Value-based potentials: Exploiting quantitative information regularity patterns in probabilistic graphical models

Gómez-Olmedo, M., Cabañas, R., Cano, A., Moral, S., & Retamero, O. P.

Published in:
International Journal of Intelligent Systems

DOI (link to publication from Publisher):
https://doi.org/10.1002/int.22573

Publication date:
2021

Document Version:
Accepted author manuscript, peer reviewed version

Value-Based Potentials: Exploiting Regularity Patterns of Quantitative Information in Probabilistic Graphical Models

Manuel Gómez-Olmedo[a], Rafael Cabañas[b], Andrés Cano[a], Serafín Moral[a], Ofelia Paula Retamero[a]

[a] *Department of Computer Science and Artificial Intelligence, University of Granada, Spain*
[b] *Istituto Dalle Molle di Studi sull'Intelligenza Artificiale, Lugano, Switzerland*

Abstract

The efficient representation of quantitative information in probabilistic graphical models (PGMs) is a challenge for complex models (i.e. problems with many variables, high degree of dependence between them or many states per variable). In this work, several alternative structures are introduced for facing this problem. All of them are guided by the values and are named Value-Based Potentials (VBPs). VBPs try to take advantage of the regularity patterns founded in the data (repetitions of values), regardless of how they appear. These new structures are compared with respect to standard tables or unidimensional arrays representation (1DA) and probability trees (PTs). This last structure seeks for reducing memory space. But this goal can only be achieved if there are context specific independence patterns, that is, repeated values correspond to consecutive indexes. VBPs structures try to overcome this limitation. The objective of this work is to analyse the capabilities of these new ways of representation. For this purpose, their features are studied from a theoretical point of view and are employed for encoding quantitative information for a set of well known models.

1. Introduction

Probabilistic graphical models (PGMs) [1] are efficient representations for problems under uncertainty. PGMs encode joint probability or utility distributions and are defined by two parts: first, a qualitative component in the form of a graph that encodes a set of dependencies among the variables (i.e., nodes) in the domain being modelled; secondly, a quantitative component consisting of a set of functions quantifying such dependencies. In PGMs over discrete domains, such as *Bayesian networks* (BN) [2, 3] or *influence diagrams* [4, 5], these functions are traditionally represented with tables or unidimensional arrays (marginal or conditional probability tables and utility tables, 1DA in general).

The sizes of 1DAs increase exponentially with the number of variables in their domains. This property may limit the ability to represent certain problems with large 1DAs (memory size requirements may be prohibitive). Moreover,

even if the model can be encoded with 1DAs problems may arise during their evaluation. In order to make inference these 1DAs are transformed for computing marginal or conditional probabilities for a certain variable, most probable explanation [6, 7], decision tables in the case of influence diagrams, etc. Some inference algorithms [8, 9, 10, 11, 12, 13, 14, 15, 16, 17] use basic operations on potentials: combination, restriction and marginalization. The first one computes the product of two potentials $\phi_1(\mathbf{X})$ and $\phi_2(\mathbf{Y})$ producing as result a new potential with higher dimension $\phi(\mathbf{X} \cup \mathbf{Y})$. In this way, 1DAs obtained as intermediate results can be so large that they exceed the memory capacity of the computer. Anyway, this paper does not consider operations on potentials. It is just a first step for analyzing the capabilities of VBPs for encoding quantitative information. The definition of operations on these structures will be a subject for future research.

Therefore, in order to deal with complex problems, it is essential to use efficient representations of the quantitative information of the model. Usually 1DAs encoding probabilities or utilities contain repeated values. For example, some combinations of values are not allowed and are represented with 0's. An efficient representation should take advantage of all these repetitions in order to reduce memory space. Moreover, a useful representation should offer the capability of being approximated with a trade-off between exactness and memory space. These two features should allow to represent and applying inference algorithms on complex problems that could not be solved if quantitative information were represented with 1DAs.

The importance of this problem is evidenced by previous attempts to obtain alternative structures to 1DAs. Two examples of alternative approaches are normal and binary probability trees (PTs and BPTs) [18, 19, 20, 21, 22, 23, 24]. These structures can capture context specific independencies [18] and save memory space when repeated values appear under certain circumstances. These representations also have the capability of obtaining approximations through pruning operation: assuming loss of information, some contiguous values can be substituted by their average value in order to reduce memory space. There are also previous work focused on improving the operations on potentials to alleviate the computational cost when dealing with complex models [25].

In this work some new alternative representations are considered. They are based on the properties of the values themselves and not on the contexts in which they appear or the structure of the potential. That is why they have been called *value-based potentials* (VBPs). The paper defines these structures, making a theoretical analysis of their properties and showing concrete examples of how they encode specific distributions of known and available Bayesian networks in the *bnlearn* package repository [26, 27] as well as other networks used in inference competitions [28, 29]. As it was mentioned before, this paper does not consider how to operate with VBPs in order to perform inference operations. The paper analyzes several alternatives for VBPs and their capabilities for offering competitive compact storage of quantitative information with respect to 1DAs and PTs.

The structure of the paper is as follows: section 2 defines some basic concepts and notation and some usual representations for potentials as arrays and trees. Section 3 introduces basic concepts about memory requirement analysis. Section 4 introduces VBPs representations. Section 5 analyzes VBPs alternatives. Section 6 presents the empirical evaluation performed for testing VBPs capabilities. Finally section 7 presents conclusions and lines for future research.

2. Basics

2.1. Definitions and notation

Let us first denote the basic notation. Upper-case *roman* letters will be used to denote random variables and lower-case for their values (or states). Thus, if X_i is a random variable, x_i will denote a generic value of X_i. The finite set of possible values of X_i is called domain and denoted Ω_{X_i}. For simplicity, we will consider variable values as integers starting with 0 and hence possible assignments will be $X_1 = 0$, $X_1 = 1$, $X_1 = 2$, etc. The cardinality of a variable, denoted $|\Omega_{X_i}|$, is the number of values in its domain. Similarly, we use bold-face upper-case *roman* letters to denote sets of variables, e.g. $\mathbf{X} := \{X_1, X_2, \dots X_N\}$ is a set of N variables. The Cartesian product $\prod_{X_i \in \mathbf{X}} \Omega_{X_i}$ is denoted by $\Omega_{\mathbf{X}}$. The elements of $\Omega_{\mathbf{X}}$ are called configurations of $\mathbf{X}$ and will be represented by $\mathbf{x} := \{X_1 = x_1, X_2 = x_2, \dots, X_N = x_N\}$ or simply $\mathbf{x} := \{x_1, x_2, \dots, x_N\}$ if the variables are obvious from the context.

Formally, a PGM contains three elements $\langle \mathbf{X}, P, \mathcal{G} \rangle$ where $\mathbf{X}$ is the set of variables in the problem with a joint probability distribution $P(\mathbf{X})$ and $\mathcal{G}$ is a graph that represents the dependence (and independence) relations between the variables. A PGM allows to represent P, which is usually high-dimensional, as a factorisation of lower dimensional local functions. For instance, in case of BNs, these are conditional distributions represented as tables or conditional probability tables (arrays in general, 1DAs). However, we will use the term *potential* which is more general: a potential ϕ for $\mathbf{X}$ is a function of $\Omega_{\mathbf{X}}$ over $\mathbb{R}_0^+$. In other words, each configuration $\mathbf{x} \in \Omega_{\mathbf{X}}$ is associated to a real value. Thus, 1DAs or any other function encoding the quantitative information in PGMs can be seen as representations of potentials.

