Table-based Probability Trees: a New Hybrid Potential Representation for Probabilistic Inference in Graphical Models

Francisco Bonillo FBONILLO@UGR.ES
Andrés Cano ACU@DECSAI.UGR.ES
Manuel Gómez-Olmedo MGOMEZ@DECSAI.UGR.ES
Serafín Moral SMC@DECSAI.UGR.ES
Departamento de Ciencias de la Computación e Inteligencia Artificial, Universidad de Granada, Granada, Spain

Editors: Gustau Camps-Valls, Manuele Leonelli and Gherardo Varando

Abstract

Probabilistic inference in graphical models, such as Bayesian and Markov networks, has traditionally relied either on tabular representations of conditional probability distributions (CPDs) or on probability trees. Tables offer straightforward marginalization and combination at the cost of exponential memory in the number of variables, while pure probability trees exploit zero entries and repeated subdistributions to achieve compact representations, but their arithmetic is subject to structural overhead when non-zero subtrees dominate. This paper introduces Table-based Probability Tree (TBPT) potentials as a hybrid potential representation that unifies both paradigms within a single, flexible data structure: a rooted ordered tree in which internal nodes are discrete random-variable selectors as usual, and terminal nodes are not only scalar values, but also table nodes storing multidimensional arrays covering the variables below a given branching point. After formalizing the operations to be applied in these potentials, empirical evaluation on standard UAI competition benchmarks is conducted by using the TBPT structure running exact inference via variable elimination (VE). This data structure is useful in cases in which potential have a high zero percentage. We propose to compute the total set of zeros in advance with a previous Boolean propagation.

Keywords: Table-based probability trees, Bayesian networks, Markov networks, exact inference, variable elimination, probability trees, table nodes, Boolean propagation, lifting.

1. Introduction

Complex problems that require the treatment of uncertainty and probabilistic reasoning have been efficiently represented via probabilistic graphical models (PGMs), such as Bayesian and Markov Networks (Pearl, 1988; Lauritzen, 1996). A central computational task in these models is exact inference: computing the marginal or conditional distribution of a set of query variables given observed evidence.

The most general exact inference algorithm in this setting is Variable Elimination (VE) (Zhang and Poole, 1994), which proceeds by successively combining and marginalizing Conditional Probability Tables (CPTs) or potentials (in general) according to a given elimination order. The computational cost of VE is dominated by the size of the intermediate

© 2026 F. Bonillo, A. Cano, M. Gómez-Olmedo & S. Moral.

potentials created during this process, which in turn depends critically on how they are represented.

The most common potential representation is tabular, storing values in a multidimensional array indexed by the states of the involved variables, as adopted by widely-used libraries such as PGMPY (Ankan and Textor, 2024). Tabular representations afford $\mathcal{O}(1)$ random access but grow exponentially with the number of variables, rendering them impractical for dense potentials involving many discrete variables.

Probability trees (Boutilier et al., 1996; Cano et al., 2000; Salmerón et al., 2000; Cano and Moral, 2002) provide a compact alternative to tables, where root-to-leaf paths encode variable assignments and leaves store potential values. Pruning can collapse uniform subtrees, while shared structures can further reduce the representation. Thus, probability trees may be exponentially smaller than tables when potentials contain repeated or zero entries, as often occurs in deterministic or context-specific relationships in CPTs (Boutilier et al., 1996) and Markov potentials with hard constraints (Lauritzen et al., 1990).

Despite their appeal, probability trees incur structural overhead during arithmetic: combining two trees whose variable sets differ may produce a tree larger than either original, and marginalizing a variable requires traversing the entire tree. Pure tables, in contrast, exploit hardware-level array operations and are generally faster when potentials are dense. The key insight motivating this work is that neither representation is universally optimal and to fill the gap between both: real CPTs and Markov potentials exhibit mixed structure, with regions of regularity (suitable for tree compression) and regions of density (suitable for tables).

A wider body of work has explored alternative factor representations. Value-based potentials (VBPs) (Gómez-Olmedo et al., 2021) exploit repeated values regardless of whether those repetitions correspond to context-specific independence patterns, achieving memory savings in settings where probability trees offer no advantage. Binary probability trees (Cano et al., 2011) refine the standard tree by splitting branches into binary choices, enabling finer-grained representation of such independences. Recursive probability trees (RPTs) (Cano et al., 2015) provide a flexible learnable variant of probability trees that can be fitted directly from data. Other graphical models such as influence diagrams are used for lazy evaluation techniques (Cabañas et al., 2016), which defer combinations until a variable is actually to be marginalized, reducing unnecessary work in the presence of many local structure patterns.

A different family of representations is based on decision diagrams. Ordered Binary Decision Diagrams (OBDDs) (Bryant, 1986) provide canonical representations of Boolean functions and have been widely used for reasoning under deterministic constraints. Algebraic Decision Diagrams (ADDs) (Bahar et al., 1993) extend OBDDs by allowing arbitrary numerical values at terminal nodes. Zero-suppressed Decision Diagrams (ZDDs) (Minato, 1993) are particularly effective for sparse combinatorial structures where absent assignments dominate. More expressive graphical decompositions include AND/OR Multi-Valued Decision Diagrams (AOMDDs) (Mateescu et al., 2008), which exploit both decomposition and context-specific independence to obtain compact representations for graphical models. Sentential Decision Diagrams (SDDs) (Darwiche, 2011) provide canonical decomposable representations supporting efficient logical transformations, while Probabilistic SDDs (PSDDs) (Kisa et al., 2014) and Conditional PSDDs (Dang et al., 2018) extend this framework

to probabilistic modeling and tractable inference. The latter PSDD representations belong to a subclass of arithmetic circuits, for which tractable multiplication and marginalization algorithms have been efficiently developed (Shen et al., 2016).

This paper intend to contribute in several aspects. We define a hybrid potential representation called Table-based Probability Trees (TBPT), a probability tree in which any subtree can be replaced by a multidimensional array –that is, table node can be terminal nodes, not only scalar values– and establish the formal correctness of the combination and marginalization operations on this structure. We introduce several lifting strategies in the construction of this trees, that determine when and how portions of the tree are converted into table nodes. We describe the complete Variable Elimination algorithm used though these potentials and develop a Boolean propagation framework that runs a forward-backward pass on a Boolean correspondence of the network to pre-filter zero-dense regions before VE, further reducing the work performed during inference. We present an empirical evaluation inference configuration (varying lifting strategy and Boolean propagation) against pure tree and tabular VE on multiple benchmark networks, reporting execution time and tree-size metrics.

The remainder of the paper is organized as follows. Sections 2–4 present the theoretical framework. Section 5 reports experimental results, and Section 6 concludes the research work.

2. TBPT representation: Probability Trees with Table Nodes

2.1. Notation and preliminaries. Node types and well-formedness

Let $\mathbf{X} = \{X_1, \ldots, X_n\}$ be a set of $n \in \mathbb{N}$ discrete random variables and each X_i takes values in $\Omega_{X_i} = \{0, 1, \ldots, r_i - 1\}$, where $r_i := |\Omega_{X_i}|$ is its cardinality, i.e., the possible number of states of such variable. Let $k, a_1, \ldots, a_k \in \mathbb{N}$, with $k \leq n$ and $a_1 < \cdots < a_k$. Let $\mathbf{S} = \{X_{a_1}, \ldots, X_{a_k}\} \subseteq \mathbf{X}$ and $\Omega_{\mathbf{S}} := \prod_j \Omega_{X_{a_j}}$. A *potential* (also known as factor) over such subset $\mathbf{S} \subseteq \mathbf{X}$ is a function $\phi : \Omega_{\mathbf{S}} \to \mathbb{R}_0^+$.

The Boolean potential associated to a given potential ϕ will be denoted as ϕ^B, and it is defined as follows:

$$\phi^B : \Omega_{\mathbf{S}} \to \{0,1\}, \phi^B(x_{a_1}, \ldots, x_{a_k}) = \begin{cases} 0, \text{ if } \phi(x_{a_1}, \ldots, x_{a_k}) = 0 \\ 1, \text{ if } \phi(x_{a_1}, \ldots, x_{a_k}) \neq 0 \end{cases},$$

understanding that the binary codomain consists of the Boolean values False (0) and True (1). Partial or complete assignments are written as $\{X_{a_1} \mapsto x_{a_1}, \ldots, X_{a_k} \mapsto x_{a_k}\}$ and are used to represent evidence, restricting the potential to a subtree or subtable.

