# Ordering Variables for Weighted Model Integration

**Vincent Derkinderen**[1,2], **Evert Heylen**[1], **Pedro Zuidberg Dos Martires**[1,2], **Samuel Kolb**[1,2], **Luc De Raedt**[1,2,3]
[1]Dept. of Computer Science, KU Leuven, B-3000 Leuven, Belgium
[2]Leuven.AI - KU Leuven Institute for AI, B-3000 Leuven, Belgium
[3]Center for Applied Autonomous Systems, Örebro University, Sweden

## Abstract

State-of-the-art probabilistic inference algorithms, such as variable elimination and search-based approaches, rely heavily on the order in which variables are marginalized. Finding the optimal ordering is an NP-complete problem. This computational hardness has led to heuristics to find adequate variable orderings. However, these heuristics have mostly been targeting discrete random variables. We show how variable ordering heuristics from the discrete domain can be ported to the discrete-continuous domain. We equip the state-of-the-art F-XSDD(BR) solver for discrete-continuous problems with such heuristics. Additionally, we propose a novel heuristic called bottom-up min-fill (BU-MiF), yielding a solver capable of determining good variable orderings without having to rely on the user to provide such an ordering. We empirically demonstrate its performance on a set of benchmark problems.

## 1 INTRODUCTION

Probabilistic graphical models are used to model complex probability distributions over discrete and continuous random variables. The expressiveness and elegance of these models have enabled their deployment in a variety of applications. One only needs to model the problem and perform probabilistic inference using specialized algorithms such as the *sum-product algorithm* [Pearl, 1982], *variable elimination* (VE) [Zhang and Poole, 1994], or *bucket elimination* [Dechter, 1999]. These algorithms heavily rely on finding variable orderings that minimise the cost of computation, i.e., on finding a variable ordering in which to marginalize out single random variables. Finding the optimal variable ordering is NP-complete [Arnborg, 1985], which has led to the development of several heuristics for finding such orderings, e.g. [Kjærulff, 1990, Darwiche, 2009, Dechter, 2013].

Probabilistic inference in discrete graphical models is reducible to *weighted model counting* (WMC) [Darwiche, 2009]. WMC is the problem of calculating the probability of a Boolean formula being satisfied given that the literals are only true with a certain probability, which is in general a #P-hard problem [Chavira and Darwiche, 2008].

*Weighted model integration* (WMI) [Belle et al., 2015] is a recent extension of WMC from discrete to discrete-continuous domains. This introduces an additional major complication, namely integrating out continuous variables: in the case of integrating polynomials over convex polytopes, for instance, this problem is also #P-hard [Valiant, 1979]. WMI consists thus of two computationally hard problems 1) a combinatorial problem (also present in WMC) and 2) integrating out continuous variables. In practice, for many WMI problems the integration of continuous variables is the main bottleneck. Several works have studied inference for WMI through repeated integration (e.g. [Kolb et al., 2018, Kolb et al., 2019b, Zeng and Van den Broeck, 2019]). Such approaches, like the discrete-only variable elimination algorithms, are very sensitive to the variable ordering, i.e., the ordering in which integrations are performed. However, in the WMI literature it is typically assumed that a (good) ordering is provided by the user, which is unrealistic. Therefore, we study the problem of automatically determining a good variable ordering in the context of WMI.

Specifically, we show, as a **first contribution**, how established variable ordering techniques in the discrete setting can be extended to the discrete-continuous (hybrid) setting (Section 3). Extending these techniques to

*Proceedings of the 36th Conference on Uncertainty in Artificial Intelligence (UAI)*, PMLR volume 124, 2020.

the hybrid setting is not straightforward as problems with continuous variables exhibit additional dependencies that impact the difficulty of the integration steps. Additionally, we also map the concept of *variable trees* (vtree) [Pipatsrisawat and Darwiche, 2008], which generalizes the concept of variable orderings, to the continuous setting (Section 4). This allows us to develop (tree-based) ordering techniques for the state-of-the-art WMI solver F-XSDD(BR) [Kolb et al., 2019b]. Our **second contribution** is BU-MiF, a novel heuristic that produces a variable tree for the discrete-continuous domain, and which has no direct analog in the discrete domain. **Thirdly**, we extend F-XSDD(BR) with our new heuristic and experimentally show its benefits. We demonstrate our approach on a set of benchmark problems using PyWMI [Kolb et al., 2019a] and show that it provides better performance than currently available orderings. Furthermore, the BU-MiF heuristic allows the F-XSDD(BR) algorithm to perform well on a set of benchmark problems from a tractable subset of WMI, outperforming the specialized SMI solver [Zeng and Van den Broeck, 2019] introduced to tackle this class of problems.

# 2 PRELIMINARIES

## 2.1 Weighted Model Integration

Encoding a probabilistic inference problem as a weighted model counting problem has emerged as the canonical technique when dealing with discrete random variables. Probabilistic inference is reduced to determining the probability of a propositional logic formula being satisfied given that the literals present in the formula are themselves satisfied only probabilistically. Weighted model integration extends WMC from the setting of propositional logic formulas to so-called *satisfiability modulo theory* (SMT) formulas. Following [Morettin et al., 2017], we define SMT formulas over *linear real arithmetics*.

**Definition 1 (SMT($\mathcal{LRA}$)).** Let $\mathbf{b}$ be a set of $M$ Boolean and $\mathbf{x}$ a set of $N$ real variables. An atomic formula is an expression of the form $\sum_i c_i \cdot x_i \bowtie c$, where the $x_i \in \mathbf{x}$ and $c_i, c \in \mathbb{Q}$, and $\bowtie \in \{=, \neq, \geq, \leq, >, <\}$. We then define SMT($\mathcal{LRA}$) theories as Boolean combinations (by means of the standard Boolean operators $\{\neg, \wedge, \vee\}$) of Boolean variables $b_i \in \mathbf{b}$ and of atomic formulas in $\mathbf{x}$.

We define weighted model integration in terms of an indicator function over an SMT($\mathcal{LRA}$) formula, cf. [Kolb et al., 2019b].