Example 1. *Let us consider the variables X_1, X_2 and X_3 with 2, 3 and 2 states respectively. Then $\phi(X_1, X_2, X_3)$ is a potential defined on such variables with the values shown in Figure 1. Note that this potential is also the conditional distribution $P(X_3|X_1, X_2)$.*

The definition of structures for representing potentials requires the introduction of the following concept: *index of a configuration*. It is a unique numeric identifier representing each configuration in a given domain $|\Omega_{\mathbf{X}}|$. We will consider indexes starting with 0 (all the variables take their first value) and ending with $|\Omega_{\mathbf{X}}| - 1$. In the potential given in Figure 1, the index 0 is associated to $\{0, 0, 0\}$, the index 1 to $\{0, 0, 1\}$ and so on until the last one 11 which is associated to $\{1, 2, 1\}$.

x_1	x_2	x_3	$\phi(x_1, x_2, x_3)$
0	0	0	0.1
0	0	1	0.9
0	1	0	0.5
0	1	1	0.5
0	2	0	0.0
0	2	1	1
1	0	0	0.8
1	0	1	0.2
1	1	0	0.2
1	1	1	0.8
1	2	0	0.9
1	2	1	0.1

Figure 1: Representation of the potential $\phi(X_1, X_2, X_3)$ as a mapping assigning a numeric value to each configuration

It is possible to set a correspondence between indexes and configurations based on the concept of *weight* (a.k.a. *stride* or *step size*). Let us suppose a domain $\mathbf{X} := \{X_1, X_2, \ldots, X_N\}$. Each variable X_i has a weight w_i computed as follows:

$$w_i = \begin{cases} 1 & \text{if i=N} \\ |\Omega_{X_{i+1}}| \cdot w_{i+1} & \text{otherwise} \end{cases} \tag{1}$$

In the potential considered in Example 1 the values of *weights* are: $w_3 = 1, w_2 = 2, w_1 = 6$. Therefore the most left variable is the one with highest *weight*. Using *weights* the index of a certain configuration $\mathbf{x} := \{x_1, x_2, \ldots, x_N\}$ can be computed with the following expression:

$$index(\mathbf{x}) = \sum_{i=1}^{N} x_i \cdot w_i \tag{2}$$

Given a certain index k, its *associated configuration* is denoted $\mathbf{x}^{(k)}$ and satisfies that $index(\mathbf{x}^{(k)}) = k$. Given a certain index k, the value assigned to each variable X_i can be computed using the following expression:

$$x_i = (index//w_i)\%|\Omega_{\mathbf{X}_i}| \tag{3}$$

where // denotes integer division and % the module of the division.

Example 2. *Let us consider the potential $\phi(X_1, X_2, X_3)$ given in Example 1, with $w_1 = 6$, $w_2 = 2$ and $w_3 = 1$. Then, the indexes of the configurations in the domain can be computed as follows.*

$$
\begin{aligned}
index(\{0,0,0\}) &= 0 \cdot 6 + 0 \cdot 2 + 0 \cdot 1 = 0 \\
index(\{0,0,1\}) &= 0 \cdot 6 + 0 \cdot 2 + 1 \cdot 1 = 1 \\
index(\{0,1,0\}) &= 0 \cdot 6 + 1 \cdot 2 + 0 \cdot 1 = 2 \\
&\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\
index(\{1,2,1\}) &= 1 \cdot 6 + 2 \cdot 2 + 1 \cdot 1 = 11
\end{aligned}
$$

Note that the association between indexes and configurations requires an order of the variables in the domain. Though any order is valid, we will consider by default the order in which variables are written. E.g., in the potential $\phi(X, Y, Z)$, the first variable would be X. In addition, here we consider the first variable to have the highest weight. However, the opposite approach could be also considered: the first variable has weight 1 whereas the last one has the highest.

2.2. Representation for potentials

A potential is basically a multidimensional object with a dimension per variable. Thus, a standard method for storing it is to flatten it into a single 1D-array (1DA) in computer memory [1]. Thus, potential ϕ defined on a set of N variables, can be represented by the array $\mathcal{A}_\phi$ as follows.

$$
\mathcal{A}_\phi := \Big\{ \phi(0,0,\ldots,0), \phi(0,0,\ldots,1), \ldots, \phi(|\Omega_{X_1}|-1, |\Omega_{X_2}|-1, \ldots, |\Omega_{X_N}|-1) \Big\} \tag{4}
$$

The main advantage of the representation with 1DA consists in the fact that the position in which each value is stored coincides with the index of the corresponding configuration. This makes very efficient the access with indexes. The size of a 1DA, denoted $size(\mathcal{A}_\phi)$, is the number of entries and is equal to the number of configurations in the potential.

Example 3. *The potential $\phi(X_1, X_2, X_3)$ given in Example 1, can be represented as the following 1DA with 12 entries shown in Figure 2*

0	1	2	3	4	5	6	7	8	9	10	11
0.1	0.9	0.5	0.5	0.0	1.0	0.8	0.2	0.2	0.8	0.9	0.1

Figure 2: $\phi(X_1, X_2, X_3)$ as a 1DA

Some limitations of this representation are: the access to 1DA with configurations would require a transformation into indexes using Eq. 2; the coincidence between indexes and storage positions makes it necessary to store all values. Therefore, the storage of repeated values in consecutive positions cannot be avoided.

Probability trees (PTs) are an alternative structure that have been used for storing and operating with potentials in PGMs [19, 20, 22], both accurately and approximately. The $\mathcal{T}_\phi$ tree that represents a potential ϕ is a directed tree and labelled with two types of nodes: internal *nodes* that represent variables and *leaf* nodes representing the values of the potential. The internal nodes have outgoing arcs (one for each state of the associated variable). The size of a $\mathcal{T}$ tree, represented by $size(\mathcal{T})$ is defined as the number of nodes it contains.

Example 4. *The same potential given in the previous example as a PT is presented in Figure 3. This PT has a size of 21 nodes (12 leaves and 9 internal nodes).*

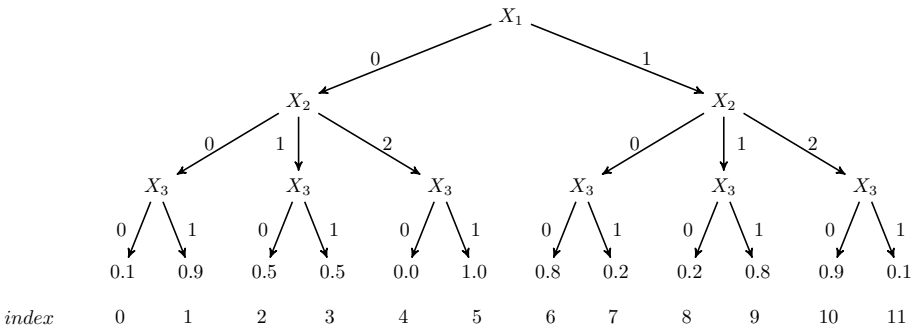

Figure 3: $\phi(X_1, X_2, X_3)$ as PT

With PTs, the most efficient way of access is via configuration: the tree must be traversed from root to leaves selecting the corresponding values for each variable until reaching a leaf node with its value. In addition, PTs can take advantage of context-specific independencies [18] so that many identical values can be grouped into a single one offering a compact storage. The operation of collapsing identical values is called *pruning*.

