximum operation between two BUTs.

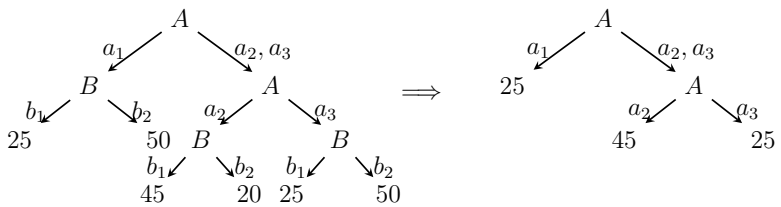


Fig. 7. Restriction of a BUT to the configuration $\{B = b_1\}$.

The left child of the new tree is the result of the generic combination operation between the left child of t_1 and the restriction of t_2 to the states of $L_{lb(t_1)}$ (the restriction operation is later explained). Similarly, the right child of the new tree is the result of the generic combination operation between the right child of t_1 and the restriction of t_2 to the states of $L_{rb(t_1)}$. If t_1 is a leaf node but t_2 is not (lines 7 to 11), the tree $\mathcal{BT}_2$ is traversed until both nodes are leaf nodes. In that case (line 5), a new node labelled with the result of applying the corresponding function f is built. Figure 6 shows an example of the maximum operation between two BUTs. The result of applying any other combination operation $\otimes$ to these BTs is another BT with the same structure but with different values in their leaves.

The *restriction* operation of a given $\mathcal{BT}$ to a set of states S_{X_j} is denoted by $\mathcal{BT}^{R(X_j,S_{X_j})}$. In lines 17 and 18 of Algorithm 1 this operation is used to calculate ${\mathcal{BT}_2}^{R(X_i,L_{lb(t_1)})}$ and ${\mathcal{BT}_2}^{R(X_i,L_{rb(t_1)})}$. The restriction of a potential to a given configuration $\mathbf{x}_J$ consists of returning the part of the potential that is consistent with the configuration. If $\mathcal{BT}$ is a binary tree defined on $\mathbf{X}_I$, and $\mathbf{x}_J$ a configuration for $\mathbf{X}_J$, $\mathbf{X}_J \subseteq \mathbf{X}_I$, then $\mathcal{BT}^{R(\mathbf{x}_J)}$ will denote the tree restricted to those configurations compatible with $\mathbf{x}_J$. For binary trees, we need to extend the definition of this operation, to an extended configuration $A_{\mathbf{X}_J}$, which means returning the part of the tree consistent with the configurations in the set $S_{A_{\mathbf{X}_I}}$. Algorithm 2 describes the restriction of a $\mathcal{BT}$ to a set of states S_{X_j} ($S_{X_j} \subseteq \Omega_{X_j}$) of a variable X_j. As an example, Fig. 7 shows the restriction of a BUT to the configuration $\{B = b_1\}$. The restriction of a binary tree to an extended configuration $A_{\mathbf{X}_J}$ can be performed by repeating Algorithm 2 for each one of the variables in $\mathbf{X}_J$.

Algorithm 2: Restriction

input : t (root node of $\mathcal{BT}$); X_j (variable to restrict); S_{X_j} (set of states of X_j to restrict)
output: The root of $\mathcal{BT}^{R(S_{X_j})}$

1 **if** *t is not a leaf node* **then**
2 **if** $L_t == X_j$ **then**
3 Set $S^l_{X_j} = L_{lb(t)} \cap S_{X_j}$ and $S^r_{X_j} = L_{rb(t)} \cap S_{X_j}$;
4 **if** $S^l_{X_j} == \emptyset$ **then**
5 **return** *Restriction*$(t_r, X_j, S^r_{X_j})$
6 **end**
7 **else if** $S^r_{X_j} == \emptyset$ **then**
8 **return** *Restriction*$(t_l, X_j, S^l_{X_j})$
9 **end**
10 **else**
11 $L_{lb(t)} = S^l_{X_j}$ //Sets the labels of both branches
12 $L_{rb(t)} = S^r_{X_j}$
13 $t_l = Restriction(t_l, X_j, S^l_{X_j})$ //the new left child of t;
14 $t_r = Restriction(t_r, X_j, S^r_{X_j})$ //the new right child of t
15 **end**
16 **end**
17 **else**
18 $t_l = Restriction(t_l, X_j, S_{X_j})$ // Sets children;
19 $t_r = Restriction(t_r, X_j, S_{X_j})$
20 **end**
21 **end**
22 **return** t

Algorithm 3 describes the generic marginalization M of a variable Y out of a binary tree $\mathcal{BT}$. This algorithm must be called using $|\Omega_Y|$ as the input parameter k. In recursive calls to the algorithm, k will be set to the number of available states of Y at current node of the tree. This algorithm is recursively executed until a node labelled with the variable to be removed is found. When it happens (lines 12 to 17), the algorithm marginalizes the left and right children trees and combines both them. The type of this combination depends on the type of marginalization: addition is used when sum-marginalizing while maximum is used when max-marginalizing. If the variable L_t labelling the current node t differs from the variable to be removed Y (lines 19 to 24), a new node labelled with L_t is built. Their children are then marginalized. When a leaf node is reached (lines 3 to 8), the new leaf node is returned whose value depends on the specific type of marginalization: for the max-marginalization this node takes the value L_t and for the sum-marginalization takes

Algorithm 3: Generic Marginalization algorithm

input : t (root node of $\mathcal{BT}$); Y (variable to remove); k (a factor for multiplying the labels of leaf nodes)

; **output**: the root of $\mathsf{M}_Y\mathcal{BT}$
if *t is a leaf node* **then**
  Build a new node t_n
  // Sets the label of t_n
  if *sum-marginalization* **then**
    $L_{t_n} = L_t \cdot k$
  else
    $L_{t_n} = L_t$ //max-marginalization
else
  // If the variable of the node is the one to remove
  if $L_t == Y$ **then**
    t_1 = marginalization(t_l, Y, $|L_{lb(t_n)}|$) // Make recursive calls
    t_2 = marginalization(t_r, Y, $|L_{lr(t_n)}|$)
    if *sum-marginalization* **then**
      $t_n = addition(t_1, t_2)$
    else
      $t_n = maximum(t_1, t_2)$
  else
    Build a new node t_n
    $L_{t_n} = L_t$ // Sets the label of the new node
    $L_{lb(t_n)} = L_{lb(t)}$ // Sets the label of both branches
    $L_{lr(tn)} = L_{lr(t)}$
    t_{nl}=marginalization(t_l, Y, k) // Make recursive calls on children
    t_{nr}=marginalization(t_r, Y, k)
return t_n

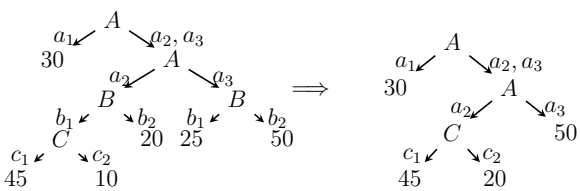


Fig. 8. Max-marginalization of a BUT with respect to variable B.

the value $L_t \cdot k$. Figure 8 shows the application of this operation to a BUT in order to remove variable B.