A table-based probability tree (TBPT) $\mathcal{T}$ is a tree which represents a potential ϕ consisting of a root node and a set of variables $\mathbf{S} = \{X_{a_1}, \ldots, X_{a_k}\} \subseteq \mathbf{X}$ (semantic scope). The root node and possibily more underlying nodes can be of three kinds. A scalar node is a terminal node storing a single scalar $v \in \mathbb{R}$ (the leaf of a classical probability tree). A table node is another possible terminal node (new in this hybrid data structure) storing a multi-dimensional array $\mathbf{T} \in \mathbb{R}^{r_{b_1} \times \cdots \times r_{b_{k'}}}$ indexed by the ordered variable sequence $\mathbf{S}' = (X_{b_1}, \ldots, X_{b_{k'}}) \subseteq \mathbf{S}$, for $k', b_1, \ldots, b_{k'} \in \mathbb{N}$, with $k' \leq k \leq n$, which is the active variable scope. The last type of node is the branch node, which is an internal node labelled with

a variable $Y \in \mathbf{S}$ and that have one child node for each state $i \in \Omega_Y$. Each child can be another branch node, or a terminal one (table or scalar node). Moreover, its active variable scope can be computed recursively considering the name attribute of each branch node and considering successively its children, so that implicit variables (those that does not appear in the nodes of a TBPT but in fact they are semantically in the potential) can be identified.

The TBPT is well-formed if every branch node labels a variable in $\mathbf{S}$, no variable appears twice on any root-to-leaf path, every table node's array shape matches the corresponding cardinalities and the same cardinality issue related to child nodes of all branch nodes. Moreover, as later we will cover, structural operations on these TBPT potentials (lifting and pruning) preserve well-formedness by construction and also the arithmetic operations (restriction, combination, marginalization), by definition and verifiable by structural induction.

As a remark, in relation to the existing probability trees and tabular representations, this hybrid data structure acts in the extreme cases as generalization of them. That is, when no table node exists and every branch node leads either to another branch node or to a scalar node, the TBPT coincides with a standard probability tree and when the root itself is a table node, the TBPT is equivalent to a tabular potential. As another observation, a potential ϕ can be represented by many different TBPTs, due to the choice of ordering in internal nodes can vary or the way that context-independence may appear can prune some subtrees, or even some subtrees can be substituted by a table node.

2.2. Constructing a TBPT

The same as for building probability trees (Salmerón et al., 2000), the starting point of the proposed representation is a standard tabular potential, stored as a multidimensional array containing the values of a conditional probability table (CPT) or Markov potential. From this dense representation, a complete probability tree is recursively constructed by selecting, at each internal node, one variable according to a predefined variable-selection heuristic. Then, a pruned probability tree is obtained and after an additional lifting operation, the final TBPT is built. This construction process is exemplified in Figure 1.

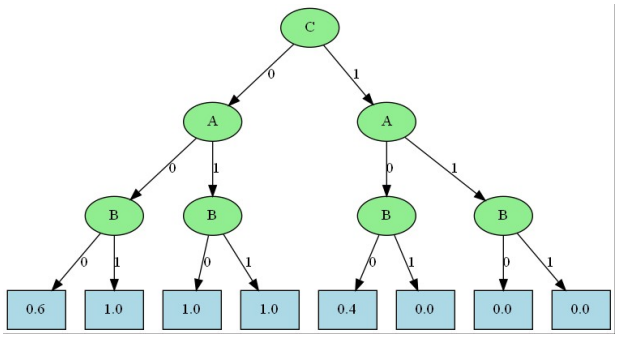
Step 1 – Complete probability tree. The tree is built recursively by selecting one variable at each internal node according to a variable-selection heuristic. Following Quinlan (1986)'s ID3 algorithm, we use information measures derived from Shannon entropy (Shannon, 1948). Given a variable X and remaining variables $\mathbf{Y} = \mathbf{X} \setminus \{X\}$, the joint entropy decomposes as $H(X, \mathbf{Y}) = H(X) + H(\mathbf{Y} \mid X)$, yielding two natural criteria: minimize $H(X)$ (marginal entropy of X) or maximize $H(\mathbf{Y} \mid X)$ (conditional entropy of the remaining variables given X). The first places variables with concentrated marginal distributions at the top of the tree, producing large regions of repeated values after recursive decomposition; the second is equivalent but emphasizes structural information about the remaining variables. We use *minimizing marginal entropy* as the preferred heuristic, since it is simpler to compute and tends to group zero-valued configurations within the same subtrees, which is precisely what benefits the subsequent pruning and lifting steps.

Step 2 – Pruning. The tree is traversed bottom-up. Whenever all children of a branch node are scalar nodes storing the same value $v \in \mathbb{R}$, the entire subtree is replaced by a

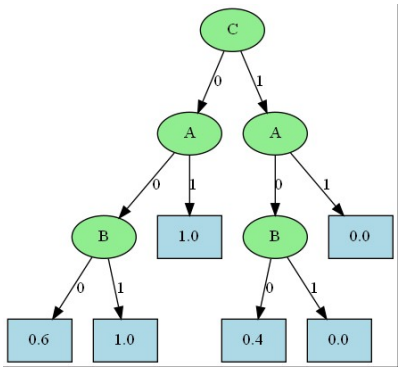
single scalar node with value v. This replacement may expose new pruning opportunities higher in the tree, so the process continues recursively until no further reductions are possible. The effectiveness of pruning depends on the variable ordering: the marginal-entropy criterion groups equal values within the same subtrees, increasing sibling-leaf equality and thus pruning efficiency, particularly for zero-valued configurations.

A	B	C	$\phi(A,B,C)$
0	0	0	0.6
0	0	1	0.4
0	1	0	1.0
0	1	1	0.0
1	0	0	1.0
1	0	1	0.0
1	1	0	1.0
1	1	1	0.0

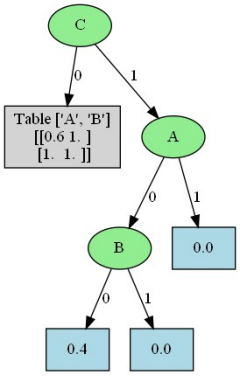
(a) Tabular potential



(b) Complete probability tree



(c) Pruned probability tree



(d) Table-based probability tree

Figure 1: Example of the step-by-step construction of a table-based probability tree from a tabular potential $\phi(A,B,C) = P(C \mid A,B)$ representing a CPD.

Step 3 – Lifting. The pruned tree is transformed into the final TBPT by a lifting operation that replaces dense non-zero regions with table nodes. We prevent zeros from appearing in any table: a subtree is lifted to a table node only when all its terminal descendants are non-zero. This is determined by inspecting the Boolean tree ϕ^B (same structure as ϕ): whenever a subtree of ϕ^B evaluates to all True (all non-zero), the corresponding subtree in ϕ is converted into a table node. If any terminal in the subtree is zero, lifting is stopped at that level to preserve the structural sparsity encoded by the tree. The intuition is that zero values act as absorbing elements in multiplication ($v \cdot 0 = 0$), so keeping them as explicit scalar nodes enables symbolic short-circuiting of combinations without materializing large dense tables. This construction is illustrated in Figure 1.

This hybridization allows arithmetic operations such as combination (pointwise product) and marginalization (sum-out) to exploit efficient array-based implementations and still benefiting from the compact sparse representation provided by probability trees which lets zeros to be propagated symbolically without explicitly materializing large dense tables.

2.3. Arithmetic Operations on TBPT Potentials

We define three fundamental potential operations in this new data structure: restriction, combination and marginalization. In what follows $e = \{X_{a_j} \mapsto v_j\}_{j \in J}$ denotes an evidence assignment of a set of variables identified according to the index set J, X a variable to be marginalized, and ϕ_1, ϕ_2 two TBPT potentials over variable scopes $\mathbf{S}_1 = \text{vars}(\phi_1)$ and $\mathbf{S}_2 = \text{vars}(\phi_2)$.

We adopt the arrow notation $\phi^{\downarrow \text{vars}(\phi) \setminus X}$ denoting the potential obtained by marginalizing the variable X out of potential ϕ, and $\phi_{|_e}$ denotes restriction to evidence e.

Restriction (Algorithm 1) traverses the tree from the root and when a branch node for an evidence variable X_i is encountered, it is replaced by its v_i-th child. Table nodes are restricted by array slicing so that the axis corresponding to X_i is indexed at position v_i and removed from the shape. After restriction the scope shrinks to $\mathbf{S} \setminus \{X_{a_j}\}_{j \in J}$, and the well-formedness of the resulting TBPT is maintained because no variable labels two nodes on the same path and the array shapes remain consistent.