Example 5. *The potential in previous examples presents a context-specific independence that allows a reduction of its size: the value for $X_1 = 0, X_2 = 1$ is 0.5 regardless of the value of X_3. If the pruning is done, the result is a PPT (pruned PT) of size 19 shown in Figure 4.*

However another patterns of repetitions can not be pruned. Let us consider the tree presented in Figure 3. The values for indexes 7 and 8 are the same and are consecutive, but they can not be pruned because they correspond to configuration varying both in X_2 and X_3.

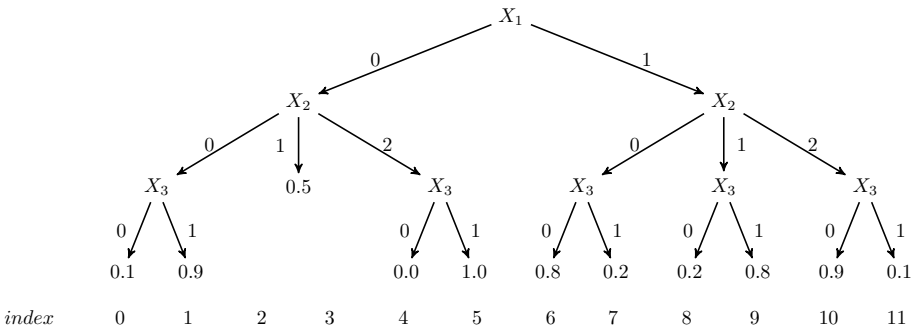

Figure 4: $\phi(X_1, X_2, X_3)$ as PPT

A variant of PTs, called binary trees (BPTs) [22] , can divide the domain of each variable into two subsets of states. This would allow finer grain context independencies to be exploited with respect to regular trees. For example, in the PT (left part of Figure 5), the values 0.4 in c configuration (left subtree) can not be pruned. However, with BPTs grouping states 0 and 2 of X_k would allow the value 0.4 to be represented with a single leaf node (as showed in the BPT of the right part of Figure 5). This reduces memory space for c context, although it would require more nodes for the right subtree (c' context). For this reason, only PTs and PPTs are considered in this paper.

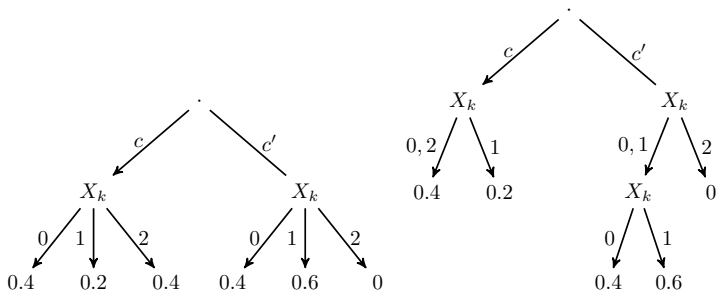

Figure 5: Binary tree representation

3. Memory requirements analysis

Even though the size of a representation gives an idea of the its complexity, a more accurate analysis of its memory space requirements is needed: any representation will consume additional elements (e.g., pointers, meta-information, etc.) and each of them use different data types. For this analysis we will consider a potential ϕ defined over a set of N variables $\mathbf{X} := \{X_1, X_2, \ldots X_N\}$. Additionally, the following notation related to the different memory sizes is defined.

- s_f will be the memory size required for storing a float value.
- s_i is the memory size for storing an index denoting a concrete configuration of $\Omega_{\mathbf{X}}$.
- s_r denotes the size of a reference to an object.
- s_v represents the memory space required for storing the information about a variable: name, cardinality and state names. As this depends of the names of variables and states we will assume a fix value for all of them (in fact, this memory space will be negligible respect to the whole memory used for storing a potential). Moreover we can define a standard way of coding variables with numerical identifiers and employ the same idea for their states.
- s_s denotes the memory size of the data structure used for storing information (array, dictionary, set, etc).

As it was mentioned before, the representation by means of 1D-array (1DA) has an important advantage: the values for configurations are stored consecutively. That is, the value in position k corresponds to index k configuration. This way it is not necessary to store information about indexes. Therefore, its codification supposes an amount of memory given by the number of values to store, the size of the array data structure and the memory required for its variables:

Proposition 1 (Memory space for an array representing a potential)**.** *Let $\mathcal{A}_\phi$ be a 1DA representing $\phi(\mathbf{X})$. Then the amount of required memory is given by the following expression.*

$$memory(\mathcal{A}_\phi) = N \cdot s_v + m \cdot s_f + s_s \tag{5}$$

where $m = |\Omega_{\mathbf{X}}|$ is the number of entries in the array.

Example 6. *From the previous examples, consider the potential $\phi(X_1, X_2, X3)$ and its codification as a 1D-array given in Example 3. Then the estimation of the memory size is:*

$$memory(\mathcal{A}_\phi) = 12s_f + s_s + 3s_v \tag{6}$$

The tree representation (PT and PPT) is usually less efficient in terms of memory requirements as the full structure of the tree must be stored. Thus, the amount of memory depends on the number of internal nodes, denoted n_I, and the number of leaves n_L. Note that it holds that $n_I + n_L = size(\mathcal{T})$. Internal nodes store links (or references) to sub-trees (each for a state of the corresponding variable). These links are stored into an array. Then, it is relevant to consider the number of outgoing arcs: $n_I^{(j)}$ denotes the total number of internal nodes for variables with j states.

Proposition 2 (Memory space for a tree representing a potential)**.** *Let $\mathcal{T}_\phi$ be a tree representing $\phi(\mathbf{X})$. Then the amount of required memory is estimated as follows.*

$$memory(\mathcal{T}_\phi) = N \cdot s_v + n_L \cdot s_f + \sum_{j=2}^{K} n_I^{(j)} \cdot (s_v + s_s + j \cdot s_r) \tag{7}$$

where $K = \max\{|\Omega_{X_i}| : X_i \in \mathbf{X}|\}$ is the maximal cardinality among the variables in the potential.

In case of a non-pruned tree, the number of leaf nodes will be equal to the number of configurations in the potential. As a consequence, the first terms in Equations(5) and (7) are equal. It can then be seen that main increase of trees is due to the data structures for storing links to subtrees and the repetition of variables information.

Example 7. *Let $\mathcal{T}_\phi$ be the PT from Example 4 (Figure 3) containing* 7 *internal nodes for binary variables and* 2 *internal nodes for the ternary ones and* 12 *leaves. Similarly, let $\mathcal{T}'_\phi$ be the PPT from Example 5 (Figure 4) with* 6 *internal nodes for binary variables,* 2 *internal nodes for ternary variables and* 11 *leaf nodes. Then, their memory cost can be computed with the following expressions:*

$$memory(\mathcal{T}_\phi) = 3s_v + 12s_f + 7(s_v + s_s + 2s_r) + 2(s_v + s_s + 3s_r) \tag{8}$$

$$memory(\mathcal{T}'_\phi) = 3s_v + 11s_f + 6(s_v + s_s + 2s_r) + 2(s_v + s_s + 3s_r) \tag{9}$$

In the previous example, the pruning operation has reduced the number of internal nodes as well as the number of leaves, but anyway the cost is superior to the size of the table representation. A large proportion of repeated values in the potential is usually required for the tree use less space than an array.

For a concrete analysis, the following sizes for data types are assumed (the real sizes can depend on the machine architecture; in fact, real sizes are not relevant as long as the same sizes are used for all the comparisons):

- long: 4 bytes (for indexes).
- float: 8 bytes (for real values).
- pointer or reference: 8 bytes (memory addresses).
- variable: 50 bytes (this includes the space for storing name, state names, etc). That is, $s_v = 50$.
- the concrete value of s_s will depend on the data structure employed:
  - array and list data structure (s_{arr} and s_{list} respectively): 16 bytes.
  - set data structure (s_{set}): 32 bytes.

