**Optimal decision-making under constraints and uncertainty**
Latour, A.L.D.

**Citation**
Latour, A. L. D. (2022, September 13). *Optimal decision-making under constraints and uncertainty*. *SIKS Dissertation Series*. Retrieved from https://hdl.handle.net/1887/3455662

| | |
|---|---|
| Version: | Publisher's Version |
| License: | [Licence agreement concerning inclusion of doctoral thesis in the Institutional Repository of the University of Leiden] |
| Downloaded from: | https://hdl.handle.net/1887/3455662 |

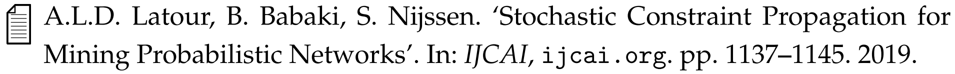
**Note:** To cite this publication please use the final published version (if applicable).

# 6

# Global SCMD propagation

In this chapter, we first show that a weakness of the decomposition methods presented in the previous chapter is that they do not guarantee *generalised arc consistency (GAC)*. This may cause them to not prune parts of the search space that do not contain any solutions. Here, we show why that is the case, and that a straightforward modification of these methods such that they do guarantee GAC, does not notably improve solving times. For the specific case of *stochastic constraint (optimisation) problems (SCPs)* on *monotonic* probability distributions (which are the probability distributions on which *stochastic constraint (optimisation) problems on monotonic distributions (SCPMDs)* are formulated), we propose an alternative method: a new propagator for a global *ordered binary decision diagram (OBDD)*-based constraint. We show that this propagator has a time complexity that is linear in the size of the OBDD and maintains GAC. We experimentally evaluate the effectiveness of this global constraint in comparison to decomposition-based approaches, using problems from the data mining literature. We find that the approach that uses this global *stochastic constraint on monotonic distributions*

*(SCMD)* propagator outperforms the *constraint programming (CP)*-based decomposition methods and performs complementarily to the *mixed integer programming (MIP)*-based decomposition method. This chapter is based on the following publication:


A.L.D. Latour, B. Babaki, S. Nijssen. 'Stochastic Constraint Propagation for Mining Probabilistic Networks'. In: *IJCAI*, `ijcai.org`. pp. 1137–1145. 2019.

## 6.1 Introduction

Recall the spread of influence and power grid reliability examples of SCPs described in Section 1.1. We observe that, in the spread of influence problem, adding a person to the set of people who receive a free product sample can never decrease the expected number of people that will become customers. Similarly, in the power grid reliability problem, we observe that adding a power line to the set of lines that are reinforced can never decrease the expected number of households that still have power after a disaster.

Clearly, the probability distributions in these SCPs have a characteristic in common: the probabilities and expectations are higher if more nodes or edges are selected, which makes these probability distributions *monotonic*. This makes them special cases of SCPs, namely *stochastic constraint (optimisation) problems on monotonic distributions (SCPMDs)*.

While this characteristic seems limiting, problems that have this property are plentiful in network analysis; examples include the applications mentioned above, but also the *signalling-regulatory pathway inference* problem described in the bioinformatics literature [50, 133] and in Section 4.5.1, and a variant on the landscape connectivity problem [185].

In this chapter we present an approach to solving SCPMDs (which can only be applied to those, not to SCPs in general). We use OBDDs to model the probability distributions and impose a stochastic constraint on the *arithmetic circuit (AC)* that can be used to compute probabilities from these OBDD representations. Crucially, we exploit structures in those OBDD representations that result from monotonicity to obtain a *global* constraint propagation algorithm for solving SCMDs. Recall from Section 3.3.3 that global constraints can have greater propagation and search tree pruning power than local constraints, like the ones used in the decomposition method of Chapter 5. Thus, provided that the time and space complexity of a global constraint are polynomial, global constraints can have an advantage over local constraints by potentially pruning the search space better (by removing more values from the domains of free variables during inference).

The main algorithmic contributions in this chapter are the following:

1. We show that the decomposition approach described in Section 5.2.1 is not GAC, thus causing it to prune the search space insufficiently (Section 6.2), and that a straightforward arc consistent modification of this approach does not significantly improve performance (Section 6.5).
2. To address this inefficiency in the search, we introduce a global constraint on OBDD representations of monotonic distributions, which we call the SCMD (Section 6.3), and introduce a GAC-by-design propagation algorithm for this constraint (Section 6.4).