Algorithm 1: Restriction: $\phi_{|_e}$

Input: Node node (firstly, the root of the TBPT), evidence $e = \{X_{a_j} \mapsto v_j\}_{j \in J}$
Output: Restricted node (the root of the restricted TBPT)

```
if node is scalar_node then return Value of node
if node is table_node then return Restricted array slicing for evidence variable's axis
if node ∈ e then
    state ← e[node.varname]
    return Restriction(node.children[state], e)
else
    c' ← [Restriction(c_i, e) for c_i ∈ node.children]
    return branch_node(node, c')
```

The combination / product (Algorithm 2) of two TBPT potentials ϕ_1 and ϕ_2 over variables $\mathbf{S}_1$ and $\mathbf{S}_2$, respectively, is defined pointwise over the union of scopes:

$$(\phi_1 \otimes \phi_2)(\mathbf{x}) = \phi_1(\mathbf{x}|_{\mathbf{S}_1}) \cdot \phi_2(\mathbf{x}|_{\mathbf{S}_2}), \quad \forall \mathbf{x} \in \prod_{X \in \mathbf{S}_1 \cup \mathbf{S}_2} \Omega_X \tag{1}$$

The recursive implementation traverses both trees simultaneously, promoting variables of branch nodes encountered in either tree to the top when they are absent from the other tree. Terminal nodes implement the product via overloaded operators that handle all combinations: product of leaf nodes or between a leaf node and a table is immediate, and the product of two table nodes involves constructing a new potential over the union of their variables and multiplying all compatible entries sharing the same assignments for common variables. Internally, multidimensional arrays are aligned by expanding missing dimensions and rearranging variable orders before performing tensor multiplication, as it is the case of our implementation based on the one from pgmpy (Ankan and Textor, 2024) using NumPy's einsum operation.

The marginalization (Algorithm 3) of one variable X out of potential ϕ, in terms of projecting over the remaining variables $\mathbf{Y} := \text{vars}(\phi) \setminus X$, can be defined as:

$$\phi^{\downarrow \mathbf{Y}}(\mathbf{y}) = \sum_{x \in \Omega_X} \phi(\mathbf{y}, x), \quad \forall \mathbf{y} \in \prod_{Y \in \mathbf{Y}} \Omega_Y \tag{2}$$

Algorithm 2: Combination: $\phi_1 \otimes \phi_2$

Input: Nodes n_1, n_2 (roots of ϕ_1, ϕ_2); cardinalities $\{r_i \mid i = 1, \ldots, n\}$
Output: Root of $\phi_1 \otimes \phi_2$

if n_i *is scalar_node with value* 0 **then return** scalar_node(0) ▷ $i \in \{1, 2\}$
if n_i *is scalar_node with value* 1 **then return** copy(n_j) ▷ $i, j \in \{1, 2\}, i \neq j$
if n_1 *is terminal* & n_2 *is terminal* **then return** $n_1 \otimes n_2$ ▷ `Direct product`
else if n_1 *is terminal* & n_2 *is not terminal* **then**
  return Combination$(n_2, n_1, \text{cardinalities})$
else
  $X \leftarrow n_1$.varname
  $\mathbf{c} \leftarrow \left[\text{Combination}(c_i,\ \text{Restriction}(n_2, \{X \mapsto i\}), \text{cardinalities})\right]_{i=0}^{r_X - 1}$
  return branch_node(X, $\mathbf{c}$)

Firstly, when the root is a branch node for X, its children are summed recursively using a tree-addition operator $\oplus$ collapsing the variable X, which is fully analogous to the combination algorithm but changing product to sum. Specifically, two terminal nodes are added component-wise as expected between scalar values or tables, and in case a terminal node is to be added to a branch node, the children are recursively considered for the sum. Secondly, when the root is a branch node for $Y \neq X$, marginalization is propagated into each child. Finally, values from scalar nodes are multiplied by r_X (the variable is absent, so the sum duplicates the value r_X times) [1] and table nodes are summed along the axis corresponding to the variable X.

Note that the marginalization of more than one variable out can be approached by marginalizing each of them one after another.

Algorithm 3: Marginalization: $\phi^{\downarrow \text{vars}(\phi) \setminus \{X\}}$

Input: Node node (root of ϕ); variable X; cardinalities $\{r_i \mid i = 1, \ldots, n\}$
Output: Root of the marginalized potential

if node *is scalar_node* **then return** scalar_node(node.value $\cdot\, r_X$)
if node *is table_node* **then return** node.values.sum(axis = index(X))
if node.varname $= X$ **then**
  return $\bigoplus_{i=0}^{r_X - 1}$ node.children[i] `/* sum children */`
else
  $\mathbf{c}' \leftarrow$ [Marginalization$(c_i, X, \text{cardinalities})$ for $c_i \in$ node.children]
  return branch_node(node.varname, $\mathbf{c}'$)

Proposition 1 (Correctness) *For any well-formed TBPT $\mathcal{T}$ representing ϕ, the operations $\phi_{|_e}$, $\phi_1 \otimes \phi_2$, and $\phi^{\downarrow \text{vars}(\phi) \setminus \{X\}}$ return correct values for every assignment, regardless of which subtrees have been pruned or lifted.*

Proof By structural induction on the table-based probability tree $\mathcal{T}$, the base cases are as follows: scalar nodes store the correct value by construction, and table nodes index the correct entry by array indexing (restriction), broadcast (combination), or axis summation

1. Multiplying a scalar leaf by r_X is valid in the context of VE algorithm, which first combines the relevant potentials and then marginalizes the variable being eliminated, X, which is always semantically within the scope of the potential being marginalized, even if it has been structurally removed by pruning.

(marginalization). As inductive step, branch nodes recurse correctly because restriction to an evidence variable selects the correct child, combination promotes the branch-node variable to the top and restricts the other operand (preserving pointwise equality), and marginalization sums the recursively computed children. The scalar-node marginalization formula $v \cdot r_X$ is correct because the pruned subtree represents a region where ϕ is constant in X and due to the VE idiosyncrasy, so $\sum_{x \in \Omega_X} v = v \cdot r_X$. ■

3. Variable Elimination on TBPT Potentials

3.1. Algorithm overview

Variable Elimination (VE) constitutes the standard dynamic-programming approach for exact inference in PGMs, to solve the computation problem of the marginal or conditional distribution of a set of query variables given observed evidence. It exploits the factorized structure of Bayesian and Markov networks by avoiding the explicit construction of the full joint probability distribution, whose size grows exponentially with the number of variables.

Algorithm 4 describes VE on a list of TBPT potentials. Particularly, after restricting all potentials to the observed evidence, it eliminates each non-query variable X_{a_j} according to a given elimination order σ. It collects all potentials whose scope contains the variable X_{a_j}, combines them, optionally applying an adjustment step (pruning or pruning and lifting, or making a structural reduce on tables similarly as pruning in complete trees), marginalizes the variable out with another optional adjustment, and re-inserts the result.

Algorithm 4: Variable Elimination

Input: TBPT Potentials $\mathcal{F}$, query variables $\mathbf{Q}$, evidence e, elimination order $\sigma = (X_{a_1}, \ldots, X_{a_m})$, adjustment type α
Output: Unnormalized posterior over $\mathbf{Q}$

1 **foreach** $\phi_i \in \mathcal{F}$ **do** $\phi_i \leftarrow \phi_{i|_e}$ ▷ Apply restriction e to all potentials
2
3 **for** $j \leftarrow 1$ **to** m **do**
4 $\quad \mathcal{F}_{X_{a_j}} \leftarrow \{\phi \in \mathcal{F} \mid X_{a_j} \in \text{vars}(\phi)\}$
5 $\quad \mathcal{F} \leftarrow \mathcal{F} \setminus \mathcal{F}_{X_{a_j}}$
6 $\quad \psi \leftarrow \bigotimes_{\phi \in \mathcal{F}_{X_{a_j}}} \phi$ ▷ Optional adjustment α after every combination
7 $\quad \psi' \leftarrow \psi^{\downarrow \text{vars}(()\psi) \setminus \{X_{a_j}\}}$ ▷ Marginalization + Optional adjustment α
8 $\quad \mathcal{F} \leftarrow \mathcal{F} \cup \{\psi'\}$
9 **end**
10 **return** $\bigotimes_{\phi \in \mathcal{F}} \phi$ ▷ Multiply last potentials only containing query variables

3.2. Elimination order heuristic

In order to get an elimination order for VE, we implement a greedy heuristic that at each step, the variable X to be eliminated is the one minimizing the net size increase, i.e. the number of values of the potential to be added to the active potential list $\mathcal{F}$ subtracting the values of each individual potential recently combined in that step. Mathematically, the variable X to eliminate in each step is the one minimizing:

$$\prod_{Y \in T_X \setminus \{X\}} r_Y - \sum_{\substack{\phi \\ \text{vars}(\phi) \ni X}} \prod_{Y \in \text{vars}(\phi)} r_Y \tag{3}$$

where $T_X = \bigcup_{\phi \in \mathcal{F}} \{\text{vars}(\phi) \mid \text{vars}(\phi) \ni X\}$ is the set of variables appearing in some of the cliques containing the variable X, and r_Y represents the cardinality of any variable Y.