**Definition 2 (WMI).** Given a set $\mathbf{b}$ of $M$ Boolean variables, $\mathbf{x}$ of $N$ real variables, a weight function $w{:}\mathbb{B}^M \times \mathbb{R}^N \to \mathbb{R}_{\geq 0}$, and a support $\phi$, in the form of an SMT formula, over $\mathbf{b}$ and $\mathbf{x}$, the weighted model integral

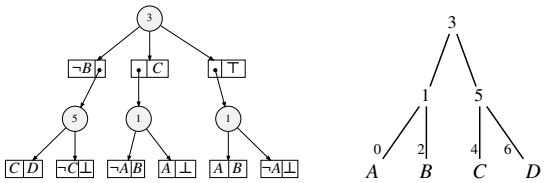



Figure 1: An SDD representing $(A \wedge B) \vee (C \wedge D) \vee (B \wedge C)$ (left) and the vtree used for its construction (right).

is given by:

$$\mathrm{WMI}(\phi, w|\mathbf{x}, \mathbf{b}) = \sum_{\mathbf{b}} \int [\![\phi(\mathbf{x}, \mathbf{b})]\!] w(\mathbf{x}, \mathbf{b}) d\mathbf{x} \quad (1)$$

where we use the Iverson bracket notation in $[\![\phi(\mathbf{x}, \mathbf{b})]\!]$ to denote the indicator function of $\phi(\mathbf{x}, \mathbf{b})$. If the set of real variables $\mathbf{x}$ is empty the WMI task reduces to WMC.

## 2.2 Sentential Decision Diagrams

Knowledge compilation [Darwiche and Marquis, 2002] is the process of compiling a propositional logic formula into a particular form that allows to execute certain operations, such as WMC, efficiently. A popular target language to compile propositional formulas into are *Sentential Decision Diagrams* (SDDs) [Darwiche, 2011]. The knowledge compilation step constitutes the #P-complete part of probabilistic inference via WMC.

An SDD $\alpha$ is a structure that represents a propositional logic theory $\langle \alpha \rangle$. Each SDD node $\alpha$ represents either a terminal ($\top$ or $\bot$), a literal (e.g. $X$ or $\neg X$) or a decomposition ($\{(p_1, s_1), \ldots, (p_n, s_n)\}$). The latter can be considered an OR-node, $\langle \alpha \rangle = \bigvee_{i=1}^{n} \langle p_i \rangle \wedge \langle s_i \rangle$ with $p_i$ and $s_i$ both also SDDs. Each decomposition node is graphically represented as a circular node connecting to each of its pairs. A pair $(p_i, s_i)$ is a conjunction of a prime $p_i$ and a sub $s_i$, represented as a paired box with $p_i$ on the left and $s_i$ on the right (Figure 1). From a top-down perspective, the theory $\langle \alpha \rangle$ of a decomposition node, is partitioned into several branches such that $\langle s_i \rangle$ is $\langle \alpha \rangle$ conditioned on $\langle p_i \rangle$.

To determine what variables to condition on (the variables in the prime), a vtree is used to recursively guide the SDD construction process. Each decomposition node respects a node in the vtree $v$ such that the variables in the prime are all variables in $v^l$ (left branch) and all variables in the subs are in $v^r$ (right branch).

**Definition 3 (Vtree).** A *vtree* for variables $\mathbf{X}$ is a full binary tree whose leaves are in one-to-one correspondence with the variables in $\mathbf{X}$.

**Example 1.** Figure 1 (left) illustrates an SDD of the theory $\langle \alpha \rangle = (A \wedge B) \vee (C \wedge D) \vee (B \wedge C)$ as a decomposition of three elements. Notice that each sub indeed represents

the theory of the parent node conditioned on the prime (e.g. $\langle s_1 \rangle = C \wedge D = \langle \alpha \rangle | \neg B$). The vtree guiding the SDD construction is given in Figure 1 (right).

Vtrees generalise the concept of variable ordering, which is, for instance, present in *ordered binary decision diagrams* (OBDD) [Bryant, 1986]. A variable ordering $d$ can be converted into a corresponding vtree by creating a right-linear vtree where each left subvtree is a leaf node with variables respecting $d$. As every variable ordering is a vtree but not every vtree is a variable ordering and as vtrees underlie SDDs and variable orderings underlie OBDDs, SDDs constitute a richer compilation language than OBDDs. As a consequence, SDDs are strictly more succinct than OBDDs (every OBDD is an SDD but not vice-versa). The formal definition of an SDD with respect to a given vtree can be found in [Darwiche, 2011].

### 2.3 WMI with SDDs

When an SDD is used to represent a propositional theory $T$, the weighted model count of $T$ can be computed in time linear to the size of the SDD. This is used by state-of-the-art approaches to obtain efficient probabilistic inference for the discrete domain.

Similar to knowledge compilation approaches for WMC, WMI has also received attention from this direction in order to build solvers. Such solvers have either been based on *extended algebraic decision diagrams*[1] [Kolb et al., 2018] or *extended SDDs* (XSDDs) [Zuidberg Dos Martires et al., 2019]. These *extended* representation allow, in addition to Boolean literals, also the use of SMT atoms.

A state-of-the-art solver based on XSDDs is F-XSDD(BR) [Kolb et al., 2019b]. Currently, a major problem of using XSDDs for solving WMI problems is that the vtree underlying the XSDD construction does not take into account which continuous variables are present in which SMT atomic literals, i.e. the compilation step of an XSDD is agnostic towards the extra information present in atomic SMT literals that might be helpful when integrating out the continuous random variables and only *sees* Boolean abstractions of these SMT literals. We now show how we repair this problem.

## 3 VARIABLE ORDERINGS

### 3.1 How to Exploit Structure

Probabilistic inference is a computationally hard problem and exploiting the structure that is present in any given problem is crucial in order to manage this hardness. We start this subsection by showing what it means to exploit structure in the discrete setting (WMC) and continue with the discrete-continuous setting (WMI).