– dictionary data structure (s_{dict}): 64 bytes.

Using these sizes, the estimations of memory for the representations $\mathcal{A}_\phi$, $\mathcal{T}_\phi$ and $\mathcal{T}'_\phi$ are 262, 1000 and 910 respectively.

4. Value-based representations

4.1. Motivation

At this point, it is clear the necessity of having efficient mechanisms for handling quantitative information for the tasks of representation, inference and learning with PGMs. We have already considered that PTs allow to capture some patterns of repetition in very specific situations. The underlying idea in VBPs is to let the representation process be guided by the values themselves and saving space from repetitions as much as possible. Therefore, the objectives of VBPs are:

- being able to take advantage of all repetition patterns, regardless of the order in which they appear. This is the capability to be analysed in this paper in order to check how VBPs allow to reduce memory space consumption.
- to facilitate the approximation task and the parallel management. This capabilities will be explored in a future work but it is important to consider these issues for reaching a good design.

4.2. Alternatives

The proposed alternatives can be classified in two groups depending on how to make the queries. First, *driven by values* approaches are based on the use of dictionaries in which the keys will be the values in the potentials. VDG and VDI alternatives belong to this group. Secondly, *driven by indices* alternatives where keys are indexes and not values. Into this group we present IDP and IDS and both are based on the use a combination of two different arrays.

The particular features for all of them will be presented below. However, a common capability is outlined here. It must be clear that these representations require a search for managing the information. This can be exploited defining a default value to return when the search fails. This default value can be set to 0.0 (this is the proposed alternative taking into account it will be the better approach for making inference operations) or fixed after analyzing the values of the potential. In this case it would help to select as default value the most repeated one (this alternative can reduce memory space but requires more computation time).

5. VBPs description

5.1. ***VDG: value-driven with grains***

Identical values in potentials will often appear in configurations with consecutive indexes. Consider for instance the potential given in Example 3, in which the value 0.5 appear in positions 2 and 3. Similar situation happens with value 0.2 in positions 7 and 8. As a consequence, a compact way of defining sets of configurations associated to a the same value could be by means of intervals (i.e., grains). Formally, a grain can be defined as follows.

Definition 1 (Grain)**.** *Let* $\mathbf{X}$ *be a set of variables and* i *and* j *indexes of valid configurations on* $\Omega_{\mathbf{X}}$*. A grain* $g(i, j)$ *defines a sequence of consecutive indexes* $i, i+1, \ldots j$*. Grains will be used for representing sequences of repeated values in VBPs.*

The VDG structure is based on the concept of grain: each possible value in a given potential will have associated one or more grains defining all the indexes for which the potential takes this value. More formally, a VDG can be defined as follows.

Definition 2 (Value-driven with grains)**.** *Let* ϕ *be a potential defined over* $\mathbf{X}$*, then a value-driven with grains (VDG) representing* ϕ *is a dictionary assigning to each value* $v \in \phi$ *a list of grains* L *such that for each grain* $g(i, j) \in L$ *it holds that* $\phi(\mathbf{x}_k) = v$ *for* $k = i, i+1, \ldots, j$*.*

Therefore, a VDG will store only non repeated values; for each of them it will be necessary to store the corresponding grains of information. Grains will be stored in lists. The link between values and grains is preserved with a dictionary data structure containing pairs $< value - list < grain >>$.

Example 8. *The potential* $\phi(X_1, X_2, X_3)$ *used in previous examples and presented in Figure 1 will be represented as VDG as showed in Figure 6. The key of each pair is the value (circle) and gives access to a list (rectangle of rounded corner) of grains (pair of indexes storing a pattern of repetition). It can be seen in Figure 6 that each value is stored only once. Some values appear only once and the corresponding entry contains a single grain (is the case of* 1*) with* 5 *as starting and ending index. The rest of values are repeated. For example,* 0.1 *is the value corresponding to indexes* 0 *and* 11*. As this indexes are not consecutive must be stored in two different grains. Values* 0.2 *and* 0.5 *appear in consecutive indexes and their corresponding grains capture the sequences of repetitions.*

An improvement consists in considering a value as default, an hence it is not stored explicitly. The default value could be, for instance, the most common value. However, for simplicity here we will always consider 0.0 as default.

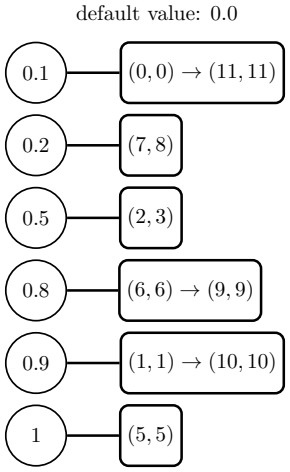

Figure 6: $\phi(X_1, X_2, X_3)$ as VDG

Assume that you want to access the value associated with index 6. This access involves searching among the dictionary the value containing this index in the paired list. Therefore, the dictionary is traversed using values (keys of the entries). It can be noticed that this search can be made in a parallel way. If the search does not produce any result then the default value is returned. This is the case for index 4 corresponding to 0.0.

Proposition 3 (Memory space for a VDG representing a potential)**.** *Let VDG_ϕ be the representation of $\phi(\mathbf{X})$. Let assume d represents the number of different values in the potential (discarding the default value). The number of grains for each value are denoted by $g_1 \dots g_d$. Then the amount of memory required is estimated as follows.*

$$memory(VDG_\phi) = N \cdot s_v + s_f + s_{dict} + d \cdot (s_f + s_{list}) + \sum_{j=1}^{d} 2 \cdot g_j \cdot s_i \quad (10)$$

The terms in Equation (10) consider the sizes for: variables; storage for default value; dictionary; values and lists; and grains with 2 indexes per grain. The number of grains for each value will depend of the sequences of repetitions. It will be lower as long as the sequences are longer. Therefore, the critical point in this representation is the number of grains required, given by $\sum_{j=1}^{d} g_j$.

Example 9. *Let VDG_ϕ be the VDG from Example 8 with* 6 *different values to store in the dictionary and* 0.0 *as default value. Therefore, the dictionary stores* 6 *entries. The sequences of repetitions requires* 9 *grains. Then, the memory cost can be computed with the following expression:*

$$memory(VDG_\phi) = 3s_v + s_f + s_{dict} + 6 \cdot (s_f + s_{list}) + 9 \cdot 2 \cdot s_i \quad (11)$$

Using the concrete memory sizes described in Section 3 the complete amount of memory is 438 bytes (sizes of 1AD, PT and PPT are 262, 1000 and 910 respectively).

5.2. ***VDI: value-driven with indexes***

Even though the previous structure with grains is a compact representation for potentials, it could encode unnecessary information when repeated values are not in consecutive positions. This is the case of value 0.1 in Figure 6 which has associated to 2 grains of length 1: $(0, 0)$ and $(11, 11)$. Instead, we could consider to simply associate to this value the list of indexes $\{0, 11\}$. In doing so, the required memory for indices (in this case) will be reduced. Having this idea in mind, the following representation can be defined.