4.2. *Complexity analysis*

The complexity of the operations on potentials depends on the size of the structure used to represent them. That is, the number of values in a table or the number of nodes in the completely expanded tree (internal nodes and leaves).

Let $\phi(X_1, X_2, \dots X_n)$ be a potential with n variables. Assuming that all the variables have the same number of states, i.e. $k = |\Omega_{X_1}| = |\Omega_{X_2}| = \cdots = |\Omega_{X_n}|$, the size of a table $\mathcal{T}$, representing this potential is given by Eq. (10), which corresponds with the size of the Cartesian product of the variable domains:

$$size(\mathcal{T}) = |\Omega_{X_1}| \cdot |\Omega_{X_2}| \cdot \ldots \cdot |\Omega_{X_n}| = k^n \,. \tag{10}$$

In the level 0 of a $\mathcal{NT}$ representing ϕ there is only one node (the root). In a level i with $i = 1, \dots, n$, there are k^i nodes. Then, the size of $\mathcal{NT}$:

$$size(\mathcal{NT}) = 1 + \sum_{i=1}^{n} k^i = \frac{k^{n+1} - 1}{k - 1} \,. \tag{11}$$

A $\mathcal{BT}$ representing ϕ has k^n leaves (number of values in ϕ) and k^n internal nodes. Then, the size of $\mathcal{BT}$ is:

$$size(\mathcal{BT}) = 2 \cdot k^n - 1 \,. \tag{12}$$

It can be observed that in general the size of BTs is higher than NTs or tables but the complexity of traversing these representations is similar: exponential $\mathcal{O}(k^n)$. However, as BTs allow a more compact representation encoding more context-specific independencies, it is expected a certain benefit when pruning operations are performed. This point is proved empirically in Sec. 5.

4.3. *ID evaluation algorithms with BTs*

Inference algorithms for IDs can be easily adapted for working with BTs. The global structure of the algorithms is not changed: the main difference is that they require an initialization phase where initial BTs are built from tables and pruned in order to obtain smaller trees. BUTs representing utility potentials are built and pruned using the procedures explained in Secs 3.1 and 3.2 respectively. Procedures for building and pruning BPTs representing probability potentials are explained in a previous work.[11] Once all the potentials are transformed into BTs and pruned, the evaluation algorithms are quite similar to the algorithms that use tables: computation is done using operations for BTs (Sec. 4.1), instead of their counterparts for tables. Herein three ID evaluation algorithms from the literature are described for working with BTs: *Variable Elimination*,[12,13] *Lazy Evaluation*[14,15] and *Symbolic Probabilistic Inference*.[16]

4.3.1. *Variable elimination*

The *Variable Elimination* algorithm (VE)[13] is one of the most common algorithms used for evaluating IDs. It has many similarities with the corresponding method for

BNs:[12] it starts with a set of potentials and it eliminates all the variables one by one. There are however some differences compared to the VE algorithm for BNs. First, all the variables must be removed in reverse order of information precedence given by $\prec$. Secondly, chance variables are removed using *sum-marginalization* whereas for decisions *max-marginalization* is used. That is, it first sum-marginalizes $\mathcal{I}_n$, then max-marginalizes D_n, sum-marginalizes $\mathcal{I}_{i-1}$, etc. This type of elimination order is called a *strong elimination order*.[21] The method adapted for working with BTs is shown below.

(1) Initialization phase
   (a) For each $\phi \in \Phi$ obtain a BPT representing ϕ.
   (b) For each $\psi \in \Psi$ obtain a BUT representing ψ.
   (c) Prune all trees.

(2) While there are variables to remove
   (a) Select the next variable to remove (X).
   (b) Let Φ_X the set of all BPTs containing X. Combine (multiplication) all the BPTs in Φ_X giving as a result a new tree $\mathcal{BPT}_X$.
   (c) Let Ψ_X the set of all BUTs containing X. Combine (addition) all the BUTs in Ψ_X giving as a result a new tree $\mathcal{BUT}_X$.
   (d) Remove the variable X using sum-marginalization (chance nodes) or max-marginalization (decision nodes) from $\mathcal{BPT}_X$ and $\mathcal{BUT}_X$. New trees are obtained as result: $\mathcal{BUT}'_X$ and $\mathcal{BPT}'_X$.
   (e) Update the potential sets:

$$\Phi = (\Phi \backslash \Phi_X) \cup \{\mathcal{BPT}'_X\} \qquad \Psi = (\Psi \backslash \Psi_X) \cup \left\{ \frac{\mathcal{BUT}'_X}{\mathcal{BPT}'_X} \right\}.$$

4.3.2. *Lazy evaluation*

Lazy Evaluation (LE)[14,22] is an evaluation algorithm based on message passing in a *strong junction tree*, which is a representation of an ID built by moralization and by triangulating the graph using a strong elimination order.[21] Nodes in the strong junction tree correspond to *cliques* (maximal complete sub-graphs) of the triangulated graph. Two neighbour cliques are connected by a separator which contains the intersection of the variables in both cliques. Initially, each potential is associated to the clique closest to the root containing all its variables. Propagation is performed by message-passing from leaves to the root. A message consists on a list of potentials from which variables not present in the parent separator has been removed using the VE algorithm (see[14,22] for more details about the message computation). For the message computation, operations with BTs are used instead of their counterparts for tables. The general scheme of LE for working with BTs is shown below.

(1) Initialization phase:
   (a) For each $\phi \in \Phi$ obtain a BPT representing ϕ.