**Weighted Model Counting** We explain the discrete setting in the context of conditional probabilities, also called factors. Consider the problem of computing the probability $P(C)$ using the factors $P(A)$, $P(B|A)$ and $P(C|B)$. This can be done as follows:

$$P(C) = \sum_A \sum_B P(A, B, C) \tag{2}$$
$$= \sum_A \sum_B P(C|B)P(B|A)P(A) \tag{3}$$

In variable elimination approaches, evaluating this translates to first computing $\sum_B P(C|B)P(B|A)P(A)$, resulting in a new factor $f(A, C)$. The size of an intermediate factor is exponential in the number of its variables causing both the time and space complexity to be exponential. Fortunately, distributivity, commutativity and associativity can be used to reduce the number of operations that have to be performed. We can for example push inside the summation over $A$

$$P(C) = \sum_B P(C|B) \sum_A P(A)P(B|A) \tag{4}$$

This leads to an intermediate factor $f(B) = \sum_A P(A)P(B|A)$ depending on only one variable. The complexity is now no longer necessarily exponential in the number of variables but is instead determined by the problem structure and the variable ordering $d$ — here $d=(B, A)$. The importance of the latter becomes apparent if we consider $d=(A, B)$ instead:

$$P(C) = \sum_A P(A) \sum_B P(B|A)P(C|B) \tag{5}$$

We obtain the factor $f(A, C) = \sum_B P(B|A)P(C|B)$, depending on two variables.

Unfortunately, finding a variable ordering that leads to intermediate factors with the lowest maximum number of variables is in general NP-complete [Dechter, 2013].

**Weighted Model Integration** Pushing the sum operation inside, as done in the discrete case, has recently also been studied for the continuous case [Kolb et al., 2019b], the key difference being that integrations are pushed inside instead of summations.

Reconsider the definition of a weighted model integral (cf. Equation 1). Let us assume, for the sake of simplicity, that the weight function $w$ does not depend on Boolean variables and fully factorizes, i.e. it is separable into factors depending only on single continuous variables:

$$\mathrm{WMI}(\phi, w|\mathbf{x}, \mathbf{b}) = \sum_{\mathbf{b}} \int [\![\phi(\mathbf{x}, \mathbf{b})]\!] \big[ \overbrace{\textstyle\prod_{x_i} w_i(x_i)}^{=w_i(\mathbf{x})} \big] d\mathbf{x}$$

[1] Algebraic decision diagrams are an extension of OBDDs.

Such separable weight functions allow us to push inside integrations over specific variables in an integrand. For instance, consider the function $p(z)$:

$$p(z) = \int ([\![0<z<1]\!][\![y\leq z]\!][\![x\leq y]\!]xyz)\, dxdy \quad (6)$$

Due to the separable weight function $xyz$ we can push the integrations over $x$ and $y$ inside the integrand, similar to pushing inside summations in the discrete case.

$$p(z) = [\![0<z<1]\!] \left(\int [\![y\leq z]\!] \left(\int [\![x\leq y]\!] x dx\right) y dy\right) z \quad (7)$$

Similarly again to the discrete setting, choosing different orders in which to push inside the integrations can have tremendous effects on the space and time requirements of running an inference algorithm.

**Example 2.** Given the weight function $w{=}1$ and support

$$\phi{=}(\textstyle\bigwedge_{i=\{1,\ldots,4\}}(x_0{\leq}x_i)){\wedge}\bigwedge_{i=\{0,\ldots,4\}}(0{\leq}x_i{\leq}1) \quad (8)$$

where $x_1, x_2, x_3$ and $x_4$ all interact with $x_0$. Using $bnds_0$ to denote $\prod_{i=\{1,2,3,4\}}[\![0 \leq x_i \leq 1]\!]$ and $bnds_1$ for $\prod_{i=\{0,2,3,4\}}[\![0 \leq x_i \leq 1]\!]$. If we first integrate out $x_0$ we obtain:

$$\begin{aligned}\int [\![\phi]\!] w dx_0 {=} & x_1 [\![x_1{<}x_2]\!][\![x_1{<}x_3]\!][\![x_1{<}x_4]\!] bnds_0 + \\ & x_2 [\![x_1{\geq}x_2]\!][\![x_2{<}x_3]\!][\![x_2{<}x_4]\!] bnds_0 + \\ & x_3 [\![x_1{\geq}x_2]\!][\![x_2{\geq}x_3]\!][\![x_3{<}x_4]\!] bnds_0 + \\ & x_3 [\![x_1{<}x_2]\!][\![x_1{\geq}x_3]\!][\![x_3{<}x_4]\!] bnds_0 + \\ & x_4 [\![x_1{\geq}x_2]\!][\![x_2{\geq}x_3]\!][\![x_3{\geq}x_4]\!] bnds_0 + \\ & x_4 [\![x_1{\geq}x_2]\!][\![x_2{<}x_3]\!][\![x_2{\geq}x_4]\!] bnds_0 + \\ & x_4 [\![x_1{<}x_2]\!][\![x_1{<}x_3]\!][\![x_1{\geq}x_4]\!] bnds_0 + \\ & x_4 [\![x_1{<}x_2]\!][\![x_1{\geq}x_3]\!][\![x_3{\geq}x_4]\!] bnds_0 \end{aligned} \quad (9)$$

However, first integrating out $x_1$ instead, yields the more compact intermediate result, resulting in more efficient subsequent computations (the symbolic expression tree representing the integrand is much smaller).

$$\int [\![\phi]\!] w dx_1 {=} (1{-}x_0)(\textstyle\prod_{i=\{2,3,4\}}[\![x_0{<}x_i]\!]) bnds_1 \quad (10)$$

Even though [Kolb et al., 2019b] studied pushing inside integrations, they did not develop any heuristics to do so. Their approach relied on hand-crafting specific integration orders for specific problems. In the following two sections we delineate how variable ordering strategies in the discrete setting can be adapted for the hybrid domain.

### 3.2 How to Order Variables

Since finding the best variable ordering (i.e. smallest intermediate size) is in general NP-complete, we instead use heuristics. We first introduce additional concepts used to analyse and find good variable orderings.

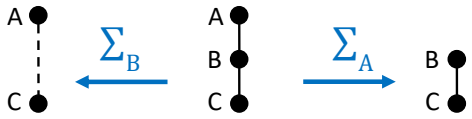



Figure 2: The interaction graph for factors $f(A)$, $f(A,B)$ and $f(B,C)$ (middle), the graph when $B$ is summed out (left) and the graph when $A$ is summed out (right).