In summary, the benefits of the decomposition methods described in Chapter 5 and Section 6.2 are:

- They are applicable to the more generic SCPs (Section 1.2).
- Different types of *decision diagrams (DDs)* can be used to represent the probability distributions (Sections 5.2.1 and 5.2.2).
- The implementation is straightforward and compatible with different off-the-shelf CP or MIP solvers (Section 5.2.3).

Conversely, the benefits of our global constraint specifically for SCPMDs are:

- It guarantees GAC by design, contrary to decomposition methods that do not guarantee GAC, and therefore traverses the search space more efficiently (Section 6.4.1).
- Its space complexity is better than that of decomposition methods that do guarantee GAC (Sections 6.2 and 6.4.3).
- Its worst-case time complexity is $O(m + n)$ with OBDD size $m$ and $n$ decision variables (Section 6.4.3).
- It outperforms CP-based decomposition methods and complements MIP-based methods, while scaling better with OBDD size than MIP-based methods (Section 6.5).

The main feature that distinguishes our work from similar works on stochastic constraint satisfaction and optimisation is that we exploit the structure of the probability distribution in our global SCMD propagator. These structures arise from the fact that SCPMDs are formulated on *monotonic distributions*. In exploiting those structures, our method distinguishes itself from more general approaches taken earlier [181], and from the majority of existing methods, which sample scenarios from a distribution, and hence ignore such structures [78].

The remainder of this chapter is organised as follows. First, in Section 6.2 we revisit probabilistic inference with OBDD representations of probability distributions, showing how the decomposition methods of Chapter 5 can be made to guarantee GAC. We then define monotonic probability distributions and how they relate to OBDDs in Section 6.3, using this monotonicity to define a SCMD. Then, in Section 6.4, we describe three global SCMD propagation algorithms that each preserve GAC by design: a naïve algorithm with quadratic time complexity, a more efficient algorithm with linear time complexity, and an incremental version of that last algorithm that has the potential to be more efficient in practice. In Section 6.5, we compare the performance of the linear-time global propagation algorithms to CP-based and MIP-based decomposition methods in solving SCPMDs. We conclude this chapter in Section 6.6 with a brief summary of our approach and results, and recommendations for future research.

## 6.2 OBDDs and generalised arc consistency

When using a CP solver to solve the decomposed constraint on the probability distribution represented by an OBDD, we encounter the following problem:

**Theorem 6.2.1.** *Propagation on the decomposed representation of the SCMD as described in Section 5.2.1 is not GAC.*

*Proof.* Assume that propagation in the decomposition method in Section 5.2.1 is GAC (Section 3.3). Then, the following counterexample leads to a contradiction.

Consider the OBDD in Figure 5.1 and associated constraint $P(\phi \mid \sigma) \geq 0.4$. Observe that the four possible strategies yield these conditional probabilities:

$$P(\phi \mid X = Y = 0) = 0 \qquad P(\phi \mid X = 1, Y = 0) = 0.3$$
$$P(\phi \mid X = Y = 1) = 0.6 \quad P(\phi \mid X = 0, Y = 1) = 0.6$$

From this, we conclude that only those strategies in which $Y = 1$ holds can possibly satisfy the constraint. A propagator that ensures GAC on the Boolean variables will detect this before the start of the search and fix $Y := 1$.

Suppose a constraint propagator is called on the decomposed model in Figure 5.1, before the search starts. This propagator may start by trying to infer the minimum value that $Z_{Y_1}$ needs to take if $Z_X$ takes its maximum possible value. To do this, the propagator assumes that $Z_X = 0.6$ holds. Now it can infer that, in order for the constraint to be satisfied, $Z_{y_1} \geq {}^{(0.4 - 0.9 \cdot 0.6)}/_{0.1} = -1.4$ should hold. Unfortunately, it already knew that $dom(Y) = \{0, 1\}$ and thus does not include $-1.4$. Based on this, it cannot remove 0 from $dom(Y)$. Repeating a similar

procedure to determine a bound for $Z_X$, $Z_{Y_1}$ and $Z_{Y_2}$ does not yield conclusive evidence to deduce that $Y$ must be fixed to 1, either. □

As a result, the search tree of a CP system is unnecessarily large. One solution may seem to create a decomposed representation that is GAC. We can achieve this by means of two modifications to the decomposition method. First, we replace the encoding of the score of OBDD node $r_D$, $v(r_D) := v\left(r_D^+\right)$ if $D = \top$ and $v(r_D) := v\left(r_D^-\right)$ if $d = \bot$, with $v\left(r_D\right) := \max\left(d \cdot v\left(r_D^+\right), (1-D) \cdot v\left(r_D^-\right)\right)$, because this improves propagation in cases where $D$ is yet unassigned. Additionally, we add the (redundant) constraint $v|_{D=0}(root) < \theta \rightarrow D := 1$ to the decomposition for each decision variable $d$. Here, $v|_{D=0}(root)$ represents the expression at the root of the diagram, as obtained from Equation 2.11, conditioned on $D = \bot$. Note that adding the extra constraint for each decision variable $D$ requires us to make a copy of the original diagram, only with $D = \bot$.