3.3. Clique Tree Construction

Apart from the VE algorithm, it is also known the clique tree propagation Lauritzen and Spiegelhalter (1988) for exact inference, in which a secondary structure called clique or junction tree is used during the VE process to enable making inference with more than one query variable. What will be useful for us for the Boolean propagation scheme we propose is the way in which the cliques are built in order to triangulate the graphical model.

For every variable X in a given elimination order, the triangulation algorithm computes a clique C as the union of all current scopes containing X. This clique represents the set of variables of the intermediate potential that would appear during VE. The separator associated with the elimination is then $S = C \setminus \{X\}$, that is, the variables remaining after eliminating X. Scopes are updated by removing those containing X and adding the separator S, reproducing the fill-in effect of VE. Each clique is attached to the previously generated clique that shares all variables in its separator. The query variable and an artificial empty root clique are appended at the end.

The algorithm returns the generated cliques, the inverse incorporation order to the clique tree structure, and the parent-child structure of the clique tree.

4. Boolean Propagation Framework

In many real-world applications of Bayesian and Markov potentials, a significant fraction of entries are zero, reflecting logical constraints, deterministic relationships or hard constraints in Markov random fields. These zeros propagate through the VE steps: a product of two potentials has a zero in a subtree wherever either potential has zero entries, so many entries of intermediate potentials may be partly zero even if neither original potential is entirely zero. The Boolean propagation framework identifies, before running VE, which entries of the original potentials $\mathcal{F}$ are guaranteed to contribute zero to the final result, and filters them out before inference.

The complete Boolean propagation algorithm follows a forward-backward schedule on the clique tree induced by triangulation. In particular, given an elimination order $\sigma = (X_{a_1}, \ldots, X_{a_m})$ that will be applied for VE, the first step is to convert all the potentials in $\mathcal{F}$ into Boolean potentials to obtain $\mathcal{F}^B$, restricting to evidence if needed. Note that the operations on Boolean TBPT are the same as numerical ones, with the marginalization being the logical *or*, and the combination the logical *and*.

It is well known that each elimination order σ gives rise to a join tree with hyper-nodes $T_{X_j} = \bigcup_{\phi^B \in \mathcal{F}^\mathcal{B}} \{\text{vars}(\phi^B) \mid \text{vars}(\phi^B) \ni X_j\}$ Shenoy and Shafer (1990). Each hypernode has as parent one of the nodes T_{X_i} corresponding to the deletion of variable X_i (deleted after X_j), which contains all the variables in $T_{X_j} \setminus \{X_j\}$, with separator $S(X_i, X_j) = T_{X_i} \cap T_{X_j}$.

In this join tree we can assign each potential ϕ to one of the hyper-nodes containing the variables of ϕ and carry out a classical propagation algorithm with an inwards (from leaves to root) and outwards (from root to leaves) phases. Working with Boolean potentials, we can simplify the probabilistic propagation algorithms taking into account that potentials are idempotent, and the three stages are as follows:

Forward pass (Boolean VE). Assign to each hyper-node T_{X_i} the potential ψ computed in line 6 of Algorithm 4 when deleting variable X_i. Let us call this potential ψ_{X_i}.

Backward pass. From root to leaves update potential ψ_{X_i} by combining it with $\psi_{X_j}^{\downarrow S(X_i,X_j)}$ where T_{X_j} is the parent of T_{X_i}. The potential in the root is left unchanged.

Filtering original potentials. Once that Boolean potentials propagation is carried out, we can update the original potentials of the probabilistic propagation as follows: for each probabilistic potential ϕ with scope $\text{vars}(\phi)$, let X_i be the first variable in σ such that $\text{vars}(\phi) \subseteq T_{X_i}$. We can update ϕ to $\phi \otimes \psi_{X_i}^{\downarrow \text{vars}(\phi)}$, setting to zero all entries identified as globally zero by the Boolean pass. The final result of VE on the updated potentials is identical to that on the originals, as Boolean propagation only zeroes entries that could not contribute to the posterior. The updated potentials, however, have more zeros and are therefore better suited to the TBPT representation.

5. Experimental Evaluation

5.1. Experimental setup

The evaluation uses networks drawn from the UAI 2022 Competition MAR-2 Final Evaluation Benchmarks[2]. We restrict attention to networks satisfying three criteria simultaneously: all variables have cardinality 2 (binary), the global zero percentage of the parameter tables have to exceed 30%, and the total number of parameters does not exceed 9 000. These criteria select networks that are challenging enough to stress test the lifting and Boolean propagation mechanisms (high sparsity) while remaining tractable within a time budget.

The selected networks belong almost entirely to the PROMEDAS `or_chain` family [3], whose instances are Markov networks for medical diagnosis originally built from QMR-DT Bayesian networks with noisy-OR structure. Structural and parametric statistics for all selected networks are provided in Table 2 (see Appendix A).

All experiments compute a single marginal probability query (first variable, `var_1`) given the evidence encoded in each network. VE is run under five configurations that differ in how the probabilistic potentials are represented and whether Boolean propagation is applied as a pre-processing step.

Inference case 1 (pgmpy tabular baseline). It represents a standard tabular VE using the PGMPY library.

Inference case 2 (TBPT collapsed to unique table). Each potential is built as a probability tree and immediately collapsed into a single root table node covering all variables — equivalent to the tabular representation of Case 1. No adjustment is applied after each VE operation.

Inference case 3 (TBPT with no-zero lifting). Potentials are built as probability trees, pruned, and then lifted so that no zero appears in any table node. Here we see if the proposed hybrid structure help us with zeros only as scalar nodes and not in table nodes to try to simplify operations such as combination. No further adjustment (pruning or lifting) is applied after each VE operation.

2. https://uaicompetition.github.io/uci-2022/results/benchmarks/
3. https://ics.uci.edu/~ihler/uai-data/

Inference case 4 (TBPT as pure probability trees – pruning at each step). Potentials are built as probability trees, pruned at the beginning and after every VE operation.

Inference case 5 (Boolean propagation and pure probability trees – pruning at each step). Boolean propagation is applied as a pre-processing step. The floating-point potentials are pure probability trees (TBPT without lifting): they are pruned after creation and after each VE step. Boolean potentials follow the same structure and prune-only adjustment. This case isolates the benefit of Boolean pre-filtering without the possible overhead of lifting.

Inference case 6 (Boolean propagation and unique-table TBPTs – structural reduce at each step). Boolean propagation is applied as a pre-processing step. Both the floating-point and Boolean potentials are collapsed to unique root table nodes after creation. After each VE operation the Boolean table potentials undergo a structural reduction: axes (dimensions) that are constant across all states of the corresponding variable are removed iteratively, analogously to pruning in a tree. This reduces the cost of subsequent Boolean *and* combinations and *or* marginalizations in the backward propagation pass.

All experiments were run on a single CPU core with a hard time limit of 20 minutes per inference case. Cases that exceeded this limit, are reported as "-" in Table 1. Other network results were not included when any of the 5 inference cases finished on time, or in case of encountering a network-file error (e.g. a malformed CPT).

5.2. Results

Table 1 reports execution times (in seconds) for each network and inference case. For Cases 5 and 6, which use Boolean propagation, the two reported values correspond to the Boolean preprocessing time and the subsequent VE time, respectively.

For small networks, the differences between the tabular configurations are relatively small. As the network size increases, however, Case 2 consistently outperforms the PGMPY baseline (Case 1). Both ultimately use the same tabular numerical representation, since Case 2 collapses each TBPT into a single table node. The difference is therefore mainly due to the VE implementation: profiling indicates that the PGMPY implementation incurs additional bookkeeping overhead when inserting intermediate factors into its dictionary-based factor structure, whereas our list-based implementation avoids most of this cost. This difference becomes increasingly relevant as intermediate factors grow.

Case 4, which uses pure probability trees throughout VE, represents an extreme configuration that fully exploits structural sparsity, but it could also incur structural overhead if it was not for the fact that pruning adjustment is done after each VE operation. For small networks this representation can be competitive, since the cost of manipulating the trees remains limited. As the intermediate potentials become larger, however, the cost of repeated tree combination and marginalization becomes more significant. This motivates the study of the hybrid representation in Case 3, to analyze the trade-off by retaining tree structure for sparse regions while using table nodes for denser regions. Nevertheless, Case 2 and then Case 4 mostly remains as strong faster baselines (with some exceptions).