Then, we explain three simple and common variable ordering heuristics that use these concepts. While this explanation is based on existing work for discrete problems [Darwiche, 2009, Dechter, 2013], we show how to apply it to the hybrid problem setting, a problem that, to the best of our knowledge, has not yet received much attention.

#### 3.2.1 Interaction Graphs

An important structure used to analyse a discrete problem is the interaction graph of factors [Darwiche, 2009].

**Definition 4 (Interaction graph).** Let $V$ be a set of vertices and $E$ a set of edges. A *factor interaction graph* $G{=}(V,E)$ of a set of factors $\{f_1,\ldots,f_n\}$ is an undirected graph with a vertex $v_i \in V$ for each variable $x_i$ and an edge between two nodes, $v_j$ and $v_k$ when the corresponding variables $x_j$ and $x_k$ appear together in at least one factor (we say that $x_j$ and $x_k$ co-appear or *interact*).

Before we are able to eliminate a variable $v$ when applying variable elimination, we must first multiply all factors in which $v$ appears. After eliminating $v$, we obtain a factor containing all the variables that were in the multiplied factors. In the factor interaction graph, these two steps correspond to connecting all the neighbors of $v$ with each other and removing $v$ from the graph. The additional edges are called *fill-in edges*. In this process, the number of variables in the resulting factor is equal to the number of neighbors of $v$ while the size of a factor is exponential in the number of variables.

**Example 3.** In the discrete example of factors $P(A), P(B|A)$ and $P(C|B)$, $A$ interacts with $B$ and $B$ interacts with $C$ (Figure 2, middle). Eliminating $B$ results in new interaction between $A$ and $C$ (Figure 2, left) while eliminating $A$ does not result in any new interactions (Figure 2, right). This shows that first eliminating $A$ is more beneficial as the intermediate factor $f(A,B)$ is smaller (fewer neighboring variables).

In order to utilize the concept of interaction graph for hybrid domains, we introduce the concept of an **interaction graph of atomic SMT($\mathcal{LRA}$) literals**. Such an interaction graph is obtained by interpreting atomic SMT literals as factors. The vertices in the interaction graph then correspond to real variables appearing in the atomic

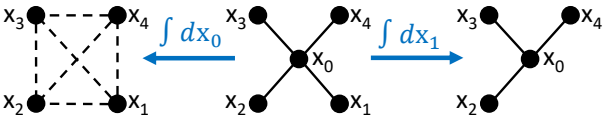



Figure 3: The interaction graph of Example 2 (middle), the graph when $x_0$ is integrated out (left) and the graph when $x_1$ is integrated out (right).

SMT literals and they are connected to each other if they jointly appear in at least one atomic SMT literal. This elegant mapping of SMT literals to factors allows us to deploy the plethora of concepts on variable ordering developed for the discrete setting in the hybrid setting.

**Example 4.** In Equation 8, $x_0$ interacts with all other continuous variables (Figure 3, middle). Integrating out $x_0$ results in new inequalities in which $x_1, x_2, x_3$ and $x_4$ interact with each other (Equation 9). In the interaction graph this implies the removal of $x_0$ and the addition of new edges (dashed, Figure 3, left). When integrating out $x_1$ instead, no new edges are introduced in the interaction graph (Figure 3, right), implying a more compact intermediate result (Equation 10).

If a continuous variable $v$ occurs in multiple inequalities with other continuous variables $X$, integrating out $v$ will yield new inequalities between all the variables of $X$. This process also exhibits an exponential relation for the hybrid setting between the number of neighbors for $v$ and the size of the result after integrating out $v$. This is related to the exponential complexity of Fourier-Motzkin elimination [Imbert, 1990].

An interaction graph and a variable ordering together form an ordered graph. The following two definitions are from [Dechter, 2013].

**Definition 5 (Ordered graph).** Given an undirected graph $G = (V, E)$, the *ordered graph* $(G, d)$ is obtained by ordering the nodes along ordering $d$. The *parents* of a node $v$ are the nodes connected to $v$ (see $E$) which occur earlier in the ordering. The *width of node* $v$ in $(G, d)$ is the number of parents $v$ has. The *width of ordered graph* $(G, d)$ is the maximum width of all nodes in $(G, d)$.

**Definition 6 (Induced ordered graph).** An *induced ordered graph* $(G^*, d)$ of $(G, d)$ is an ordered graph obtained from $(G, d)$ by processing the nodes in reverse order of $d$ (last to first, top to bottom). A node is processed by adding edges between all its parents. The *induced width of ordered graph* $(G, d)$ is the maximum number of parents any node has in $(G^*, d)$. The *induced width of graph* $G$ is the minimal induced width over all possible orderings $d$.

The process to construct $(G^*, d)$ matches the behavior of an interaction graph when eliminating variables in the reverse order of $d$. Given a variable elimination approach for a discrete setting with starting interaction graph $G$ and elimination ordering $d$, the number of variables in the largest intermediate factor is equal to the induced width of $(G, d)$ plus one. The time and space complexity of the variable elimination approach is exponential in the induced width [Dechter, 2013].

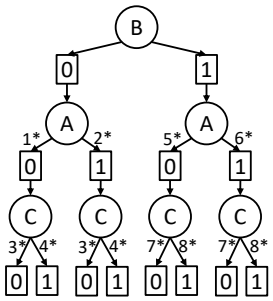
### 3.2.2 Heuristic Variable Ordering

Given an interaction graph $G$, $d$ should be chosen such that the induced width of $(G, d)$ is minimal. This minimises the size of the intermediate factors for the discrete setting and the size of the resulting equation (symbolic expression tree) for the hybrid setting. Unfortunately, finding the minimum induced width of a graph is NP-complete in general [Dechter, 2013]. Nevertheless, there are reasonable heuristics such as min-degree, min-induced-width and min-fill. *Min-degree* constructs the ordering $d$ for interaction graph $G$ in reverse order by iteratively selecting the variable $v$ with the lowest degree in $G$ and removing $v$ and its edges from $G$. This idea is also used in Linear Decision Diagrams [Chaki et al., 2009] to perform existential quantification of continuous variables. Min-induced-width and min-fill are similar but connect all neighbors of $v$ before removing it. *Min-induced-width* selects the node with the lowest degree but, because of the modification, also accounts for previously added edges. *Min-fill* selects $v$ based on the minimum number of edges required to connect the neighbors (the fill-in edges). None of the heuristics work best on all problems. In general, min-fill has shown to be usually slightly better than min-induced-width and min-degree has shown to be the worst of the three [Koller and Friedman, 2009, Dechter, 2013, Kask et al., 2011].