(b) For each $\psi \in \Psi$ obtain a BUT representing ψ.
(c) Prune all trees.

(2) Build the Strong Junction Tree from the ID.
(3) Propagation phase:
   (a) Associate each potential (trees) in $\Phi \cup \Psi$ to only one clique containing all the variables in its domain.
   (b) Start message-passing from leaves to root.

4.3.3. *Symbolic probabilistic inference*

The *Symbolic Probabilistic Inference* algorithm (SPI)[16,23,24] is an algorithm that tries to find the optimal order for the combinations and marginalizations by choosing at each step the best operation. For evaluating IDs, as VE does, SPI removes all variables in the decision problem in reverse order of the partial ordering imposed by the information constraints. Yet, VE is guided by an elimination order while SPI is guided by a combination order. That is, VE chooses at each step the next variable to remove while SPI chooses the next pair of potentials to combine and eliminate the variables when possible. In this sense SPI is finer grained than VE. More details about this method for evaluating IDs are given in a previous work.[16] The general scheme of SPI algorithm with BTs is:

(1) Initialization phase
   (a) For each $\phi \in \Phi$ obtain a BPT representing ϕ.
   (b) For each $\psi \in \Psi$ obtain a BUT representing ψ.
   (c) Prune all trees.
(2) For $k = n$ until 1
   (a) Using a greedy algorithm, combine potentials (trees) containing any variable in $\mathcal{I}_k$. After each iteration, sum-marginalize out all the variables in $\mathcal{I}_k$ that are contained in only one potential.
   (b) Using a greedy algorithm combine all potentials (trees) containing D_k. Then max-marginalize out D_k.
(3) Proceed as in step 2(a) for removing variables in $\mathcal{I}_0$.

5. Experimental Work

In this section, the performance of NTs and BTs for IDs inference is analyzed. However, we must introduce first some concepts about *Multi-Objective Optimization Problems*.

5.1. *Multi-objective optimization problems*

When approximating a potential represented as a tree there are two objectives to consider: size and error of the approximation. These two objectives can be controlled

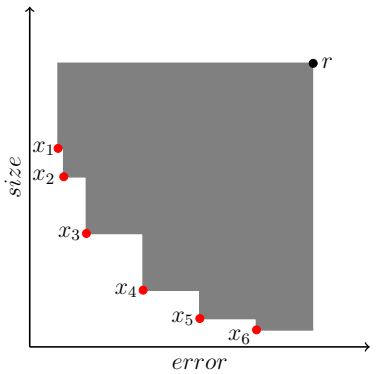


Fig. 9. Hyper-volume for a minimization problem.

with a threshold for pruning, ε. Low values of ε will produce large trees with low errors while low values of ε will lead to small trees and big errors. Thus, the problem of finding the best approximation can be considered as a multi-objective optimization problem (MOP)[25] with two objectives to be minimized. Hence, optimizing means finding a solution with acceptable values for all the objectives.

There are several possible solutions for this optimization problem as there is not additional information about the preferred objective. Moreover, both objectives are in conflict as it was stated when the role of the threshold for pruning ε was discussed. In MOPs the set of acceptable solutions composes the Pareto set (non-dominated solutions). In order to compare two solution sets (produced by two different representations of potentials) we have used the *hyper-volume* indicator.[26] It is based on representing the solution set in a n-dimensional space being n the number of objectives. In this space a reference point r characterizes the worst possible solution. In our case, the axis of the space represents the size of the potentials and the error produced by the approximation. The hyper-volume indicator is an unary value measuring the percentage of area dominated by a certain set of solutions (Pareto-set). Its maximum value is 1 and corresponds to a solution set dominating the rest of possible solutions (the best one). Figure 9 shows an example of a Pareto-set containing six different trees represented by x_1, x_2, x_3, x_4, x_5, and x_6. The hyper-volume measures the portion of area in grey.

5.2. *Objectives and procedure*

There are several objectives related to the set of experiments performed in this paper. Given a certain set of IDs whose potentials are represented as BTs, NTs and tables we try:

- To test if the representation as BTs requires less memory space than using NTs or tables. The gain in memory space should imply two benefits: a reduction in the computation time as well as the ability to evaluate more complex models.

- To check what representation gives better performance (respect to memory space and error) when computing approximate solutions varying ε.
- To compare BTs with other representations as algebraic decision diagrams (ADDs)[27] and *affine algebraic decision diagrams* (AADDs).[17]

The set of IDs used for this experimental work[a] must be complex enough in order to justify the computation of an approximate solution. However, they must be simple enough to allow an exact evaluation as well. This is the only way to compute the error introduced in the approximate solutions. The IDs used for testing contain:

- A real world ID used for the treatment of gastric NHL disease[18] with 3 decision nodes, 1 value node and 17 chance nodes. The average number of states in each variable is 2.90 and the average potential size is 468.11 values.
- A set of 100 randomly generated IDs. The features for this set are shown in Table 3.

Table 3. Features of random generated IDs.

	Average	Minimum	Maximum
Chance nodes	25.6	8	46
Value nodes	1	1	1
Decision nodes	2.5	2	3
Average number of states	3.33	2	5.38
Average potential size	13.2	7.2	22.4

For the experiments, each ID is evaluated using tables, NTs and BTs with different ε in the interval $[0, 1]$. The inference algorithms employed are VE, LE, and SPI (see Sec. 4.3). For each evaluation it is measured:

- The storage size of potentials is measured before and after the removal of each variable (or a set of variables for the SPI algorithm). The reduction in memory space requirements is computed as

$$spaceSavings = 1 - \frac{meanPotSize_{trees}}{meanPotSize_{tables}}\,. \tag{13}$$

- The gain respect to computation time is computed with Eq. (14). It should be noticed that the evaluation time with trees also includes the time required for building the trees from tables and pruning them.

$$speedup = \frac{time_{tables}}{time_{trees}}\,. \tag{14}$$

- To analyze the error produced by the approximation, the MEU is calculated using trees and tables (see Eq. (3)). Then, the absolute error is computed with

[a]All the IDs are available in `http://leo.ugr.es/rcabanas/binarytrees/`

Eq. (15). The error is analyzed together with the storage requirements: all the pairs $(meanPotSize_{trees}, absoluteError)$ for the same kind of representation compose a solution set. Pareto front and hyper-volume are computed for every solution set.