Boolean propagation (Cases 5 and 6) substantially reduces VE time after the preprocessing step. However, adding up the time for the two phases, it is not so efficient as expected in majority of networks due to the zero percentage, which is high but not enough.

Network	Case 1	Case 2	Case 3	Case 4	Case 5	Case 6
or_chain_175.fg.uai	0.1283	0.1101	0.0361	0.0374	0.0894 / 0.0423	0.0741 / 0.0391
or_chain_180.fg.uai	0.1285	0.1094	0.0429	0.0482	0.0916 / 0.0518	0.0753 / 0.0485
or_chain_203.fg.uai	0.1294	0.1100	0.0366	0.0387	0.0902 / 0.0424	0.0734 / 0.0393
or_chain_154.fg.uai	0.1282	0.1098	0.0366	0.0394	0.0923 / 0.0433	0.0750 / 0.0396
or_chain_197.fg.uai	0.1260	0.1081	0.0367	0.0391	0.0925 / 0.0430	0.0755 / 0.0403
or_chain_85.fg.uai	0.1233	0.0298	11.1003	4.1946	0.1924 / 5.6165	0.0888 / 28.7160
or_chain_65.fg.uai	-	419.6897	-	-	-	-
or_chain_147.fg.uai	10.3802	0.2896	62.2847	57.1459	21.3749 / 59.4319	0.5302 / 97.1923
or_chain_182.fg.uai	1.2672	0.0753	34.9838	14.1524	17.3813 / 14.5579	0.3700 / 41.6451
or_chain_148.fg.uai	3.4480	0.1274	46.1200	30.2484	20.4653 / 31.4921	0.3947 / 46.7452
or_chain_133.fg.uai	-	163.4265	-	-	-	-
or_chain_168.fg.uai	0.0711	0.0252	2.0146	1.3578	1.9687 / 1.2502	0.1343 / 2.1183
or_chain_225.fg.uai	0.1647	0.0264	2.0142	1.3762	1.7725 / 1.5764	0.1453 / 2.1081
or_chain_201.fg.uai	0.1640	0.0249	1.6502	1.3587	2.4063 / 1.2076	0.1357 / 1.7837
or_chain_74.fg.uai	0.0638	0.0243	1.2413	1.0245	1.3742 / 1.2065	0.1285 / 1.3084
or_chain_10.fg.uai	452.6408	10.2309	-	1060.0257	-	-
or_chain_176.fg.uai	0.0478	0.0241	0.2609	0.1474	0.3727 / 0.2786	0.1165 / 0.2803
or_chain_17.fg.uai	7.4780	0.2118	199.1675	98.3047	43.2906 / 94.5516	1.0834 / 201.7990
or_chain_72.fg.uai	57.1647	1.2161	768.1628	391.7486	83.6787 / 400.9893	-
or_chain_68.fg.uai	256.2131	7.3527	-	-	-	-
or_chain_91.fg.uai	151.6876	2.6720	-	-	-	-
or_chain_45.fg.uai	0.0874	0.0403	1.2116	1.3422	1.1119 / 1.5882	0.1913 / 1.6491
or_chain_247.fg.uai	-	31.1615	-	-	-	-
or_chain_48.fg.uai	1088.4457	34.3893	-	-	-	-
or_chain_18.fg.uai	0.0852	0.0521	0.5219	0.5638	0.5522 / 0.6058	0.3746 / 0.4717
or_chain_129.fg.uai	249.7765	3.1772	-	-	-	-
or_chain_236.fg.uai	23.1650	0.5882	271.6528	233.5126	110.4698 / 234.4854	2.2863 / 285.8626
fs-07.uai	-	-	25.2566	0.9402	6.2943 / 0.9839	-

Table 1: Inference times (in seconds) for each network and query case. Each entry shows VE time or Boolean / VE time. "-" refers to exceeding the 20 minute limit.

The clearest improvement occurs in `fs-07.uai`, the largest and sparsest network (44.08% of global zeros, 7798 parameters), with a key factor due to large number of evidence nodes. It is shown here that TBPT potentials (case 3) can be leveraged from the fact that the large proportion of zeros in this network keeps zero-free table nodes small and manageable throughout VE. Even more beneficial it is the use of the Boolean propagation here (Case 5), anticipating and propagating the abundant zeros throughout the structure before VE begins, yielding an even more compressed set of potentials consuming less time. In this case, the pure probability-tree configuration with our original idea of applying adjustments of pruning (Case 4) obtains the most competitive result, completing the inference in only 0.94 s, while the Boolean-assisted Case 5 also substantially reduces the VE phase to 0.98 s, although its preprocessing overhead makes the total time higher.

The sensitivity of these results to the amount of sparsity is further examined in Appendix B, where the zero density of selected networks is systematically varied between 10%, 20%, 30%, the original value, and 40%.

6. Conclusion

We have introduced the TBPT hybrid potential representation, which unifies probability trees and multi-dimensional tables within a single recursive data structure. Key contributions include the formal definition of this structure introducing table nodes as new terminal nodes, the extension of operations to be applied in these potentials, the lifting strategy and a forward-backward Boolean propagation scheme before VE algorithm. Empirical evidence shows that this structure can provide a considerable trade-off between memory efficiency and computational time, specially in networks with high zero percentage.

Acknowledgments

This research is part of the project PID2022-139293NB-C33, funded by MICIU/AEI/ 10.13039/501100011033/ and the European Regional Development Fund (ERDF/EU).

References

Ankur Ankan and Johannes Textor. pgmpy: A python toolkit for bayesian networks. *Journal of Machine Learning Research*, 25(265):1–8, 2024. URL http://jmlr.org/papers/v25/23-0487.html.

R.I. Bahar, E.A. Frohm, C.M. Gaona, G.D. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebraic decision diagrams and their applications. In *Proceedings of 1993 International Conference on Computer Aided Design (ICCAD)*, pages 188–191, 1993.

Craig Boutilier, Nir Friedman, Moises Goldszmidt, and Daphne Koller. Context-specific independence in bayesian networks. In *Proceedings of the 12th Annual Conference on Uncertainty in Artificial Intelligence (UAI-96)*, pages 115–123, Portland, OR, USA, 1996.

Randal E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, 35(8):677–691, 1986.

Rafael Cabañas, Andrés Cano, Manuel Gómez-Olmedo, and Anders L. Madsen. Improvements to variable elimination and symbolic probabilistic inference for evaluating influence diagrams. *International Journal of Approximate Reasoning*, 70:13–35, 2016.

Andrés Cano and Serafín Moral. Using probability trees to compute marginals with imprecise probabilities. *International Journal of Approximate Reasoning*, 29(1):1–46, 2002.

Andrés Cano, Serafín Moral, and Antonio Salmerón. Penniless propagation in join trees. *International Journal of Intelligent Systems*, 15(11):1027–1059, 2000.

Andrés Cano, Manuel Gómez-Olmedo, and Serafín Moral. Approximate inference in bayesian networks using binary probability trees. *International Journal of Approximate Reasoning*, 52(1):49–62, 2011. Tenth European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU 2009).

Andrés Cano, Manuel Gómez-Olmedo, and Cora B. Pérez-Ariza. An extended approach to learning recursive probability trees from data. *International Journal of Intelligent Systems*, 30(3):355–383, 2015.

Meihua Dang, Guy Van den Broeck, Arthur Choi, and Adnan Darwiche. Conditional psdds: Modeling and learning with modular knowledge. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, 2018.

Adnan Darwiche. SDD: A New Canonical Representation of Propositional Knowledge Bases. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI)*, pages 819–826, 2011.

Manuel Gómez-Olmedo, Rafael Cabañas, Andrés Cano, Serafín Moral, and Ofelia P. Retamero. Value-based potentials: Exploiting quantitative information regularity patterns in probabilistic graphical models. *International Journal of Intelligent Systems*, 36(11): 6913–6943, 2021.

Doga Kisa, Guy Van den Broeck, Arthur Choi, and Adnan Darwiche. Probabilistic Sentential Decision Diagrams. In *Proceedings of the Fourteenth International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 558–567, 2014.

S. L. Lauritzen. *Graphical Models*, volume 17 of *Oxford Statistical Science Series*. The Clarendon Press Oxford University Press, New York, 1996. Oxford Science Publications.

S. L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society: Series B (Methodological)*, 50(2):157–194, 1988.

S. L. Lauritzen, A. P. Dawid, B. N. Larsen, and H.-G. Leimer. Independence properties of directed markov fields. *Networks*, 20(5):491–505, 1990.