The three heuristics originate from the work in the discrete setting. When we construct the interaction graph for the hybrid case by treating atomic SMT($\mathcal{LRA}$) literal as factors, the interaction graph provides the same kind of information as for the discrete case (minimise the induced width). We can therefore also apply these heuristics to the hybrid setting.

## 4 VARIABLE TREES

Instead of performing computations on factors (cf. Section 3), we investigate a search-based approach which consists of recursively conditioning on variables. Consider for example $\sum_A P(B = 1|A)P(A)$, previously solved by taking the product of both $P(B = 1|A)$ and $P(A)$ before eliminating $A$. A search based approach solves this problem by first conditioning on $A = 0$, com-




| Nr. | Factors |
|---|---|
| 1* | $P(A{=}0)P(B{=}0\|A{=}0)$ |
| 2* | $P(A{=}1)P(B{=}0\|A{=}1)$ |
| 3* | $P(C{=}0\|B{=}0)$ |
| 4* | $P(C{=}1\|B{=}0)$ |
| 5* | $P(A{=}0)P(B{=}1\|A{=}0)$ |
| 6* | $P(A{=}1)P(B{=}1\|A{=}1)$ |
| 7* | $P(C{=}0\|B{=}1)$ |
| 8* | $P(C{=}1\|B{=}1)$ |

Figure 4: OR-tree with $d = B, A, C$ and table of weights (x*).

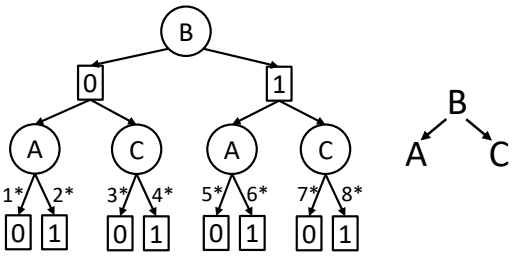



Figure 5: AND/OR-tree (left) and its guiding variable tree (right). $B = 0$ (and $B = 1$) splits into two OR-nodes, $A$ and $C$, indicating an AND-node.

puting the product, and summing it with the result of conditioning on $A = 1$. The advantage of this approach, when evaluated in a depth-first-manner, is that it requires less memory compared to reasoning over complete factors ($A = 0$ and $A = 1$ at the same time). An extension of this approach exploits independencies that result from conditioning on variables. The order in which variables are branched on, is in this extension a tree of variables instead of a simple variable ordering.

### 4.1 AND/OR Graphs

An *OR-tree* is formed by repeatedly conditioning variables according to variable ordering $d$ (Figure 4). Each conditioning represents an OR-node (circle) branching on the different possible values for the variable. A path or trace in the tree represents an assignment to each variable such that in a leaf node the variables in all factors have been instantiated to a value. Instead of computing the weight for each leaf as the product of instantiated factors, distributivity is used to push instantiated factors as close to the root as possible. This means that as soon as all variables of a factor have been instantiated, the value of the factor can be taken into account (placed on the edge), reducing computations [Dechter, 2013]. More formally, each factor $f(\mathbf{X})$ can be taken into account when all values for variables $\mathbf{X}$ have been assigned a value.

**Example 5.** The two leaves on the left of Figure 4 only differ in the assignment for $C$. Instead of computing $P(A = 0)P(B = 0|A = 0)P(C = 0|B = 0)$ for the left leaf, $P(A = 0)P(B = 0|A = 0)P(C = 1|B = 0)$ for the right leaf and summing up the results, distributivity can be used to push the shared part, $P(A = 0)P(B = 0|A = 0)$, higher in the tree.

An AND/OR tree is an extension that exploits more independencies. For example, after conditioning on $B$, $P(C|B)$ becomes independent from $P(A)P(B|A)$ (Figure 5). This is used to split the computations in multiple parts (AND-node with a branch for $A$ and one for $C$), graphically represented by connections going to multiple OR-nodes after a decision. When a subtree occurs multiple times, the parents can refer to the same subtree, reusing the computations. This behavior can be obtained through caching and results in graphs instead of trees. The AND/OR structure is not guided by a simple variable ordering but by a variable tree (Figure 5).

A guiding tree is only valid for a problem when variables that co-occur in a factor are not split over different AND branches. Given a problem with interaction graph $G$, any pseudo tree of $G$ is a valid guiding tree for that problem [Dechter, 2013].

**Definition 7.** A *pseudo tree* of interaction graph $G = (V, E)$ is a directed rooted tree $\mathcal{T} = (V, E')$ with the back-arc property. This property states that for each edge $e$, if $e \in E$ and $e \notin E'$ then $e$ is a back-arc edge, i.e. an edge that connects a node with one of its ancestors [Dechter, 2013]. The back-arc property ensures that variables that occur in the same factor are not split over different AND branches.

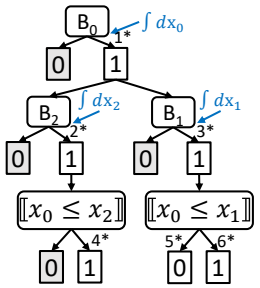
For example, when $A$ and $C$ would have co-occurred, an AND node cannot split the factors. In the guiding tree there would be a connection between $A$ and $C$ (Figure 5), violating the back-arc property.

So far, we conditioned on discrete variables, branching over the values in their domains. This is more challenging for continuous variables. As a solution, we propose to branch on the atomic SMT($\mathcal{LRA}$) literals instead, which can be considered as branching over different value intervals. This also implies that the guiding tree will contain SMT literals instead of continuous variables.