$$absoluteError = |MEU_{trees}(\widehat{\Delta}) - MEU_{tables}(\widehat{\Delta})| \,. \tag{15}$$

5.3. *Results for NHL ID*

5.3.1. *Storage requirements and computation time*

Figure 10 shows the storage requirements for handling all the potentials during the evaluation of the NHL ID with two different values of ε and the algorithms VE, LE and SPI. The vertical axis represents the storage size using a logarithmic scale. The horizontal axis indicates the evaluation stages (removal of all its variables). The corresponding space savings are shown in Table 4. It can be observed that, using any of the three algorithms, less space is needed with trees (NTs and BTs)

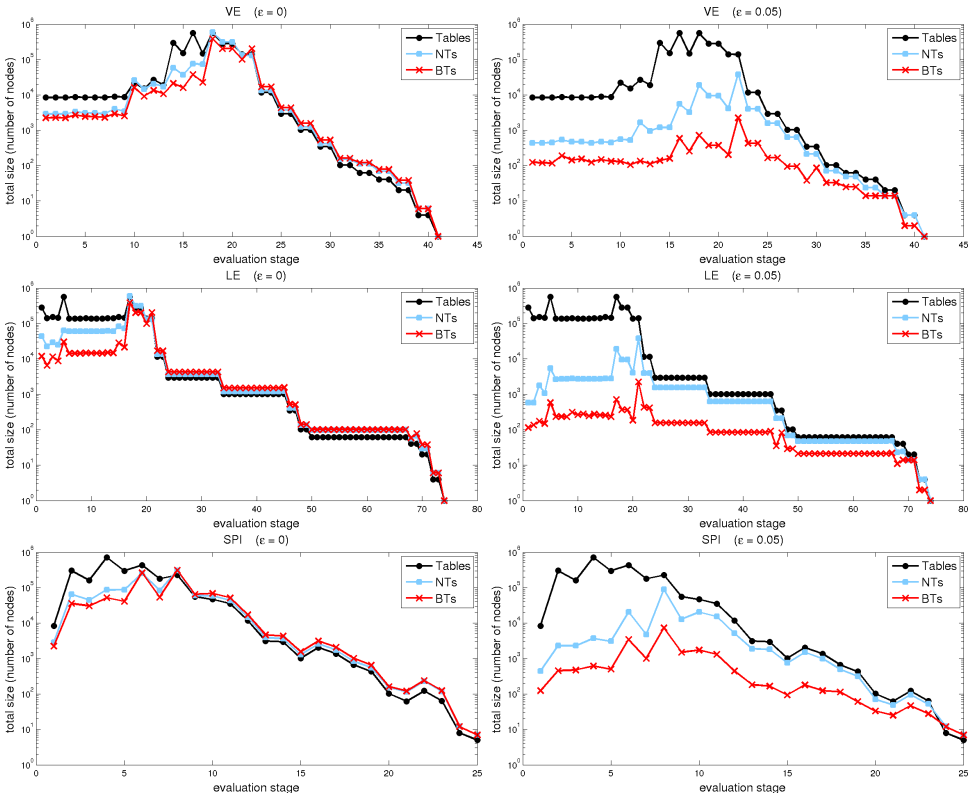


Fig. 10. Size of potentials during NHL ID evaluation with tables, NTs and BTs with two different ε threshold values and the algorithms VE, LE and SPI.

Table 4. Space savings that results from using trees (NTs and BTs) instead of tables during NHL ID evaluation with two different ε thresholds values and the algorithms VE, LE and SPI.

	VE		LE		SPI	
	$\varepsilon = 0.0$	$\varepsilon = 0.05$	$\varepsilon = 0.0$	$\varepsilon = 0.05$	$\varepsilon = 0.0$	$\varepsilon = 0.05$
NTs	0.318	0.959	0.431	0.965	0.55	0.924
BTs	**0.524**	**0.997**	**0.666**	**0.997**	**0.592**	**0.992**

than with tables. As long as ε is increased the memory space reduction is more noticeable. We can also observe that, in the final evaluation stages, there are not noticeable differences in storage requirements. This may be due to the combination of the potentials producing another ones where the effect of the initial prune is lost. If we compare the space savings obtained with both kinds of trees, the reduction is higher with BTs than with NTs. Similar space savings values are obtained for all the evaluation algorithms analyzed.

Figure 11 shows the computation time with different ε values. Since the evaluation time using tables is much higher than using trees, this is not shown in the graphic. The evaluation time for each algorithm with tables is approximately 14000 ms, 13000 ms and 10000 ms.

It can be observed that the computation with BTs requires less time with a decreasing reduction as long as ε value increases. Perhaps this can be explained due to the behaviour of pruning operation on NTs: the values for all the states are collapsed into a single one. That is, this operation is more drastic on NTs although more error will be introduced in the solutions. If the speed up values obtained with each algorithm are compared, it can be observed that the highest improvements are obtained with the VE algorithm for BTs. The reason for that is that the VE algorithm for tables is the slowest one. We can also observe that the computation time required by LE with trees is higher than the required one by VE: the reason for that is that LE has an overhead due to the time needed for building the strong junction tree (which is independent of the potential representation). Thus, another conclusion is that the use of BTs makes the VE algorithm faster than LE.

5.3.2. *Error against size*

The experiments reveal that, using any of the algorithms proposed, BTs produce better approximations than NTs: the same error level is achieved using smaller trees. This situation can be shown in Fig. 12. For each algorithm, it includes one graphic showing a comparison of the absolute error versus the mean potential size obtained for computing the MEU of the NHL ID. Each point corresponds to a different evaluation with certain value of ε. It can be observed that the dominated area is always bigger for BTs.