The downside of this approach is that we need to add a large number of linear constraints to the model, resulting in a space complexity of $O(|\mathbf{D}| \cdot |OBDD| \cdot \tau)$ for this approach, where $\mathbf{D}$ is the set of decision variables, $|OBDD|$ the number of nodes in the OBDD, and $\tau$ the depth of the search tree. We demonstrate the practical inferiority of this approach in Section 5.3.

## 6.3 Monotonicity

A special case of the constraint in Equation 1.1 is one where we require each probability distribution $P\left(\phi|_\sigma\right)$ to be *monotonic*.

### 6.3.1 Monotonic probability distributions

Intuitively, taking the spread of influence problem as an example, monotonicity means that adding a person to the set of people who receive a free product sample, cannot decrease the expected number of eventual customers (Example 4.2.1). Likewise, taking the power grid reliability problem as an example, adding a power line to the set of lines that receive maintenance cannot decrease the expected number of households that still have power after a natural disaster (Example 4.2.2).

We formally define a *monotonic probability distribution* as follows:

**Definition 6.3.1.** *Let $\phi(\mathbf{D}, \mathbf{T})$ be a propositional formula on Boolean decision variables $\mathbf{D}$ and Boolean stochastic variables $\mathbf{T}$, as defined in Section 1.2. We call the probability distribution $P(\phi|_\sigma)$ a* monotonic distribution *if, for all strategies $\sigma$ and each $D \in \mathbf{D}$,*

*the following holds:*

$$P\left(\phi|_{\sigma_{D=\bot}}\right) \leq P\left(\phi|_{\sigma_{D=\top}}\right), \tag{6.1}$$

*where strategies $\sigma_{D=\bot}$ and $\sigma_{D=\top}$ only differ in the truth values that they assign to $D$ ($\bot$ and $\top$, respectively).*

## 6.3.2 Local monotonicity

For OBDD representations of probability distributions, we also define the concept of *local monotonicity*:

**Definition 6.3.2.** *Let $\phi(\mathbf{D},\mathbf{T})$, $\sigma$ and $P(\phi|_\sigma)$ be defined as in Section 1.2. We call an OBDD representation of a probability distribution whose score at the root equals $P(\phi|_\sigma)$* locally monotonic, *iff the following holds for any projected $\sigma$ (see Section 2.5.1):*

$$v\left(r_D^-\right) \leq v\left(r_D^+\right) \tag{6.2}$$

*for each OBDD node $r_D$ labelled with decision variable $D \in \mathbf{D}$, using Equation* 2.11 *(page 39) to compute $v\left(r_D^-\right)$ and $v\left(r_D^+\right)$.*

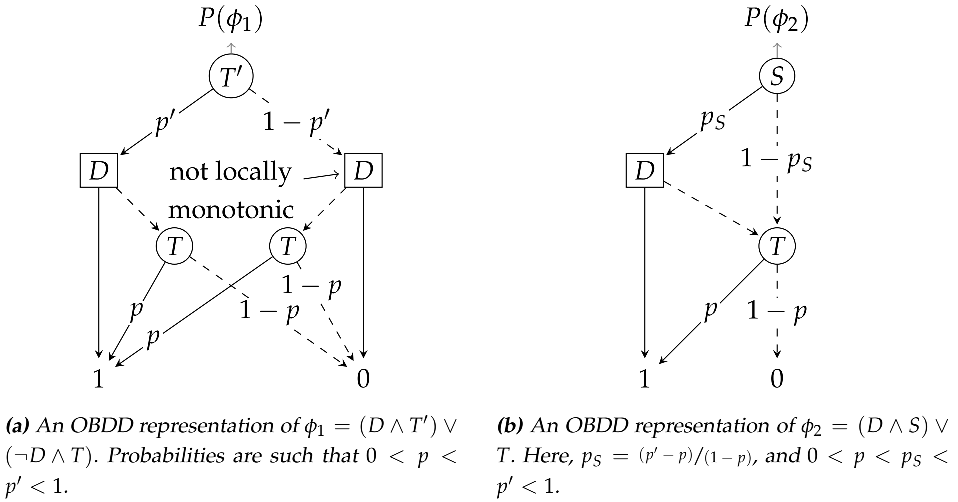
**Theorem 6.3.1.** *If a probability distribution $P(\phi|_\sigma)$ can be represented by a locally monotonic OBDD as defined in Definition 6.3.2, then it is a monotonic distribution, as per Definition 6.3.1.*