**Example 6.** The AND/OR graph in Figure 6 represents the following problem,

$$\int \Big[ c_1 [\![x_0 \leq x_1]\!] [\![x_0 \leq x_2]\!] (\textstyle\prod_{i=0,1,2} [\![0 \leq x_i \leq 1]\!]) + \quad (11)$$
$$c_2 [\![x_0 > x_1]\!] [\![x_0 \leq x_2]\!] (\textstyle\prod_{i=0,1,2} [\![0 \leq x_i \leq 1]\!]) \Big] dx_0 dx_1 dx_2$$

with $B_i = [\![0 \leq x_i \leq 1]\!]$ and $c_1$ and $c_2$ as two constants. The weight of all grey decisions is 0 and for all others, it is equal to the decision itself (see table). The structure allows parallel integration of $x_1$ and $x_2$ (Figure 7).




| Nr. | Weights |
|---|---|
| 1* | $[\![0 \leq x_0 \leq 1]\!]$ |
| 2* | $[\![0 \leq x_2 \leq 1]\!]$ |
| 3* | $[\![0 \leq x_1 \leq 1]\!]$ |
| 4* | $[\![x_0 \leq x_2]\!]$ |
| 5* | $c_2[\![x_0 > x_1]\!]$ |
| 6* | $c_1[\![x_0 \leq x_1]\!]$ |

Figure 6: AND/OR Graph and weight table (x*) for the continuous setting.

To evaluate this structure, perform $+$ and $\times$ bottom-up for each OR- and AND-node and integrate out continuous variables as soon as possible. For example, after obtaining $[\![x_0 \leq x_2]\!]B_2$, we can integrate out $x_2$ as it does not occur at any later point. The order in which continuous variables can be integrated out forms an integration tree (Figure 7, left).

**Definition 8.** An *integration tree* is a pseudo tree where each node is associated with a continuous variable, except for the root node where it is optional. When the interaction graph contains disconnected subgraphs, the root of the integration tree can be empty.

Previously the ordering to compute the induced-width was the variable ordering $d$, now it is specified by the ancestor relation in the integration tree.

Integrating out continuous variables yields new inequalities, much like the intermediate factors created in the discrete setting. However, our continuous approach branches on SMT literals (corresponding to factors in the discrete setting). Do note that newly introduced inequalities do not become part of the search structure, only of the intermediate computations. The complexity of evaluating the structure is influenced by the depth of the guiding tree and the complexity of the partial integrations and their intermediate results. The latter is related to the induced-width of the integration tree. While the guiding tree affects the size of the structure, we empirically found minimising the integration time to be more important. Hence, we propose to first use heuristics to find an integration tree and only then convert the tree into a pseudo tree of SMT($\mathcal{LRA}$) literals that respects this integration order (Figure 7).

**Sentential Decision Diagrams** We explained the role of the variable ordering and how to analyse its influence, in the context of AND/OR graphs. We do stress that our AND/OR graph in the context of continuous variables is solely illustrative. F-XSDD(BR), the state-of-the-art approach that we extend and evaluate in the experiments, uses SDDs. Even though there are many

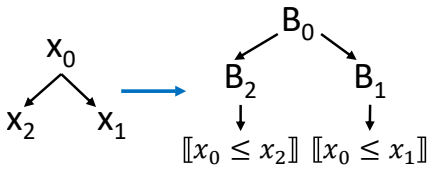



Figure 7: Integration tree and a guiding tree respecting it.

diferences between these two structures (e.g. conditioning on a variable versus a sentence in SDDs), the role of the variable ordering and its influence on the computations remains the same. We illustrate the influence using AND/OR graphs as its existing literature on orderings is closer in focus to our approach. A pseudo tree of SMT literals used to guide the construction of an AND/OR graph can easily be translated into a vtree to guide an SDD. Boolean variables are not constrained by the integration tree. In our implementation we use them to heuristically balance the vtree.

### 4.2 Peudo-Tree Heuristics

The size (and complexity) of AND/OR graphs are controlled by their guiding tree. Finding a minimal height pseudo tree is, similar to minimum induced width, NP-complete [Dechter, 2013]. We discuss three heuristics, the first two minimise the induced width, the third one the tree height. The second heuristic is novel, the first and third were adapted to the continuous setting.

**Top-Down Pseudo-Tree** This heuristic constructs a pseudo tree in two steps. First, obtain a variable ordering $d$ through, for example, the previously discussed min-fill approach. Second, given the induced interaction graph $G$ along $d$, a pseudo tree can be constructed top-down by traversing the induced-ordered $G$ in a depth first manner starting from the first variable in $d$ and prioritising variables earlier in $d$ to break ties [Dechter, 2013].

**Bottom-Up Pseudo-Tree** The first step in the previous approach, obtaining $d$, does not have any information on the second step, constructing the pseudo tree. When breaking ties, it hence does not consider the effects on the height of the resulting tree. We propose a new heuristic which interleaves both steps and constructs the pseudo tree bottom-up. By interleaving, the variable selection heuristic can consider the effect on the tree height and take decisions to minimise it. Our heuristic keeps track of several tree roots (branches that are being extended in parallel, bottom-up) which are iteratively extended with new variables. When a variable $v$ is added, it either 1) extends a root, 2) yields a new root or 3) combines multiple roots, depending on whether any of the variables in the current trees were previously neighbors (interacted with) of $v$. The next variable to add to the trees is selected using

a min-fill metric, breaking ties by prioritising the variable that results in the most shallow trees. We refer to our heuristic as *balanced bottom-up min-fill* (**BU-MiF**).

**Minimize Height** To minimise the height of a pseudo tree, a hypergraph decompositioning approach can be used to create a (roughly) balanced tree. To convert the problem into a hypergraph, create a vertex for each factor and a hyperedge for each variable $v$, connecting all factors that contain $v$. A pseudo tree can be obtained from the hypergraph by recursively partitioning the vertices into two (roughly) balanced sets while minimising the cut (hyperedges crossing the two sets) [Dechter, 2013]. When a variable is instantiated (cut), factors can become independent and can be solved separately (AND-node).

**Continuous Setting** The first two heuristics can be applied for continuous variables by changing the variable selection process to use interaction graphs adapted to the continuous setting (Section 3). For the continuous setting, these heuristics return an integration tree, providing the order in which to integrate out continuous variables (Figure 7, left). When employing an approach that conditions on SMT($\mathcal{LRA}$) literals instead of continuous variables, the integration tree must first be converted into a guiding tree of literals respecting that ordering (Figure 7, right). When using SDDs, the integration tree should instead be converted into a vtree. This two-step decomposition is not present in the discrete case and is crucial to apply these heuristics to the hybrid setting.