Definition 3 (Value-driven with indexes)**.** *Let ϕ be a potential defined over $\mathbf{X}$, then a value-driven with indexes (VDI) representing ϕ is a dictionary assigning to each value $v \in \phi$ a list of indexes L such that $\phi(\mathbf{x}_i) = v$ for each $i \in L$.*

As explained, the data structure for VDI is a dictionary of $< value, list < index >>$. Non repeated values are keys and the entries contain list of indexes. The idea behind this alternative consists of removing the duplication of indexes for grains representing isolated values (1-value sequences).

Example 10. *The potential $\phi(X_1, X_2, X_3)$ used before and described in Figure 1 will be represented as VDI as showed in Figure 7. The key of each pair is the*

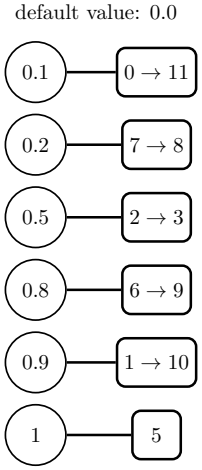

Figure 7: $\phi(X_1, X_2, X_3)$ as VDI

*value (circle). Now each key gives access to a list (rectangle of rounded corner) of indexes. As in VDG each value is stored only once (*0.0 *is the default value and it is not stored). Now there are no repetitions of indexes in the lists.*

Proposition 4 (Memory space for a VDI representing a potential)**.** *Let VDI_ϕ be the representation of $\phi(\mathbf{X})$. Let assume d represents the number of different values in the potential (discarding the default value). The number of indexes for each value are denoted by $i_1 \dots i_d$. Then the amount of memory required is estimated as follows.*

$$memory(VDI_\phi) = N \cdot s_v + s_f + s_{dict} + d \cdot (s_f + s_{list}) + \sum_{j=1}^{d} i_j \cdot s_i \qquad (12)$$

The terms of the Equation 12 consider the sizes for: variables; default value; dictionary; values and lists; and indices for non zero values. In this alternative, the main saving comes from the reduction in the number of values to store, but all the indexes (except those related to default value) must be stored.

Example 11. *Let VDI_ϕ be the VDI from Example 10 with* 6 *different values to store in the dictionary and* 0.0 *the default value. Therefore, the dictionary stores* 6 *pairs and the lists contain* 11 *indexes. Then, the memory cost can be computed with the following expression:*

$$memory(VDI_\phi) = 3s_v + s_f + s_{dict} + 6 \cdot (s_f + s_{list}) + 11 \cdot s_i \qquad (13)$$

Using the concrete memory sizes described in Section 3 the complete amount of memory is 410 bytes (a bit lower than VDG).

5.3. ***IDP: index-driven with pairs***

The problem of using dictionaries is that accessing the value associated to a given index could be inefficient: all its entries are traversed until the one with target index is reached. For solving this, a potential can be represented with two arrays: an array for the values and another one for the relations between indexes and values. Thus, we can formally define the following VBP alternative as follows.

Definition 4 (Index-driven with pairs)**.** *Let ϕ be a potential defined over $\mathbf{X}$, then a structure index-driven with pairs (IDP) representing ϕ is a pair of arrays: V and L. The non-repeated values in ϕ are stored in $V := \{v_0, v_1, \dots, v_{d-1}\}$. The array L is defined as follows.*

$$L := \{(i, j) : \phi(\mathbf{x}_i) = v_j \ \ and \ \ i = 0, 1, \dots, |\Omega_\mathbf{X}| - 1\} \qquad (14)$$

That is, IDP is based on the following two elements. First, an array storing the values (without repetitions, as before, and excluding 0.0 as default value). Secondly, an array of pairs (index in potential - index in array of values). The second index of the pair keeps the relation between indexes and values (this relation was maintained using dictionaries in previous alternatives).

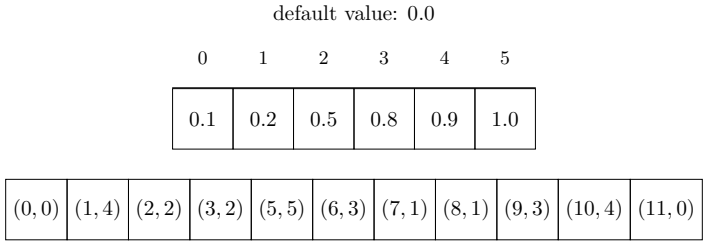

Figure 8: $\phi(X_1, X_2, X_3)$ as IDP

Example 12. *The representation as IDP of the potential $\phi(X_1, X_2, X_3)$ presented in Figure 1 is showed in Figure 8. As explained before, this representation is based on two coherent arrays. One of them stores non repeated values (except the default value) and the second one store the correspondences between potential indexes and the position of the corresponding value in the first array. Therefore two indexes are needed for each potential index storing a non zero value. A search for the value for a potential index requires a traversal through the array of pairs, until finding the target index or reaching the end of the array. Lets imagine the objective potential index is* 6*. Then the search will reach the pair* $(6, 3)$ *and the result will be the content of the position* 3 *in the array of values:* 0.8*. If the target is index* 5 *the search fails and then the default value would be returned. In this case this search can pe parallelized as well.*

Proposition 5 (Memory space for a IDP representing a potential)**.** *Let IDP_ϕ be the structure representing $\phi(\mathbf{X})$. Let assume d represents the number of different values in the potential (discarding the default value). The number of indexes corresponding to non-default value is p. Then the amount of memory is estimated as follows.*

$$memory(IDP_\phi) = N \cdot s_v + s_f + 2 \cdot s_{arr} + d \cdot (s_f) + 2 \cdot s_i \cdot p \tag{15}$$

The terms in Equation 15 considers the sizes for: variables; default value; both arrays; values; pairs of potential indexes and indexes of array value. This representation tries to use simple structures and prioritize the direct search on indices rather than on values.

Example 13. *Let IDP_ϕ be the VBP from Example 12 with* 6 *different values to store and* 0.0 *as default value. Therefore, the array of values stores* 6 *values and the array of indexes contain* 11 *pairs. Then, the memory cost can be computed with the following expression:*

$$memory(IDP_\phi) = 3s_v + s_f + 2 \cdot s_a + 6 \cdot s_f + 11 \cdot 2 \cdot s_i \tag{16}$$

Using the concrete memory sizes described in Section 3 the complete amount of memory is 326 bytes.

5.4. ***IDS: index-driven with sets***

The same underlying idea of VDIs is considered here: a set of indexes is associated to each possible value in the potential. For avoiding the problem of dictionaries in obtaining the value for a given index, we consider to use two different arrays instead. Thus a new index-based alternative is defined here below.

Definition 5 (Index-driven with sets)**.** *Let ϕ be a potential defined over $\mathbf{X}$, then a structure index-driven with sets (IDS) representing ϕ is a pair of arrays: V and L. Non-repeated values in ϕ are stored in $V := \{v_0, v_1, \ldots, v_{d-1}\}$. Let the array L be defined as $\{S_0, \ldots, S_{d-1}\}$ where each element $S_i \in L$ is a set of indexes satisfying $\phi(\mathbf{x}_k) = v_i$ for all $k \in S_i$.*

Intuitively, this alternative uses two arrays, both of them with a dimension equals to the number of different values discarding 0.0. The first one store sets of indexes and the second the values. And there is an implicit relation between them: the set of i^{th} position contains the indexes for i^{th} value.