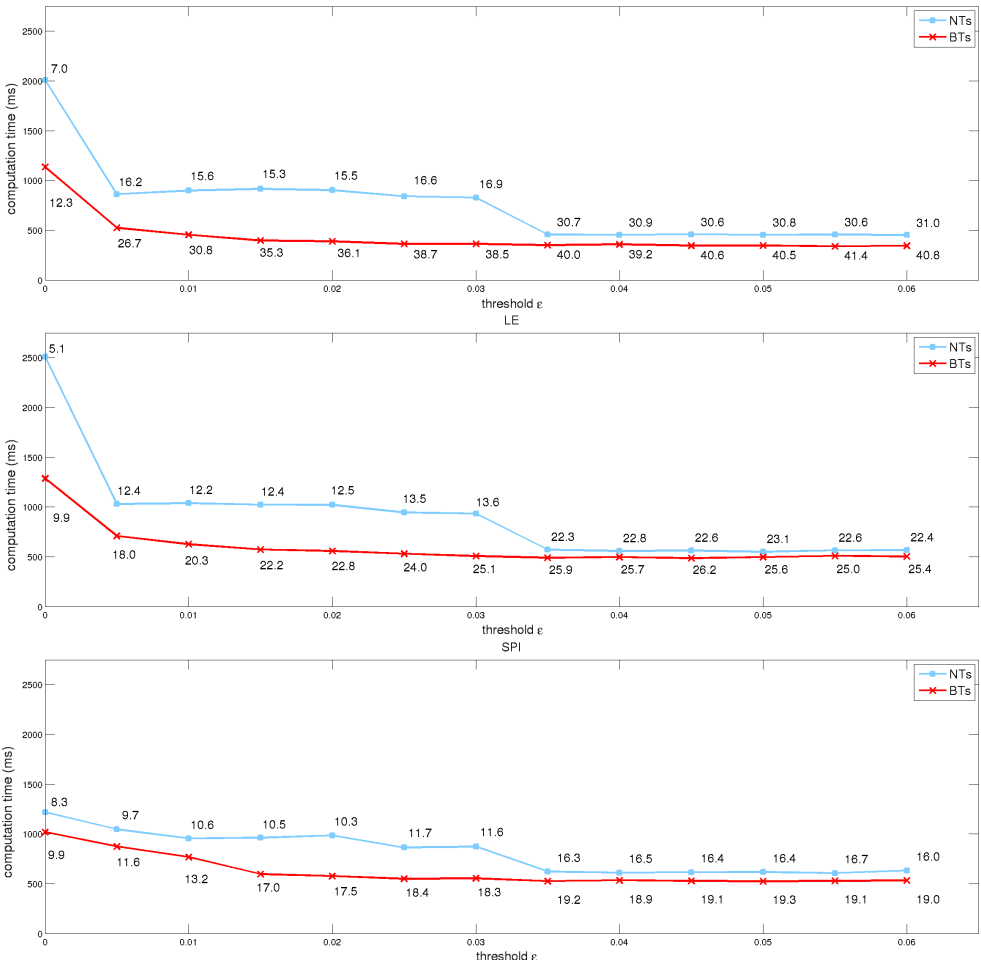


Fig. 11. Evaluation time and speed up obtained during NHL ID evaluation with NTs and BTs and different values for ε and the algorithms VE, LE and SPI. The evaluation time for each algorithm with tables is approximately 14000 ms, 13000 ms and 10000 ms.

All the pairs $(meanPotSize_{trees}, absoluteError)$ for the same representation compose a solution set. For each solution set the Pareto front and hyper-volume are computed using the reference point r. These numbers are included in Table 5. Each column corresponds to one evaluation algorithm whereas each row is used for every kind of potential: NT and BT. It can be observed that the higher hyper-volume values using BTs (H_{BT}) are always larger (better) than those using NTs (H_{BT}). Thus, we conclude that better approximations are achieved using BTs for this ID.

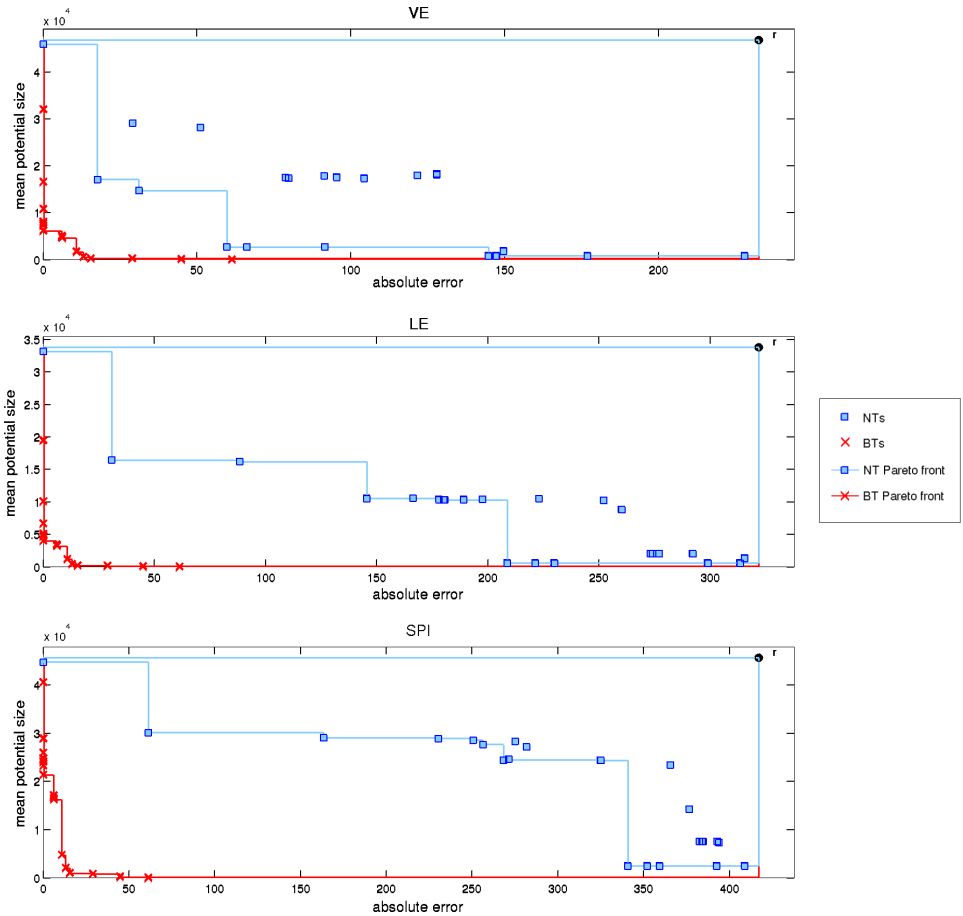


Fig. 12. Comparison of the absolute error versus the mean potential size for computing the MEU of the NHL ID. This ID has been evaluated with different threshold values and the algorithms VE, LE and SPI.

Table 5. Hyper-volume values obtained from points shown in Fig. 12.