Using the hypergraph decompositioning approach, a vtree can also be created directly by recursively partitioning the literals in two sets (minimising the cut), forming the left and right subtrees of the vtree. When using SMT literals as vertices and hyperedges as shared continuous variables, the min-cut has an additional meaning compared to the discrete setting. The min-cut is the set of variables shared by the SMT literals in both sets, indicating the depth at which those variables can be integrated out. By minimising this cut, we minimise the number of variables that can only be integrated out high in the structure, maximising deeper and smaller integrations.

## 5 EXPERIMENTS

PyWMI is a software package designed to solve WMI problems. It includes the state-of-the-art solver F-XSDD(BR) which compiles WMI problems to XSDDs heuristically minimising the height by balancing the vtree. This heuristic is agnostic to which continuous variables occur in a literal and how these continuous variables interact. We extend this solver with the vtree heuristics discussed in Section 4.2, yielding a more robust solver that no longer has to rely on a user-provided orderings. This process consists of constructing 1) the interaction graph of the problem, 2) the integration tree using the discussed heuristics, 3) a vtree that respects the ordering of the integration tree, 4) the XSDD using that vtree and 5) evaluating the XSDD to obtain the result.

We consider seven problem templates whose size is controlled by parameter n: *dual(n)*, *xor(n)*, *mutex(n)*, *click(n)*, *star(n)*, *3ary(n)* and *path(n)* [Kolb et al., 2019b, Zeng and Van den Broeck, 2019]. The last three problem classes belong to a subset of tractable WMI problems. The SMI solver [Zeng and Van den Broeck, 2019] is specialized to exploit this type of problem structure and has outperformed F-XSDD(BR) on these problems. We evaluate three sets of heuristics: 1) hypergraph decompositioning (HG-MC) and top-down min-fill (TD-MiF), both of which we adapted from the discrete setting; 2) our new bottom-up min-fill heuristic (BU-MiF); and 3) F-XSDD(BR)'s current heuristic (*balanced*) and a right-linear heuristic (corresponding to a chain variable ordering). For the first four problems we also compare to the *Manual* approach (balanced heuristic + manual variable input order), for the last three we compare to SMI.

For every problem, each heuristic is ran 10 times with randomized orderings for increasing $n$. The maximum, minimum, and average run times are recorded. For the first four problems we run up to $n = 35$ with time-out $t = 30s$, for the last three up to $n = 40$ with $t = 60s$. If, in one iteration, an heuristic times out for a given value of $n$, its run-time is set to the time-out, and larger values of $n$ are skipped. All results are shown in Figure 8. Code is available at https://github.com/VincentDerk/BU-MiF.

**Q1: How does top-down min-fill (TD-MiF) compare to the newly introduced balanced bottom-up min-fill (BU-MiF) heuristic?** Both heuristics are very similar. However, the bottom-up approach is a lot more consistent, breaking ties by focusing on the balance of the integration tree. This is especially apparent on *xor(n)*.

**Q2: Do the contributed heuristics improve the problem agnostic heuristics (balanced and right-linear)?** By analysing the problem, we perform significantly better than the previous, problem agnostic heuristics. The results also indicate that the proposed BU-MiF heuristic is less susceptible to unfavorable input orders from the user, yielding a faster and more robust solver. In addition, we also improve over SMI. We found that SMI spends a lot of time finding the integration intervals. The complexity of SMI can be super exponential in the worst case [Zeng and Van den Broeck, 2019], for instance, with a path primal graph such as in *path(n)*. BU-MiF does not always recover the best ordering. In *click(n)*, the difference in run time is caused by a large difference in SDD sizes (for Manual, the SDD size was

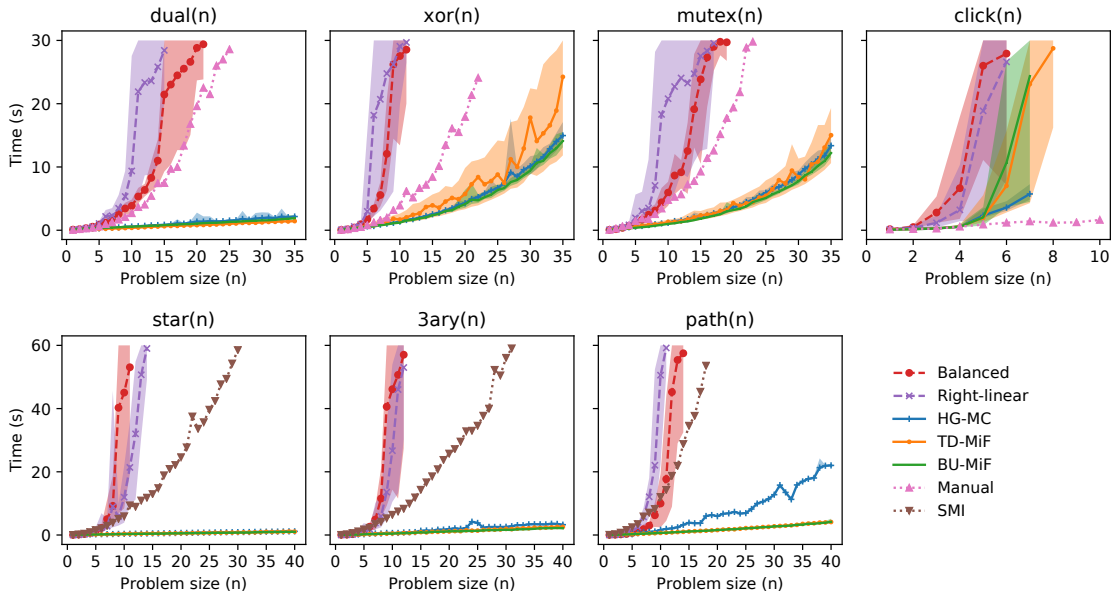



Figure 8: Comparison of run times for different variable ordering heuristics. For the F-XSDD(BR) solver, the run times include time spent on the variable ordering heuristics (negligible), the compilation step and the evaluation step.