Robert Mateescu, Rina Dechter, and Andrei Marinescu. AND/OR Multi-Valued Decision Diagrams (AOMDDs) for Graphical Models. *Journal of Artificial Intelligence Research*, 33:465–519, 2008.

S. Minato. Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems. In *30th ACM/IEEE Design Automation Conference*, pages 272–277, 1993.

Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988. ISBN 1558604790.

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

Antonio Salmerón, Andrés Cano, and Serafín Moral. Importance sampling in bayesian networks using probability trees. *Computational Statistics & Data Analysis*, 34(4):387–413, 2000.

Claude E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948.

Yujia Shen, Arthur Choi, and Adnan Darwiche. Tractable operations for arithmetic circuits of probabilistic models. In *Proceedings of the 30th International Conference on Neural Information Processing Systems (NIPS'16)*, pages 3943–3951, Red Hook, NY, USA, 2016. Curran Associates Inc.

Prakash P. Shenoy and Glenn Shafer. Axioms for probability and belief-function propagation. In *Machine intelligence and pattern recognition*, volume 9, pages 169–198. Elsevier, 1990.

Nevin Lianwen Zhang and David L. Poole. A simple approach to bayesian network computations. In *Proceedings of the Tenth Canadian Conference on Artificial Intelligence*, pages 171–178, 1994.

Appendix A. Description of the selected networks used in the experimental setup

Network	Nodes	nEvid	nNoEvid	Arcs	Card.	Params.	Global Zeros (%)	Mean Table Zeros (%)
or_chain_175.fg.uai	382	1	381	571	2	1906	33.21	20.75
or_chain_180.fg.uai	378	2	376	575	2	1906	30.38	19.21
or_chain_203.fg.uai	384	1	383	574	2	1916	33.25	20.77
or_chain_154.fg.uai	390	1	389	583	2	1946	33.20	20.74
or_chain_197.fg.uai	390	1	389	583	2	1946	33.20	20.74
or_chain_85.fg.uai	400	3	397	650	2	2102	33.40	22.03
or_chain_65.fg.uai	374	8	366	738	2	2228	36.98	27.81
or_chain_147.fg.uai	410	5	405	707	2	2234	35.72	24.48
or_chain_182.fg.uai	415	3	412	723	2	2276	36.16	24.88
or_chain_148.fg.uai	463	7	456	784	2	2494	35.00	23.73
or_chain_133.fg.uai	422	9	413	826	2	2500	36.36	27.19
or_chain_168.fg.uai	466	2	464	809	2	2550	37.10	25.43
or_chain_225.fg.uai	467	2	465	812	2	2558	37.18	25.51
or_chain_201.fg.uai	467	2	465	812	2	2558	37.18	25.51
or_chain_74.fg.uai	473	3	470	819	2	2584	37.07	25.40
or_chain_10.fg.uai	461	8	453	835	2	2594	35.85	25.38
or_chain_176.fg.uai	511	2	509	806	2	2634	32.92	21.26
or_chain_17.fg.uai	534	3	531	939	2	2946	36.66	25.35
or_chain_72.fg.uai	546	6	540	954	2	3000	35.00	24.15
or_chain_68.fg.uai	624	7	617	1104	2	3458	34.79	24.24
or_chain_91.fg.uai	674	9	665	1103	2	3556	32.06	21.29
or_chain_45.fg.uai	715	5	710	1121	2	3672	30.28	19.53
or_chain_247.fg.uai	668	5	663	1235	2	3806	35.39	25.30
or_chain_48.fg.uai	841	5	836	1441	2	4566	34.12	23.23
or_chain_18.fg.uai	894	4	890	1473	2	4736	32.58	21.63
or_chain_129.fg.uai	1005	7	998	1675	2	5360	31.23	20.91
or_chain_236.fg.uai	1039	4	1035	1696	2	5470	33.09	21.82
fs-07.uai	1225	1120	105	2483	2	7798	44.08	39.29

Table 2: Structural and Parametric Statistics of UAI Markov networks models from UAI 2022 competition MAR 2 Final Evaluation Benchmarks and the PROMEDAS `or_chain` family.

Appendix B. Additional experiments varying the zero percentage of networks

To further investigate the influence of sparsity on the different inference representations, an additional experiment was performed by systematically varying the global percentage of zero entries in a subset of the benchmark networks. The objective is to determine whether the relative behaviour observed in the main experiment is mainly associated with the network structure itself or is also strongly dependent on the amount of zeros in the potential tables.

For each selected network, additional UAI files were generated with target global zero percentages of 10%, 20%, and 30%, together with the original network (between 30% and 40%) and a target of 40%.

The zero percentage is defined globally over the complete UAI file as the number of zero entries in all tables divided by the total number of entries in all tables, multiplied by 100.

The additional UAI files were generated by modifying only tables with more than four entries. In the binary networks considered here, each such table can be viewed as a collection of columns corresponding to configurations of the remaining variables. The values along the first variable form a probability distribution. Zero entries were therefore introduced

or removed at the level of complete columns rather than individual entries, preserving the internal consistency of each distribution.

To increase the zero density, selected non-deterministic columns were determinized: the largest value in a column was set to 1 and the remaining values to 0. To decrease the zero density, selected deterministic columns were replaced by strictly positive probability vectors. The columns to be modified were selected globally across all modifiable tables, allowing the target number of zero entries to be approached while keeping the resulting tables valid.

The resulting files were then evaluated using the same inference procedure as in the main experiment. The results are reported in Tables 3, 4 and 5.

Overall, the experiment shows that changing the global zero percentage between 10% and 40% has relatively little effect on the runtime of Case 2 for most networks. This is consistent with Case 2 operating directly on a collapsed tabular representation, whose main computational behaviour is determined by the resulting factor scopes rather than by explicitly exploiting zero structure. For example, `or_chain_182` remains around 0.075 s across all zero densities in Case 2.

The same general stability can be observed for the tree-based Case 3, although its absolute runtime is usually higher than that of Case 2 on the larger networks. Case 4 also shows relatively stable behaviour in many of the tested instances, but its repeated pruning operations make its cost more dependent on the intermediate tree structure. Consequently, increasing the zero density does not automatically translate into a lower total runtime in this specific construction way: the additional sparsity can reduce the size of intermediate structures, but exploiting it also requires the corresponding tree operations.

The Boolean-based cases show a similar trade-off. Boolean preprocessing is generally inexpensive compared with the subsequent VE phase, and the Boolean representation can substantially reduce the latter. However, when the preprocessing time and VE time are added, the total runtime is not consistently better than the non-Boolean alternatives. The results therefore suggest that sparsity alone is not sufficient to guarantee an advantage from Boolean propagation; the structure of the zeros and the resulting reduction in the effective intermediate potentials are also important.

A particularly clear example is `or_chain_85`, where the runtime of the main non-Boolean cases changes very little across the generated instances. In contrast, the Boolean cases maintain a relatively large VE component despite the different zero densities. Similar stability can be observed for several of the smaller networks, where all representations remain within a narrow runtime range.

For the largest networks, the additional experiment also confirms the robust advantage of Case 2 over Case 1. For instance, `or_chain_182` remains close to 0.075 s in Case 2 across the different zero densities, whereas Case 1 remains around 1.2–1.4 s. Likewise, networks such as `or_chain_10`, `or_chain_68`, `or_chain_91`, and `or_chain_129` show that the tabular implementation of Case 2 can remain feasible even when the pgmpy baseline becomes substantially slower or reaches the time limit.