	VE	LE	SPI
H_{NT}	0.835	0.649	0.407
H_{BT}	**0.992**	**0.995**	**0.987**

5.4. *Results for random IDs*

5.4.1. *Evaluability*

Table 6 contains the percentage of IDs being evaluable using each kind of representation (exact and approximate). The pruning threshold ε used for the approximate evaluations is 1.0. Those IDs that can not be computed fail due to memory space restrictions. It must be noticed that this set of random IDs may not be optimal for

Table 6. Percentage of random IDs that are evaluable using tables, NTs and BTs.

	VE exact	VE aprox.	LE exact	LE aprox.	SPI exact	SPI aprox.
Tables	83%	–	85%	–	75%	–
NTs	79%	81%	79%	80%	73%	75%
BTs	79%	94%	79%	86%	73%	94%

making this test about evaluability: perhaps random IDs do not contain the same level of asymmetries presented in real world IDs. However we have decided to use this set of IDs due to the lack of a large set of real world IDs available for testing. Anyway it can be observed that, using any of the algorithms proposed, more IDs can be evaluated with BTs when approximate solutions are computed.

The following sections show the results obtained for each random ID in terms of storage requirements, computation time and accuracy. It should be noticed that only those IDs being evaluable using all the kinds of representations and algorithms are considered.

5.4.2. *Storage requirements and computation time*

Table 7 includes the average space savings obtained using NTs and BTs and for different ε values (see Eq. (13)). When exact evaluation is carried out ($\varepsilon = 0$) tables require less space than trees: for IDs with a low number of context-specific independencies, the representation with tables is more efficient due to the additional space required for storing internal nodes in tree representations. Yet, when approximate evaluation is performed ($\varepsilon > 0$) the representation with BTs requires less space. Similar space savings values are obtained for all the evaluation algorithms analyzed.

Table 7. Average space saving obtained using trees instead of tables with different ε values.

		$\varepsilon = 0.0$	$\varepsilon = 0.05$	$\varepsilon = 0.5$	$\varepsilon = 1.0$
VE	NTs	**−0.561**	−0.099	0.126	0.565
	BTs	−0.796	**0.266**	**0.621**	**0.963**
LE	NTs	**−0.409**	−0.082	0.064	0.511
	BTs	−0.594	**0.295**	**0.571**	**0.951**
SPI	NTs	**−0.593**	−0.202	−0.026	0.392
	BTs	−0.841	**0.281**	**0.634**	**0.966**

The improvement in computation time can be analyzed using speedup (Eq. (14)). Table 8 contains the speedup values for tables, NTs and BTs with different values of ε. It can be seen the improvements achieved by BTs: the algorithms VE and SPI are faster when using BTs than with NTs or tables. However, the LE algorithm has a better performance with tables. When this method is used with trees, the overhead introduced by pruning and sorting the trees consumes all the benefits for reduced size potentials.

Table 8. Average speedup for IDs using tables and trees (NTs and BTs).

		$\varepsilon = 0.0$	$\varepsilon = 0.05$	$\varepsilon = 0.5$	$\varepsilon = 1.0$
VE	NTs	0.926	0.974	0.99	1.075
	BTs	**1.563**	**1.831**	**1.85**	**1.821**
LE	NTs	0.296	0.323	0.318	0.35
	BTs	**0.472**	**0.606**	**0.622**	**0.649**
SPI	NTs	0.846	0.911	0.912	0.965
	BTs	**1.417**	**1.951**	**1.961**	**2.025**

5.4.3. *Error against size*

Table 9 contains the results of performing a Wilcoxon signed-rank test with the hyper-volumes for all the random IDs evaluated with the VE, LE and SPI algorithms. The null hypothesis states that there is no difference between the hyper-volumes for BTs and NTs (both share a similar performance). The significance level for rejecting the hypothesis is 5%. For each algorithm the table contains the $p-value$, the number of IDs analyzed, the percentage of IDs where BTs outperform NTs and where NTs offer better results and finally the conclusion of the test. It can be observed that the null hypothesis is always rejected. If we compare the three evaluation algorithms, the highest percentage of IDs where BTs outperform NTs is obtained with the LE algorithm. This is due to the lower number of combinations performed with LE.

Table 9. Results of the Wilcoxon test for the results using NTs and BTs.

	p-value	IDs	BTs wins	NTs wins	Rejected
VE	$5.67 \cdot 10^{-5}$	48	70.83%	25.0%	*yes*
LE	$3.52 \cdot 10^{-9}$	48	95.83%	0.00%	*yes*
SPI	$6.47 \cdot 10^{-7}$	48	79.17%	16.67%	*yes*

5.5. *Comparison with other approaches*

As it was mentioned before, the comparison with ADDs and AADDs is limited to inference in BNs due to the lack of proposals for using these structures for ID evaluation. Our experiment considers 6 BNs used for the experimental work presented in Ref. 17. Table 10 contains the storage requirements for encoding the potentials using tables, BTs, ADDs and AADDs. The data for ADDs and AADDs are taken from the results in Ref. 17. In order to simplify the results, NTs are not considered in this section. In previous sections it is already shown that NTs offer worse results than BTs.

It can be observed that exact BTs require less space than ADDs in 4 out of 6 BNs. By contrast, AADDs use less space than BTs in 4 of the 6 BNs. However,

Table 10. Number of table entries/nodes in the original Bayesian networks.

		BTs					
	Tables	$\varepsilon = 0.0$	$\varepsilon = 0.05$	$\varepsilon = 0.5$	$\varepsilon = 1.0$	ADDs	AADDs
Alarm	1192	931	467	45	37	689	405
Carpo	636	706	520	84	60	955	360
Hailfinder	9045	5974	486	56	56	4511	2538
Insurance	2104	1477	265	35	29	1596	775
NoisyMax15	$6.557 \cdot 10^4$	660	84	16	16	$1.254 \cdot 10^5$	1066
NoisyOr15	$1.311 \cdot 10^5$	338	72	16	16	$2.021 \cdot 10^5$	$4.099 \cdot 10^4$

the main benefit of BT representation is related to its capability of producing approximate solutions with significant memory savings.

The speedups comparing computation time for 100 random queries (with a target variable and one evidence variable) are shown in Table 11. It is noticeable the speedups obtained with BTs mainly for the last two BNs: their potentials contain a high number of repeated values.

Table 11. Speedup comparing BTs, ADDs and AADDs with respect to the time using tables.