5 for both $n = 6$ and 7 while for BU-MiF it was 6145 and 1537). We currently convert the integration tree into a vtree that respects the integration order and balances the literals (including Boolean variables) to minimise the depth of the SDD. In future work we can optimize this conversion by analyzing the logical theory to obtain more succinct SDDs that still respect the integration order.

**Q3: Should we minimise the induced-width or the depth of the integration tree?** We compared the hypergraph decomposition heuristic (HG-MC) with BU-MiF (min-fill metric to minimise the induced-width). In general, they seem to perform similar. We computed the induced-width of the solutions returned by both approaches and found that, except for *path(n)*, they had the same induced-width. The *path(n)* problem, where depth was prioritised over induced-width, suggests that optimising the induced-width is more important.

## 6 CONCLUSION

A crucial element of performing efficient probabilistic inference over discrete random variables is the order in which variables are marginalized out. In this paper we have shown that the importance of variable ordering also extends to problems in the discrete-continuous domain. We analyzed the influence of the variable ordering in the continuous setting by identifying parallels between probabilistic inference over discrete and continuous random variables and mapping concepts from the discrete setting, such as interaction graphs, to the continuous setting. This allowed us to adapt variable ordering heuristics developed for discrete random variables to perform probabilistic inference over continuous ones.

We introduced a new heuristic (BU-MiF), which significantly outperforms previous heuristics (Balanced, Right-linear) and is more robust than the heuristics adapted from the discrete setting (HG-MC, TD-MiF). BU-MiF also allows F-XSDD(BR) to outrun the specialized SMI solver on a set of benchmark problems from the tractable WMI subclass it addresses.

In future work we aim to exploit additional information about the logical structure of the WMI support. This could lead to smaller (more succinct) compiled representations for problems such as the *click(n)* problem. An adaptation of our heuristic to an iterative anytime scheme can also be considered [Kask et al., 2011].

### Acknowledgements

This work has received support from the Research Foundation - Flanders (1SA5520N), the Special Research Fund of the KU Leuven, the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 694980), the Flemish Government (AI Research Program) and the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. The authors thank Adnan Darwiche and YooJung Choi for discussions on variable ordering for the discrete case.

## References


[Arnborg, 1985] Arnborg, S. (1985). Efficient algorithms for combinatorial problems on graphs with bounded, decomposability—a survey. *BIT*, 25(1):2–23.

[Belle et al., 2015] Belle, V., Passerini, A., and Van den Broeck, G. (2015). Probabilistic Inference in Hybrid Domains by Weighted Model Integration. In *IJCAI*.

[Bryant, 1986] Bryant, R. E. (1986). Graph-based algorithms for boolean function manipulation. *Computers, IEEE Transactions on*, 100(8):677–691.

[Chaki et al., 2009] Chaki, S., Gurfinkel, A., and Strichman, O. (2009). Decision diagrams for linear arithmetic. In *FMCAD*, pages 53–60. IEEE.

[Chavira and Darwiche, 2008] Chavira, M. and Darwiche, A. (2008). On probabilistic inference by weighted model counting. *Artificial Intelligence*, 172(6-7):772–799.

[Darwiche, 2009] Darwiche, A. (2009). *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press.

[Darwiche, 2011] Darwiche, A. (2011). Sdd: A new canonical representation of propositional knowledge bases. In *IJCAI*.

[Darwiche and Marquis, 2002] Darwiche, A. and Marquis, P. (2002). A Knowledge Compilation Map. *J. Artif. Int. Res.*, 17(1):229–264.

[Dechter, 1999] Dechter, R. (1999). Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113(1-2):41–85.

[Dechter, 2013] Dechter, R. (2013). *Reasoning with Probabilistic and Deterministic Graphical Models: Exact Algorithms*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers.

[Imbert, 1990] Imbert, J.-L. (1990). About redundant inequalities generated by Fourier's algorithm. In *Artificial Intelligence*. Elsevier.

[Kask et al., 2011] Kask, K., Gelfand, A., Otten, L., and Dechter, R. (2011). Pushing the power of stochastic greedy ordering schemes for inference in graphical models. In *AAAI*.

[Kjærulff, 1990] Kjærulff, U. (1990). Triangulation of graphs–algorithms giving small total state space. *Technical report*.

[Kolb et al., 2018] Kolb, S., Mladenov, M., Sanner, S., Belle, V., and Kersting, K. (2018). Efficient Symbolic Integration for Probabilistic Inference. In *IJCAI*.

[Kolb et al., 2019a] Kolb, S., Morettin, P., Martires, P. Z. D., Sommavilla, F., Passerini, A., Sebastiani, R., and Raedt, L. D. (2019a). The pywmi framework and toolbox for probabilistic inference using weighted model integration. In *IJCAI*.

[Kolb et al., 2019b] Kolb, S., Zuidberg Dos Martires, P., and De Raedt, L. (2019b). How to exploit structure while solving weighted model integration problems. In *UAI*.

[Koller and Friedman, 2009] Koller, D. and Friedman, N. (2009). *Probabilistic Graphical Models - Principles and Techniques*. MIT Press.

[Morettin et al., 2017] Morettin, P., Passerini, A., and Sebastiani, R. (2017). Efficient Weighted Model Integration via SMT-Based Predicate Abstraction. In *IJCAI*.

[Pearl, 1982] Pearl, J. (1982). Reverend bayes on inference engines: a distributed hierarchical approach. In *AAAI*, pages 133–136.

[Pipatsrisawat and Darwiche, 2008] Pipatsrisawat, K. and Darwiche, A. (2008). New compilation languages based on structured decomposability. In *AAAI*, volume 8, pages 517–522.

[Valiant, 1979] Valiant, L. G. (1979). The complexity of computing the permanent. *Theoretical Computer Science*, 8(2).

[Zeng and Van den Broeck, 2019] Zeng, Z. and Van den Broeck, G. (2019). Efficient search-based weighted model integration. In *UAI*.

[Zhang and Poole, 1994] Zhang, N. L. and Poole, D. (1994). A simple approach to bayesian network computations. In *CSCSI*, pages 171–178.

[Zuidberg Dos Martires et al., 2019] Zuidberg Dos Martires, P., Dries, A., and De Raedt, L. (2019). Exact and Approximate Weighted Model Integration with Probability Density Functions Using Knowledge Compilation. In *AAAI*.