Taken together, these experiments support the conclusions of the main evaluation. The amount of zeros influences how beneficial tree pruning and Boolean propagation can be, but varying the global zero density alone does not substantially change the main advantage of the collapsed-table implementation. Case 2 remains the most consistently efficient approach,

Network	Case 1	Case 2	Case 3	Case 4	Case 5	Case 6
or_chain_175.fg_zero10.uai	0.1284	0.1102	0.0359	0.0378	0.0887 / 0.0423	0.0729 / 0.0387
or_chain_175.fg_zero20.uai	0.1253	0.1100	0.0364	0.0380	0.0899 / 0.0421	0.0727 / 0.0386
or_chain_175.fg_zero30.uai	0.1257	0.1095	0.0364	0.0374	0.0893 / 0.0422	0.0732 / 0.0386
or_chain_175.fg.uai	0.1283	0.1101	0.0361	0.0374	0.0894 / 0.0423	0.0741 / 0.0391
or_chain_175.fg_zero40.uai	0.1232	0.1103	0.0354	0.0372	0.0901 / 0.0427	0.0731 / 0.0385
or_chain_180.fg_zero10.uai	0.1291	0.1102	0.0431	0.0468	0.0912 / 0.0519	0.0746 / 0.0486
or_chain_180.fg_zero20.uai	0.1284	0.1113	0.0431	0.0470	0.0917 / 0.0522	0.0743 / 0.0487
or_chain_180.fg_zero30.uai	0.1253	0.1124	0.0429	0.0473	0.0916 / 0.0524	0.0747 / 0.0486
or_chain_180.fg.uai	0.1285	0.1094	0.0429	0.0482	0.0916 / 0.0518	0.0753 / 0.0485
or_chain_180.fg_zero40.uai	0.1285	0.1102	0.0427	0.0477	0.0923 / 0.0518	0.0748 / 0.0484
or_chain_203.fg_zero10.uai	0.1283	0.1130	0.0366	0.0390	0.0906 / 0.0428	0.0749 / 0.0401
or_chain_203.fg_zero20.uai	0.1302	0.0161	0.0364	0.0393	0.0904 / 0.0428	0.0746 / 0.0393
or_chain_203.fg_zero30.uai	0.1273	0.1103	0.0368	0.0392	0.0905 / 0.0435	0.0745 / 0.0399
or_chain_203.fg.uai	0.1294	0.1100	0.0366	0.0387	0.0902 / 0.0424	0.0734 / 0.0393
or_chain_203.fg_zero40.uai	0.1334	0.1098	0.0361	0.0387	0.0907 / 0.0427	0.0747 / 0.0397
or_chain_154.fg_zero10.uai	0.1282	0.0165	0.0369	0.0387	0.0916 / 0.0433	0.0751 / 0.0399
or_chain_154.fg_zero20.uai	0.1284	0.0165	0.0374	0.0391	0.0914 / 0.0428	0.0755 / 0.0408
or_chain_154.fg_zero30.uai	0.1289	0.1101	0.0365	0.0392	0.0930 / 0.0436	0.0749 / 0.0400
or_chain_154.fg.uai	0.1282	0.1098	0.0366	0.0394	0.0923 / 0.0433	0.0750 / 0.0396
or_chain_154.fg_zero40.uai	0.1238	0.1111	0.0368	0.0392	0.0918 / 0.0435	0.0751 / 0.0398
or_chain_197.fg_zero10.uai	0.1283	0.1111	0.0367	0.0396	0.0920 / 0.0431	0.0753 / 0.0401
or_chain_197.fg_zero20.uai	0.1331	0.1105	0.0364	0.0397	0.0930 / 0.0438	0.0761 / 0.0404
or_chain_197.fg_zero30.uai	0.1256	0.1109	0.0368	0.0391	0.0919 / 0.0434	0.0750 / 0.0398
or_chain_197.fg.uai	0.1260	0.1081	0.0367	0.0391	0.0925 / 0.0430	0.0755 / 0.0403
or_chain_197.fg_zero40.uai	0.1284	0.1098	0.0368	0.0392	0.0927 / 0.0432	0.0755 / 0.0401
or_chain_85.fg_zero10.uai	0.2176	0.0298	11.1067	4.2019	0.1978 / 5.6404	0.0888 / 28.3573
or_chain_85.fg_zero20.uai	0.2115	0.0292	11.0549	4.1499	0.1982 / 5.6949	0.0883 / 28.6644
or_chain_85.fg_zero30.uai	0.2191	0.0297	11.1280	4.2673	0.3108 / 5.5722	0.0892 / 28.3696
or_chain_85.fg.uai	0.1233	0.0298	11.1003	4.1946	0.1924 / 5.6165	0.0888 / 28.7160
or_chain_85.fg_zero40.uai	0.1240	0.0299	11.1331	4.2132	0.1920 / 5.6256	0.0876 / 28.5702
or_chain_65.fg_zero10.uai	-	423.9770	-	-	-	-
or_chain_65.fg_zero20.uai	-	423.0376	-	-	-	-
or_chain_65.fg_zero30.uai	-	421.6598	-	-	-	-
or_chain_65.fg.uai	-	419.6897	-	-	-	-
or_chain_65.fg_zero40.uai	-	419.0516	-	-	-	-
or_chain_147.fg_zero10.uai	9.3502	0.2898	63.0660	56.3548	21.6130 / 59.6148	0.9770 / 96.9044
or_chain_147.fg_zero20.uai	9.7832	0.2895	61.5691	56.6253	21.6345 / 58.4512	0.6228 / 96.6644
or_chain_147.fg_zero30.uai	11.7422	0.2896	62.2335	57.2763	21.9428 / 58.1487	0.6761 / 95.8533
or_chain_147.fg.uai	10.3802	0.2896	62.2847	57.1459	21.3749 / 59.4319	0.5302 / 97.1923
or_chain_147.fg_zero40.uai	14.8022	0.2897	62.2961	56.3877	21.1065 / 58.3862	1.1300 / 96.6053
or_chain_182.fg_zero10.uai	1.1860	0.0747	34.9486	13.9788	18.2424 / 14.1526	0.3754 / 41.9889
or_chain_182.fg_zero20.uai	1.2940	0.0745	35.0153	14.0709	17.4620 / 14.6746	0.3253 / 41.8095
or_chain_182.fg_zero30.uai	1.3792	0.0750	35.1135	14.1697	17.6069 / 14.2145	0.3081 / 41.2795
or_chain_182.fg.uai	1.2672	0.0753	34.9838	14.1524	17.3813 / 14.5579	0.3700 / 41.6451
or_chain_182.fg_zero40.uai	1.2948	0.0746	34.9718	14.0473	19.8451 / 14.7516	0.3593 / 41.2638
or_chain_148.fg_zero10.uai	3.0314	0.1264	46.1163	30.0391	19.3412 / 30.5333	0.3396 / 46.8374
or_chain_148.fg_zero20.uai	2.4724	0.1263	46.0742	30.1950	18.8974 / 30.5890	0.3552 / 46.6234
or_chain_148.fg_zero30.uai	2.7377	0.1257	46.1444	30.4638	18.0288 / 30.7854	0.4204 / 46.8552
or_chain_148.fg.uai	3.4480	0.1274	46.1200	30.2484	20.4653 / 31.4921	0.3947 / 46.7452
or_chain_148.fg_zero40.uai	2.7580	0.1260	46.5264	30.2011	21.7753 / 31.0041	0.3415 / 46.8304
or_chain_133.fg_zero10.uai	-	163.1120	-	-	-	-
or_chain_133.fg_zero20.uai	-	163.0819	-	-	-	-
or_chain_133.fg_zero30.uai	-	163.4627	-	-	-	-
or_chain_133.fg.uai	-	163.4265	-	-	-	-
or_chain_133.fg_zero40.uai	-	164.0139	-	-	-	-
or_chain_168.fg_zero10.uai	0.1637	0.0254	1.9935	1.3968	2.0070 / 1.4475	0.1344 / 2.1177
or_chain_168.fg_zero20.uai	0.1638	0.0250	2.0035	1.3580	1.7945 / 1.5442	0.1328 / 2.1039
or_chain_168.fg_zero30.uai	0.0702	0.0249	2.0117	1.3587	1.8027 / 1.5340	0.1355 / 2.0849
or_chain_168.fg.uai	0.0711	0.0252	2.0146	1.3578	1.9687 / 1.2502	0.1343 / 2.1183
or_chain_168.fg_zero40.uai	0.1651	0.0253	2.0022	1.3623	1.9609 / 1.4432	0.1357 / 2.1150

Table 3: Inference times (in seconds) for each network and query case for the additional zero percentage experiment (1). Each entry shows VE time or Boolean / VE time. "-" refers to exceeding the 20 minute limit.