Example 14. *The representation as IDS of the potential $\phi(X_1, X_2, X_3)$ presented in Figure 1 is showed in Figure 9. A search for the value for a potential*

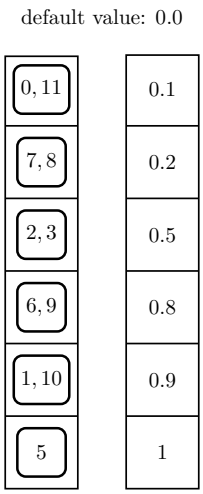

Figure 9: $\phi(X_1, X_2, X_3)$ as IDS

index requires a traversal through the array of sets, until finding a set containing the target index or reaching the end of the array. Lets imagine the objective potential index is 6*. Then the search will reach the set* $(6, 9)$ *(stored in position 3). Then result will be the content of the position* 3 *in the array of values:* 0.8*. If the target is index* 5 *the search fails and then the default value would be returned. The search through sets can be made in parallel.*

Proposition 6 (Memory space for a IDS representing a potential)**.** *Let IDS_ϕ be the structure representing $\phi(\mathbf{X})$. Let assume d represents the number of different values in the potential (discarding the default value). The number of indexes corresponding to non-default value is p. Then the amount of memory is estimated as follows.*

$$memory(IDS_\phi) = N \cdot s_v + s_f + 2 \cdot s_{arr} + d \cdot (s_{set}) + s_i \cdot p \tag{17}$$

The terms of the Equation (17) considers the sizes for: variables; default value; arrays of sets and values; sets; and indexes. This representation also tries to use simple structures and prioritize the direct search on indices (using set operations) rather than on values.

Example 15. *Let IDS_ϕ be the VBP from Example 14 with* 6 *different values to store and* 0.0 *as default value. Therefore, the arrays of sets and values have* 6 *elements. Then, the memory cost can be computed with the following expression:*

$$memory(IDS_\phi) = 3s_v + s_f + 2 \cdot s_{arr} + 6 \cdot s_{set} + 6 \cdot s_f + 11 \cdot s_i \tag{18}$$

Using the concrete memory sizes described in Section 3 the complete amount of memory is 474 bytes.

6. Empirical evaluation

The evaluation of VBPs memory sizes with respect to 1DA and trees (PTs and PPTs) is performed by considering two sets of Bayesian networks. The first set is taken from bnlearn ([26, 27]) and the second one from UAI competitions ([28, 29]). The quantitative information of these models is represented with the structures considered in the paper in order to compare their properties. The representations compared in the experiments are:

- 1DA: unidimensional arrays.
- PT: probability trees.
- PPT: probability trees with exact prune.
- VDG: value-driven search with dictionary and lists of grains.
- VDI: value.-driven search with dictionary and lists of indexes.
- IDP: index-driven search with arrays of values and indexes.
- IDS: index-driven search with array of sets of indexes and array of values.

network	nodes	arcs	min. st.	avg. st.	max. st.	parameters
cancer	5	4	2	2	2	20
asia	8	8	2	2	2	36
survey	6	6	2	2.33	2	37
sachs	11	17	3	3	3	267
child	20	25	2	3	6	344
alarm	37	46	2	2.83	4	752
win95pts	76	112	2	2	2	1148
insurance	27	52	2	3.29	5	1419
hepar2	70	123	2	2.31	4	2139
andes	223	338	2	2	2	2314
hailfinder	56	66	2	3.98	11	3741
pigs	441	592	3	3	3	8427
water	32	66	3	3.625	4	13484
munin1	186	273	2	5.33	21	19226
link	724	1125	2	2.53	4	20502
munin2	1003	1244	2	5.36	21	83920
munin3	1041	1306	2	5.38	21	85615
pathfinder	109	195	2	4.11	63	97851
munin4	1038	1388	2	5.44	21	97943
munin	1041	1397	2	5.43	21	98423
barley	48	84	2	8.77	67	130180
diabetes	413	602	3	11.34	21	461069
mildew	35	46	3	17.6	100	547158

Table 1: **bnlearn** Bayesian networks features

6.1. bnlearn networks

Some basic information about the Bayesian networks analyzed is included in Table 1: name; number of nodes; number of arcs; minimum, average and maximum number of states; and number of parameters. Networks are ordered according to the number of parameters (values to store within potentials).

The results for these networks are included in Table 2, which contains the ratios in the memory requirements for representing the quantitative information. That is, the memory estimation for each representation divided by the memory size required for 1DA. Therefore only ratios $r < 1$ indicates savings in memory space.

Some comments about the results for **bnlearn** networks are:

- PTs and PPTs always lead to ratios over 1 and very similar, except for **win95pts**: from 5.125 to 3.455 respectively (there are several potentials with repeated values where the prune operation substantially reduces the number of leaf nodes).
- in almost all networks the best representation is obtained with IDP. In the networks with a number of parameters under 8427 (**pigs**) the ratios

network	PT	PPT	VDG	VDI	IDP	IDS
cancer	2.494	2.494	1.987	1.883	**1.364**	2.091
asia	2.613	2.546	1.768	1.679	**1.274**	1.798
survey	2.84	2.84	2.125	1.989	**1.404**	2.357
sachs	3.918	3.836	2.292	2.084	**1.381**	2.829
child	3.423	3.391	1.843	2.087	**1.308**	1.973
alarm	3.721	3.665	1.407	1.554	**1.202**	1.449
win95pts	5.125	3.455	1.213	1.219	**1.094**	1.288
insurance	4.299	4.277	1.511	1.279	**1.095**	1.634
hepar2	4.431	4.421	2.711	2.361	**1.589**	3.417
andes	4.127	4.036	1.495	1.336	**1.183**	1.391
hailfinder	4.403	4.381	1.452	1.141	**1.099**	1.432
pigs	3.645	3.654	1.113	1.038	**0.937**	1.073
water	4.908	4.906	1.274	1.046	**0.766**	1.536
munin1	3.634	3.617	1.11	1.308	**0.729**	1.275
link	4.225	4.17	0.808	0.73	**0.695**	0.697
muni2	3.416	3.406	1.259	1.085	**0.791**	1.484
munin3	3.383	3.378	1.283	1.108	**0.8**	1.519
pathfinder	5.295	4.709	0.514	**0.369**	0.595	0.409
munin4	3.529	3.519	1.118	1.016	**0.76**	1.384
munin	3.545	3.503	1.173	1.003	**0.757**	1.363
barley	3.305	3.289	1.599	1.351	**1.283**	1.912
diabetes	2.671	2.671	0.365	**0.251**	0.289	0.323
mildew	2.124	2.124	0.151	0.117	**0.098**	0.171

Table 2: Ratios respect to 1DA for **bnlearn** networks

are bigger than 1 and ranging from 1.364 to 0.937. With higher number of parameters ratios are clearly under 1 except for **barley** network.

- The best ratios are those of **diabetes** and **mildew**. In both networks there are several potentials with a high number of repeated values. In the case of **diabetes** there are 25 variables with 7056 parameters but a reduced number of different values (44). The advantage of VDI representation is due to the number of indexes to store: 2040: IDP representation requires storing indexes as pairs and VDI as single values. In **mildew** there are several potentials with a high number of repeated values (39040 possible values but only 1756 different ones; 201300 parameters but only 4508 different; and 280000 parameters and 5023 different. In all these variables repeated values can not be pruned when using trees).

- VDI is the best representation for **pathfinder** and **diabetes**. In both cases there are a reduced number of different values respect to the complete number of parameters. In **pathfinder** there is a variable with 8064 parameters with 29 different values. As explained before for **mildew** the number of indexes to store is high (3520) and this explains the advantage of VDI alternative.

The difference between *small* and *large* networks is presented in Figure 10. The left part represents the behaviour for networks with a number of parameters up to 3741 (networks from **cancer** to **hailfinder**). The right part groups the results for the rest of them.