	BTs					
	$\varepsilon = 0.0$	$\varepsilon = 0.05$	$\varepsilon = 0.5$	$\varepsilon = 1.0$	ADDs	AADDs
Alarm	1.304	2.545	3.099	3.039	1.227	2.357
Carpo	1.157	1.136	1.126	1.19	1.018	1.184
Hailfinder	2.733	4.688	6.12	17.93	2.75	9.778
Insurance	3.839	11.52	25.59	23.83	2.397	7.514
NoisyMax15	686.1	800.5	960.6	960.6	0.5478	39.29
NoisyOr15	651.8	782.2	782.2	977.8	0.7859	5.759

6. Conclusions and Future Work

This paper proposes the use of BTs for representing the potentials involved in IDs. This kind of tree allows representing context-specific independencies that are finer-grained compared to those encoded using NTs or tables. In particular we focus on the BTs representing utility potentials (BUTs): detailed methods for building and pruning a BUT are given. BTs representing probability potentials were already described in a previous work.[11] It is also explained how BTs are used during the evaluation of IDs using the three different algorithms (VE, LE and SPI).

The experimental work shows that, in general, less memory space is required for storing potentials as a BT than using a NT or a table. As a consequence, the ID evaluation is faster using BTs. In fact, some IDs which are not evaluable with tables due to space restriction can be evaluated with BTs. However, for many IDs it is necessary to use a threshold for pruning higher to 0 in order to obtain any benefits from the use of BTs. Another conclusion is that using BTs for evaluating IDs offers

better approximate solutions than using NTs. The same error level is achieved using a BT of smaller size than the corresponding NT. If the three evaluation algorithms are compared, the best improvements achieved by BTs in terms of computation time and storage are obtained with the VE and SPI algorithms. By contrast, the most accurate approximate solutions are obtained using the LE algorithm. BTs are also compared with other approaches for representing potentials such as ADDs and AADDs. However it is limited to inference in BNs due to the lack of proposals for using these structures for ID evaluation. The possibility of approximating potentials using BTs allows reducing even more the size of potentials and to obtain better results with BTs than with ADDs and AADDs.

As regards future directions of research, we can study the impact that the heuristics used for choosing the elimination order has on the results. The heuristics used so far consider that potentials are represented as tables. It could be interesting to use any heuristic that considers that potentials are represented as BTs. It could also be interesting studying an alternative method for approximating utilities that prunes those values corresponding to a scenario extremely difficult to reach. Finally, another direction of research could be the integration of restrictions and BTs. This would allow addressing asymmetric decision problems.

Acknowledgments

This research was supported by the Spanish Ministry of Economy and Competitiveness under projects TIN2010-20900-C04-01 and TIN2013-46638-C3-2-P, the European Regional Development Fund (FEDER), the FPI scholarship program (BES-2011-050604). The authors have also been partially supported by "Junta de Andalucía" under projects P10-TIC-06016. We also thank the reviewers for their constructive comments.

Appendix

In this section it is proved that expression given in Proposition 1 (p. 67) for computing the information gain is equivalent to Eq. (6).

Proof. The information gain in Eq. (6) is:

$$\begin{aligned} I(t, X_i, \Omega^{t_l}_{X_i}, \Omega^{t_r}_{X_i}) &= D(\psi, \mathcal{BUT}_j) - D(\psi, \mathcal{BUT}_{j+1}(t, X_i, \Omega^{t_l}_{X_i}, \Omega^{t_l}_{X_i})) \\ &= \sqrt{\sum_{i=1}^{n} \left(v_i - \frac{\sum_{i=1}^{n} v_i}{n} \right)^2} \\ &\quad - \sqrt{\sum_{i=1}^{n_l} \left(vl_i - \frac{\sum_{i=1}^{n_r} vl_i}{n_l} \right)^2 + \sum_{i=1}^{n_r} \left(vr_i - \frac{\sum_{i=1}^{n_r} vr_i}{n_r} \right)^2}. \end{aligned} \quad (16)$$

The variance σ^2 of $v_1, v_2, \ldots, v_n$ is:

$$\sigma^2 = \frac{\sum_{i=1}^{n}\left(v_i - \frac{\sum_{i=1}^{n} v_i}{n}\right)^2}{n} \Leftrightarrow \sigma^2 \cdot n = \sum_{i=1}^{n}\left(v_i - \frac{\sum_{i=1}^{n} v_i}{n}\right)^2. \tag{17}$$

Similarly, the variance σ_l^2 of $vl_1, vl_2, \ldots, vl_{n_l}$ and the variance σ_r^2 of $vr_1, vr_2, \ldots, vr_{n_r}$ are:

$$\sigma_l^2 = \frac{\sum_{i=1}^{n_l}\left(v_{li} - \frac{\sum_{i=1}^{n_l} vl_i}{n_l}\right)^2}{n_l} \Leftrightarrow \sigma_l^2 \cdot n_l = \sum_{i=1}^{n_l}\left(vl_i - \frac{\sum_{i=1}^{n_l} vl_i}{n_l}\right)^2. \tag{18}$$

$$\sigma_r^2 = \frac{\sum_{i=1}^{n_r}\left(v_{ri} - \frac{\sum_{i=1}^{n_r} vr_i}{n_r}\right)^2}{n_r} \Leftrightarrow \sigma_r^2 \cdot n_r = \sum_{i=1}^{n_r}\left(vr_i - \frac{\sum_{i=1}^{n_r} vr_i}{n_r}\right)^2. \tag{19}$$

Using Eqs. (17), (18), and (19), the information gain in Eq. (16) can be written as:

$$I(t, X_i, \Omega_{X_i}^{t_l}, \Omega_{X_i}^{t_r}) = \sqrt{\sigma^2 \cdot n} - \sqrt{\sigma_l^2 \cdot n_l + \sigma_r^2 \cdot n_r}\,. \tag{20}$$

Since the variance of a variable is the mean of the square variable minus the square of the mean, then:

$$I(t, X_i, \Omega_{X_i}^{t_l}, \Omega_{X_i}^{t_r}) = \sqrt{\left(\frac{\sum_{i=1}^{n} v_i^2}{n} - \left(\frac{\sum_{i=1}^{n} v_i}{n}\right)^2\right) \cdot n}$$

$$- \sqrt{\left(\frac{\sum_{i=1}^{n_l} vl_i^2}{n_l} - \left(\frac{\sum_{i=1}^{n_l} vl_i}{n_l}\right)^2\right) \cdot n_l + \left(\frac{\sum_{i=1}^{n_r} vr_i^2}{n_r} - \left(\frac{\sum_{i=1}^{n_r} vr_i}{n_r}\right)^2\right) \cdot n_r}\,. \tag{21}$$