Network	Case 1	Case 2	Case 3	Case 4	Case 5	Case 6
or_chain_225.fg_zero10.uai	0.1662	0.0263	1.9985	1.3606	1.7835 / 1.5387	0.1445 / 2.1265
or_chain_225.fg_zero20.uai	0.1662	0.0257	2.0035	1.3515	1.7664 / 1.5463	0.1459 / 2.1227
or_chain_225.fg_zero30.uai	0.1679	0.0258	2.0084	1.3529	1.7981 / 1.5510	0.1461 / 2.1209
or_chain_225.fg.uai	0.1647	0.0264	2.0142	1.3762	1.7725 / 1.5764	0.1453 / 2.1081
or_chain_225.fg_zero40.uai	0.1742	0.0261	1.9866	1.3641	1.9517 / 1.2511	0.1458 / 2.1019
or_chain_201.fg_zero10.uai	0.1668	0.0250	1.6345	1.2942	2.2024 / 1.3191	0.1351 / 1.7737
or_chain_201.fg_zero20.uai	0.1661	0.0250	1.6583	1.1028	2.2242 / 1.3042	0.1364 / 1.7665
or_chain_201.fg_zero30.uai	0.1659	0.0246	1.6438	1.2911	2.2169 / 1.2843	0.1364 / 1.7730
or_chain_201.fg.uai	0.1640	0.0249	1.6502	1.3587	2.4063 / 1.2076	0.1357 / 1.7837
or_chain_201.fg_zero40.uai	0.1687	0.0249	1.6562	1.3005	2.2008 / 1.3253	0.1376 / 1.7821
or_chain_74.fg_zero10.uai	0.0673	0.0241	1.2413	1.0183	1.3760 / 1.0377	0.1281 / 1.3100
or_chain_74.fg_zero20.uai	0.0630	0.0243	1.2437	1.0284	1.4002 / 1.0292	0.1289 / 1.2980
or_chain_74.fg_zero30.uai	0.1579	0.0239	1.2501	1.0184	1.4113 / 1.0284	0.1304 / 1.3091
or_chain_74.fg.uai	0.0638	0.0243	1.2413	1.0245	1.3742 / 1.2065	0.1285 / 1.3084
or_chain_74.fg_zero40.uai	0.1585	0.0243	1.2396	1.0285	1.3782 / 1.0371	0.1297 / 1.3030
or_chain_10.fg_zero10.uai	348.4047	10.2769	-	1061.9374	-	-
or_chain_10.fg_zero20.uai	364.6859	10.2883	-	1062.6560	-	-
or_chain_10.fg_zero30.uai	409.1942	10.2280	-	1064.9046	-	-
or_chain_10.fg.uai	452.6408	10.2309	-	1060.0257	-	-
or_chain_10.fg_zero40.uai	310.3723	10.1528	-	1066.0005	-	-
or_chain_176.fg_zero10.uai	0.0476	0.0244	0.2603	0.1469	0.3774 / 0.2789	0.1174 / 0.2799
or_chain_176.fg_zero20.uai	0.0481	0.0246	0.2583	0.1470	0.3836 / 0.2787	0.1178 / 0.2821
or_chain_176.fg_zero30.uai	0.0474	0.0245	0.2592	0.1481	0.3745 / 0.2751	0.1169 / 0.2815
or_chain_176.fg.uai	0.0478	0.0241	0.2609	0.1474	0.3727 / 0.2786	0.1165 / 0.2803
or_chain_176.fg_zero40.uai	0.0481	0.0244	0.2575	0.1467	0.3780 / 0.2797	0.1175 / 0.2793
or_chain_17.fg_zero10.uai	7.4932	0.2124	196.6511	98.5166	45.7298 / 98.3962	1.1826 / 203.1902
or_chain_17.fg_zero20.uai	7.0335	0.2108	199.4396	98.0134	45.4911 / 93.5559	1.3597 / 204.2058
or_chain_17.fg_zero30.uai	7.2319	0.2116	199.0189	97.9865	43.8106 / 95.7381	1.4254 / 202.1645
or_chain_17.fg.uai	7.4780	0.2118	199.1675	98.3047	43.2906 / 94.5516	1.0834 / 201.7990
or_chain_17.fg_zero40.uai	9.9738	0.2131	197.7917	98.3515	48.2403 / 95.5104	1.2342 / 201.8132
or_chain_72.fg_zero10.uai	62.4419	1.2188	766.3108	390.9476	88.7654 / 403.8699	-
or_chain_72.fg_zero20.uai	57.6053	1.2171	762.5523	392.5074	81.9160 / 405.6272	-
or_chain_72.fg_zero30.uai	71.7539	1.2225	763.3130	394.5219	101.0492 / 409.9260	-
or_chain_72.fg.uai	57.1647	1.2161	768.1628	391.7486	83.6787 / 400.9893	-
or_chain_72.fg_zero40.uai	73.9544	1.2203	761.8952	391.9895	87.4656 / 402.2593	-
or_chain_68.fg_zero10.uai	240.6438	7.5511	-	-	-	-
or_chain_68.fg_zero20.uai	287.5478	7.3817	-	-	-	-
or_chain_68.fg_zero30.uai	315.1658	7.3476	-	-	-	-
or_chain_68.fg.uai	256.2131	7.3527	-	-	-	-
or_chain_68.fg_zero40.uai	259.3228	7.3269	-	-	-	-
or_chain_91.fg_zero10.uai	115.1240	2.6763	-	-	-	-
or_chain_91.fg_zero20.uai	174.7945	2.6842	-	-	-	-
or_chain_91.fg_zero30.uai	120.9659	2.6842	-	-	-	-
or_chain_91.fg.uai	151.6876	2.6720	-	-	-	-
or_chain_91.fg_zero40.uai	123.5251	2.6791	-	-	-	-
or_chain_45.fg_zero10.uai	0.0874	0.0403	1.1988	1.3529	1.1081 / 1.5915	0.2030 / 1.6451
or_chain_45.fg_zero20.uai	0.0873	0.0406	1.2066	1.3533	1.1281 / 1.5916	0.1928 / 1.6610
or_chain_45.fg_zero30.uai	0.0867	0.0407	1.2222	1.3548	1.0903 / 1.6063	0.1918 / 1.6621
or_chain_45.fg.uai	0.0874	0.0403	1.2116	1.3422	1.1119 / 1.5882	0.1913 / 1.6491
or_chain_45.fg_zero40.uai	0.0867	0.0406	1.2081	1.3551	1.1009 / 1.6082	0.1926 / 1.6582
or_chain_247.fg_zero10.uai	-	30.9075	-	-	-	-
or_chain_247.fg_zero20.uai	-	30.7451	-	-	-	-
or_chain_247.fg_zero30.uai	-	31.0268	-	-	-	-
or_chain_247.fg.uai	-	31.1615	-	-	-	-
or_chain_247.fg_zero40.uai	-	31.2254	-	-	-	-
or_chain_48.fg_zero10.uai	-	34.2127	-	-	-	-
or_chain_48.fg_zero20.uai	924.6847	34.3797	-	-	-	-
or_chain_48.fg_zero30.uai	872.4838	34.4897	-	-	-	-
or_chain_48.fg.uai	1088.4457	34.3893	-	-	-	-
or_chain_48.fg_zero40.uai	-	34.2337	-	-	-	-
or_chain_18.fg_zero10.uai	0.0846	0.0523	0.5195	0.5649	0.5564 / 0.6060	0.3799 / 0.4866

Table 4: Inference times (in seconds) for each network and query case for the additional zero percentage experiment (2). Each entry shows VE time or Boolean / VE time. "-" refers to exceeding the 20 minute limit.

Network	Case 1	Case 2	Case 3	Case 4	Case 5	Case 6
or_chain_18.fg_zero20.uai	0.0840	0.0519	0.5192	0.5693	0.5589 / 0.6040	0.3720 / 0.4742
or_chain_18.fg_zero30.uai	0.0846	0.0527	0.5203	0.5643	0.5584 / 0.6069	0.3745 / 0.4731
or_chain_18.fg.uai	0.0852	0.0521	0.5219	0.5638	0.5522 / 0.6058	0.3746 / 0.4717
or_chain_18.fg_zero40.uai	0.0847	0.0513	0.5257	0.5678	0.5598 / 0.6089	0.3743 / 0.4767
or_chain_129.fg_zero10.uai	274.9790	3.2386	-	-	-	-
or_chain_129.fg_zero20.uai	208.4339	3.1960	-	-	-	-
or_chain_129.fg_zero30.uai	173.4075	3.1769	-	-	-	-
or_chain_129.fg.uai	249.7765	3.1772	-	-	-	-
or_chain_129.fg_zero40.uai	286.7737	3.2231	-	-	-	-
or_chain_236.fg_zero10.uai	22.8485	0.5877	271.7141	230.3471	106.6004 / 234.0790	1.2338 / 285.8169
or_chain_236.fg_zero20.uai	20.4712	0.5879	268.1153	232.5565	109.0457 / 234.8922	1.5050 / 286.4819
or_chain_236.fg_zero30.uai	22.7516	0.5878	272.6514	230.5008	117.7269 / 224.0175	1.4140 / 287.9783
or_chain_236.fg.uai	23.1650	0.5882	271.6528	233.5126	110.4698 / 234.4854	2.2863 / 285.8626
or_chain_236.fg_zero40.uai	20.7496	0.5880	269.1649	231.3581	106.0381 / 226.0769	1.3164 / 290.7275

Table 5: Inference times (in seconds) for each network and query case for the additional zero percentage experiment (3). Each entry shows VE time or Boolean / VE time. "-" refers to exceeding the 20 minute limit.

while the tree-based and Boolean representations become particularly interesting when the zero structure is sufficiently favourable to offset the additional structural or preprocessing overhead.