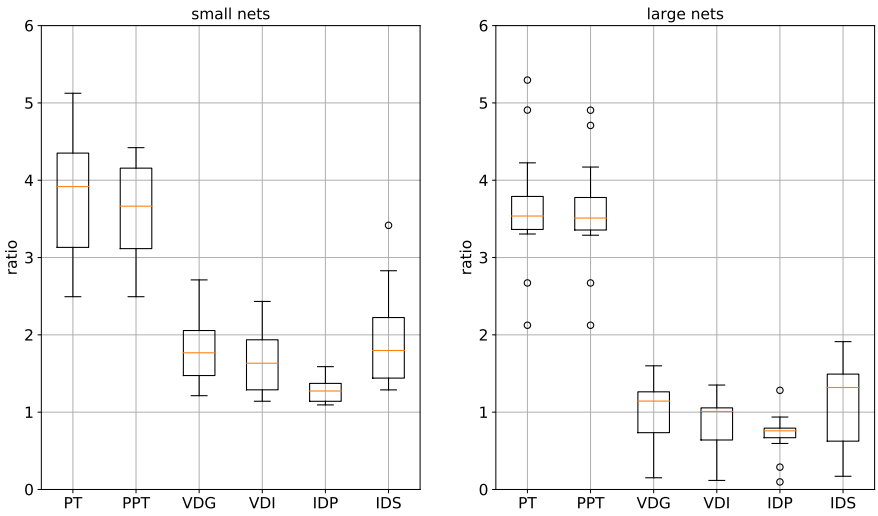

Figure 10: Graphical representation for **bnlearn** results

6.2. UAI competition networks

Table 3 gathers some basic information about the set of networks selected (the same information presented for **bnlearn** networks). The results for these networks are presented in Table 4.

The following conclusions can be outlined from these results:

- ratios for PTs and PPTs are always over 2.5, except for **BN_27**. For this network there is an important difference between PT (5.880) and PPT (0.045). In fact, in this network PPT is the best representation, although similar results are obtained for VDG. This network has 3025 nodes and the potentials for 1005 of them 3645 possible values but only 1. This leads to a very efficient representation with PPT (a single leaf node is enough) and VDG (a single grain covers all the range of indexes).

- IDP is the best representation for most of the networks, with ratios near to 1. The ratio is much more reduced for the last two networks, those with a very high number of parameters (more than 4 millions). In these two networks there are 303 potentials with 9464 parameters with two values: 0 (represented implicitly as default value) and 1. The sequences of repetitions can not be collapsed by prune operation and therefore PPT and PT offer similar ratios. In these potentials IDS representation is the

network	nodes	arcs	min. st.	avg. st.	max. st.	parameters
50-12-5	144	264	2	2	2	1058
50-14-5	196	364	2	2	2	1458
50-15-5	225	420	2	2	2	1682
50-16-5	256	480	2	2	2	1922
50-17-5	289	544	2	2	2	2178
50-18-5	324	612	2	2	2	2450
50-19-5	361	684	2	2	2	2738
90-20-5	400	760	2	2	2	3042
75-21-5	441	840	2	2	2	3362
75-22-5	484	924	2	2	2	3698
90-23-5	529	1012	2	2	2	4050
75-24-5	576	1104	2	2	2	4418
75-25-5	625	1200	2	2	2	4802
75-26-5	676	1300	2	2	2	5202
BN_126	512	768	2	2	2	5376
90-30-5	900	1740	2	2	2	6962
90-34-5	1156	2244	2	2	2	8978
90-38-5	1444	2812	2	2	2	11250
90-42-5	1764	3444	2	2	2	13778
90-46-5	2126	4140	2	2	2	16562
90-50-5	2500	4900	2	2	2	19602
BN_16	2127	3595	2	2.05	6	33659
BN_27	3025	7040	3	6	10	3698565
BN_22	2425	4239	2	18.743	91	4073904
BN_20	2843	5272	2	18.92	91	5009364

Table 3: **UAI competition** Bayesian networks features

most efficient. The average ratio presents a better result for IDP however, due to the existence of another 303 potentials with 910 parameters and 908 different values. In these potentials IDS representation requires much more space than IDP.

- all the alternative representations offer a competitive alternative respect to PTs and PPTs.

Figure 11 shows the summary view of the behaviour for *small* (from **50-12-5** to **BN_16**) and *large* ones, where for the alternative representations use to be under 1.

7. Conclusion

The examples of Bayesian networks considered in the experimental work show how, in all cases, PTs lead to a memory usage much higher than 1DAs. However, this does not make this representation useless. The true capacity of PTs is related to their efficiency for performing the operations required for inference tasks and its capability for being approximated.

network	PT	PPT	VDG	VDI	IDP	IDS
50-12-5	3.17	3.17	1.649	1.563	**1.207**	1.743
50-14-5	3.185	3.185	1.619	1.535	**1.194**	1.701
50-15-5	3.19	3.19	1.593	1.512	**1.183**	1.664
50-16-5	3.195	3.195	1.603	1.519	**1.186**	1.677
50-17-5	3.199	3.199	1.611	1.527	**1.19**	1.691
50-18-5	3.203	3.203	1.614	1.53	**1.191**	1.695
50-19-5	3.207	3.207	1.608	1.524	**1.188**	1.687
90-20-5	3.21	3.21	1.266	1.221	**1.044**	1.21
75-21-5	3.213	3.213	1.38	1.322	**1.092**	1.363
75-22-5	3.215	3.215	1.389	1.33	**1.095**	1.376
90-23-5	3.218	3.218	1.269	1.223	**1.044**	1.205
75-24-5	3.22	3.22	1.37	1.312	**1.087**	1.349
75-25-5	3.222	3.222	1.404	1.342	**1.101**	1.397
75-26-5	3.224	3.224	1.384	1.325	**1.093**	1.37
BN_126	4.166	4.073	1.232	1.182	**1.016**	1.182
90-30-5	3.32	3.23	1.277	1.228	**1.047**	1.216
90-34-5	3.234	3.234	1.252	1.207	**1.036**	1.182
90-38-5	3.328	3.238	1.257	1.211	**1.038**	1.189
90-42-5	3.241	3.241	1.253	1.315	**1.037**	1.183
90-46-5	3.243	3.243	1.257	1.211	**1.039**	1.19
90-50-5	3.245	3.245	1.253	1.206	**1.036**	1.183
BN_16	4.147	4.137	1.671	1.519	**1.167**	1.81
BN_27	5.88	**0.045**	0.06	0.541	1.011	0.555
BN_22	2.56	2.47	0.328	0.275	**0.195**	0.407
BN_20	2.549	2.451	0.34	0.296	**0.207**	0.44

Table 4: Ratios respect to 1DA for **UAI competition** networks

The approximation operation assumes loss of information. Intuitively, the idea is to group nearby values and replace them with their average (or some other measure), so that we are actually expanding the repetition patterns and, therefore, reducing the number of values to be stored. With this operation any algorithm that involves the use of PTs will become approximate and will ultimately offer non-exact solutions. In very complex problems it is always better to have at least one approximate solution (see [21, 22, 23, 24]).

The set of alternative representations offers compact representations with memory sizes below PTs and PPTs. Moreover, these alternatives can be approximated as well with an efficient method: join entries with similar values. Therefore, one of the additional benefits of PTs and PPTs is preserved with VBPs.