Finally, simplifying the following expression is obtained:

$$I(t, X_i, \Omega_{X_i}^{t_l}, \Omega_{X_i}^{t_r}) = \sqrt{\sum_{i=1}^{n} v_i^2 - \frac{\left(\sum_{i=1}^{n} v_i\right)^2}{n}}$$

$$- \sqrt{\sum_{i=1}^{n_l} vl_i^2 - \frac{\left(\sum_{i=1}^{n_l} vl_i\right)^2}{n_l} + \sum_{i=1}^{n_r} vr_i^2 - \frac{\left(\sum_{i=1}^{n_r} vr_i\right)^2}{n_r}}\,. \tag{22}$$

□

References

1. R. Cabañas, M. Gómez and A. Cano, Approximate inference in influence diagrams using binary trees, in *Proceedings of the Sixth European Workshop on Probabilistic Graphical Models* (*PGM-12*), 2012.
2. S. M. Olmsted, Representing and solving decision problems, *Dissertation Abstracts Int. Part B: Science and Engineering* **45**(3) (1984).

3. R. A. Howard and J. E. Matheson, Influence diagram retrospective, *Decision Analysis* **2**(3) (2005) 144–147.
4. S. L. Lauritzen and D. Nilsson, Representing and solving decision problems with limited information, *Management Science*, 2001, pp. 1235–1251.
5. J. M. Charnes and P. P. Shenoy, Multistage Monte Carlo method for solving influence diagrams using local computation, *Management Science*, 2004, pp. 405–418.
6. A. Cano, M. Gómez and S. Moral, A forward–backward Monte Carlo method for solving influence diagrams, *Int. J. Approximate Reasoning* **42**(1) (2006) 119–135.
7. A. Cano, S. Moral and A. Salmerón, Penniless propagation in join trees, *Int. J. Intelligent Systems* **15**(11) (2000) 1027–1059.
8. A. Salmerón, A. Cano and S. Moral, Importance sampling in Bayesian networks using probability trees, *Computational Statistics & Data Analysis* **34**(4) (2000) 387–413.
9. M. Gómez and A. Cano, Applying numerical trees to evaluate asymmetric decision problems, in *Symbolic and Quantitative Approaches to Reasoning with Uncertainty*, eds. T. D. Nielsen and N. Zhang, *Lecture Notes in Computer Science*, Vol. 2711, Springer Berlin Heidelberg, 2003, pp. 196–207.
10. C. Boutilier, N. Friedman, M. Goldszmidt and D. Koller, Context-specific independence in Bayesian networks, in *Proc. 12th Int. Conf. on Uncertainty in Artificial Intelligence*, Morgan Kaufmann Publishers Inc., 1996, pp. 115–123.
11. A. Cano, M. Gómez-Olmedo and S. Moral, Approximate inference in Bayesian networks using binary probability trees, *Int. J. Approximate Reasoning* **52**(1) (2011) 49–62.
12. N.L. Zhang and D. Poole, Exploiting causal independence in Bayesian network inference, *J. Artificial Intelligence Research* **5** (1996) 301–328.
13. F. V. Jensen and T. D. Nielsen, *Bayesian Networks and Decision Graphs* (Springer Verlag, 2007).
14. A. L. Madsen and F. V. Jensen, Lazy evaluation of symmetric Bayesian decision problems, in *Proc. 15th Conf. Uncertainty in Artificial Intelligence*, Morgan Kaufmann Publishers Inc., 1999, pp. 382–390.
15. A. L. Madsen, Improvements to message computation in lazy propagation, *Int. J. Approximate Reasoning* **51**(5) (2010) 499–514.
16. R. Cabañas, A. L. Madsen, A. Cano and M. Gómez-Olmedo, On SPI for evaluating Influence Diagrams, in *Information Processing and Management of Uncertainty in Knowledge-Based Systems*, Springer International Publishing, 2014, pp. 506–516.
17. S. Sanner and D. McAllester, Affine algebraic decision diagrams (AADDs) and their application to structured probabilistic inference, in *IJCAI* **2005** (2005) 1384–1390.
18. P. J. F. Lucas and B. Taal, Computer-based decision support in the management of primary gastric non-Hodgkin lymphoma, *UU-CS* (1998-33), 1998.
19. S. K. M. Wong and C. J. Butz, Contextual weak independence in Bayesian networks, in *Proc. 15th Conf. Uncertainty in AI*, Morgan Kaufmann Publishers Inc., 1999, pp. 670–679.
20. J. R. Quinlan, Induction of decision trees, *Machine learning* **1**(1) (1986) 81–106.
21. U. Kjærulff, Triangulation of graphs – algorithms giving small total state space, Research Report R-90-09, Department of Mathematics and Computer Science, Aalborg University, Denmark, 1990.
22. A. L. Madsen and F. V. Jensen, Lazy propagation: a junction tree inference algorithm based on lazy evaluation, *Artificial Intelligence* **113**(1-2) (2004) 203–245.
23. R. D. Shachter, B. D'Ambrosio and B. Del Favero, Symbolic probabilistic inference in belief networks, in *AAAI* **90** (1990) 126–131.

24. Z. Li and B. D'Ambrosio, Efficient inference in Bayes networks as a combinatorial optimization problem, *Int. J. Approximate Reasoning* **11**(1) (1994) 55–81.
25. Y. Jin and B. Sendhoff, Pareto-based multiobjective machine learning: An overview and case studies, *IEEE Transactions on Systems, Man and Cybernetics, Part C: Applications and Reviews* **38**(3) (2008) 397–415.
26. E. Zitzler, D. Brockhoff and L. Thiele, The hypervolume indicator revisited: On the design of pareto-compliant indicators via weighted integration, in *Evolutionary Multi-Criterion Optimization* (Springer, 2007), pp. 862–876.
27. R. I. Bahar, E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo and F. Somenzi, Algebric decision diagrams and their applications, *Formal methods in system design* **10**(2–3) (1997) 171–206.