The next step with VBPs, and mainly with IDP will consist of defining efficient algorithms for the basic operations required for inference: *combination*, *marginalization* and *restriction*. The software used in this paper is implemented with **Scala** programming language. The functional programming paradigm (combined with object orientation) offered by **Scala** can be exploited

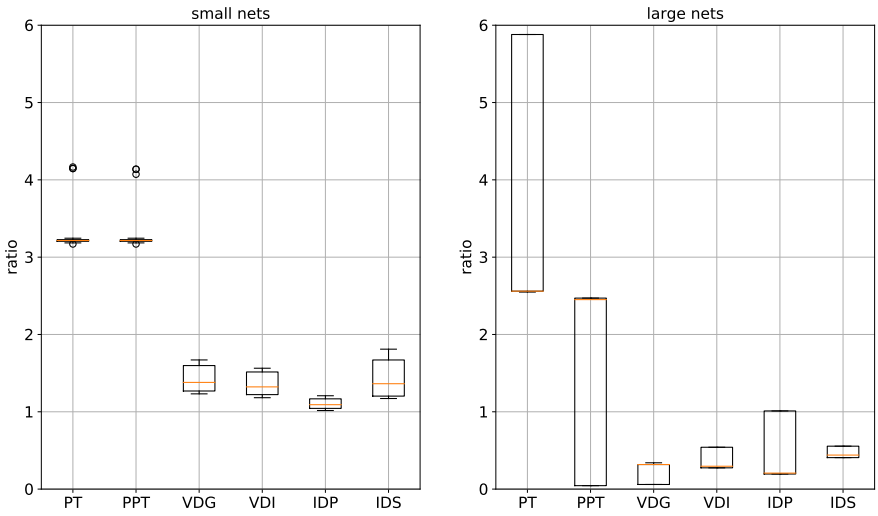

Figure 11: Graphical representation for **uai** results

for getting well defined operations easily converted into parallel ones on multi-core CPUs when possible. Some of these benefits are investigated in [30].

Acknowledgements

This research was jointly supported by the Spanish Ministry of Education and Science under projects PID2019-106758GB-C31 and TIN2016-77902-C3-2-P and the European Regional Development Fund (FEDER).

References

[1] D. Koller, N. Friedman, Probabilistic graphical models: principles and techniques, MIT press, 2009.

[2] J. Pearl, Bayesian networks: A model of self-activated memory for evidential reasoning, University of California (Los Angeles). Computer Science Department, 1985.

[3] J. Pearl, S. Russell, Bayesian networks, Computer Science Department, University of California, 1998.

[4] S. M. Olmsted, Representing and solving decision problems., Dissertation Abstracts International Part B: Science and Engineering 45 (3) (1984).

[5] R. A. Howard, J. E. Matheson, Influence diagram retrospective, Decision Analysis 2 (3) (2005) 144–147.

[6] A. P. David, Applications of a general propagation algorithm for probabilistic expert systems, Sttatistics and Computing 2 (1992) 25–36. doi:10.1007/BF01890546.

[7] J. Kwisthout, Most probable explanations in Bayesian networks: Complexity and tractability, Int. J. Approx. Reasoning 52 (9) (2011) 1452–1469. doi:10.1016/j.ijar.2011.08.003.
URL https://doi.org/10.1016/j.ijar.2011.08.003

[8] P. Dagum, M. Luby, An optimal approximation algorithm for Bayesian inference, Artificial Intelligence 93 (1) (1997) 1–27.

[9] R. Dechter, Bucket elimination: A unifying framework for probabilistic inference, in: Learning in graphical models, Springer, 1998, pp. 75–104.

[10] C. S. Jensen, U. Kjærulff, A. Kong, Blocking Gibbs sampling in very large probabilistic expert systems, International Journal of Human-Computer Studies 42 (6) (1995) 647–666.

[11] F. V. Jensen, T. D. Nielsen, Bayesian networks and decision graphs, Springer Verlag, 2007.

[12] Z. Li, B. D'Ambrosio, Efficient inference in Bayes networks as a combinatorial optimization problem, International Journal of Approximate Reasoning 11 (1) (1994) 55–81.

[13] A. L. Madsen, F. V. Jensen, Lazy evaluation of symmetric Bayesian decision problems, in: Proceedings of the 15th Conference on Uncertainty in AI, Morgan Kaufmann Publishers Inc., 1999, pp. 382–390.

[14] A. L. Madsen, F. V. Jensen, Lazy propagation: a junction tree inference algorithm based on lazy evaluation, Artificial Intelligence 113 (1-2) (2004) 203–245.

[15] J. Pearl, Evidential reasoning using stochastic simulation of causal models, Artificial Intelligence 32 (2) (1987) 245–257.

[16] R. D. Shachter, Evaluating influence diagrams, Operations research (1986) 871–882.

[17] R. D. Shachter, B. D'Ambrosio, B. Del Favero, Symbolic probabilistic inference in belief networks., in: AAAI, Vol. 90, 1990, pp. 126–131.

[18] C. Boutilier, N. Friedman, M. Goldszmidt, D. Koller, Context-specific independence in Bayesian networks, in: Proceedings of the 12th International Conference on Uncertainty in Artificial Intelligence, Morgan Kaufmann Publishers Inc., 1996, pp. 115–123.

[19] A. Cano, S. Moral, A. Salmerón, Penniless propagation in join trees, International Journal of Intelligent Systems 15 (11) (2000) 1027–1059.

[20] A. Salmerón, A. Cano, S. Moral, Importance sampling in Bayesian networks using probability trees, Computational Statistics & Data Analysis 34 (4) (2000) 387–413.

[21] M. Gómez-Olmedo, A. Cano, Applying numerical trees to evaluate asymmetric decision problems, in: T. Nielsen, N. Zhang (Eds.), Symbolic and Quantitative Approaches to Reasoning with Uncertainty, Vol. 2711 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2003, pp. 196–207. doi:10.1007/978-3-540-45062-7_16.
URL http://dx.doi.org/10.1007/978-3-540-45062-7_16

[22] A. Cano, M. Gómez-Olmedo, S. Moral, Approximate inference in Bayesian networks using binary probability trees, International Journal of Approximate Reasoning 52 (1) (2011) 49–62.

[23] R. Cabañas, M. Gómez, A. Cano, Approximate inference in influence diagrams using binary trees, in: Proceedings of the Sixth European Workshop on Probabilistic Graphical Models (PGM-12), 2012.

[24] R. Cabañas, M. Gómez-Olmedo, A. Cano, Using binary trees for the evaluation of influence diagrams, International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems 24 (01) (2016) 59–89.

[25] M. Arias, F. Díez, Operating with potentials of discrete variables, International Journal of Approximate Reasoning 46 (1) (2007) 166–187.

[26] M. Scutari, Bayesian network constraint-based structure learning algorithms: Parallel and optimized implementations in the bnlearn R package, Journal of Statistical Software 77 (2) (2017) 1–20. doi:10.18637/jss.v077.i02.

[27] M. Scutari, Learning Bayesian networks with the bnlearn R package, Journal of Statistical Software 35 (3) (2010) 1–22. doi:10.18637/jss.v035.i03.

[28] UAI 2016 Inference Competition (2016).
URL http://www.hlt.utdallas.edu/~vgogate/uai16-evaluation/

[29] UAI 2014 Inference Competition (2014).
URL http://www.hlt.utdallas.edu/~vgogate/uai14-competition/index.html

[30] A. R. Masegosa, A. M. Martinez, H. Borchani, Probabilistic graphical models on multi-core cpus using Java 8, IEEE Computational Intelligence Magazine 11 (2) (2016) 41–54.