*Proof.* In the following, we use $v\left(r|_{\sigma_{D=\bot}}\right)$ to denote the score of an OBDD node $r$, computed using Equation 2.11, for an OBDD with strategy $\sigma$, in which decision variable $D = \bot$, and analogously for $D = \top$. Since the root of the OBDD is an OBDD node, the task is to prove that, for any node $r$ in a locally monotonic OBDD, it holds that $v\left(r|_{\sigma_{D=\bot}}\right) \leq v\left(r|_{\sigma_{D=\top}}\right)$. Here, $\sigma_{D=\bot}$ and $\sigma_{D=\top}$ only differ in the truth assignment of $D$. We prove this by induction.

The inequality holds trivially if $r$ is a leaf, and is in fact an equality in this case. We now assume that the inequality holds for all descendants of a node $r$, and distinguish the following cases:

1. Node $r$ is labelled with decision variable $D$.
2. Node $r$ is labelled with a decision variable other than $D$.
3. Node $r$ is labelled with a stochastic variable.

For the first case, the inequality holds by Definition 6.3.2. For the second case, $v(r)$ is determined by only one child, because $\sigma$ assigns a truth value to each decision variable, which fixes $w(r)$ to either 0 or 1 in Equation 2.11. Since the inequality holds for this one child, it also holds for $r$. For the third case, the inequality holds




***(a)** An OBDD representation of $\phi_1 = (D \wedge T') \vee (\neg D \wedge T)$. Probabilities are such that $0 < p < p' < 1$.*

***(b)** An OBDD representation of $\phi_2 = (D \wedge S) \vee T$. Here, $p_S = (p' - p)/(1 - p)$, and $0 < p < p_S < p' < 1$.*

***Figure 6.1:** Two different OBDD representations of the same probability distribution.*

for the two children. Since $v(r)$ is the weighted sum of those two children, the inequality also holds for $r$. $\square$

It is yet unknown if the reverse of Theorem 6.3.1 holds. Ensuring that distributions are monotonic is relatively easy in the *weighted model counting (WMC)* approach: for any representation written using ProbLog [52, 64] (and thus in SC-ProbLog) *without negation*, the resulting OBDD representation is locally monotonic, which renders the probability distribution monotonic. Note that this does not represent a strong limitation, since all problems discussed earlier can be written in this form.

The following example illustrates why distributions encoded using negation do not always yield locally monotonic OBDDs.

**Example 6.3.1** (An OBDD that is not locally monotonic)**.** *Recall the power grid reliability problem of Example 4.2.4, and let us focus on encoding the survival probability of a single power line. The decision to reinforce this power line or not, is denoted by Boolean decision variable $D$. Following the notation used in Example 4.2.4, we denote its survival probability when it is not reinforced $p$, and its survival probability when it is reinforced with $p'$, such that $0 < p < p' < 1$. The associated stochastic variables are $T$ and $T'$, which take value $\top$ if the line survives and the value $\bot$ if it does not. We then encode the event $\phi_1$ of survival of this power line, with the following formula:*

$$\phi_1 = (D \wedge T) \vee \left(\neg D \wedge T'\right). \tag{6.3}$$

*When we fix $D := \top$, this formula has two models: $\{T = \top, T' = \bot\}$ (with probability $p \cdot (1-p')$) and $\{T = T' = \top\}$ (with probability $p \cdot p'$), yielding a total success probability of $P(\phi_1|_{D=\top}) = p$. Similarly, we can show that $P(\phi_1|_{D=\bot}) = p'$. Since $p' > p$ by assumption, flipping D's truth value from $\bot$ to $\top$ does not decrease the survival probability of the power line, and thus the* distribution *is monotonic, according to Definition 6.3.1.*

*An example OBDD encoding of this formula is shown in Figure 6.1a. Consider the decision node on the right. If we set $D := \bot$, the score of that node is $p$. However, if we set $D := \top$, then the score in that node is* 0. *Consequently, flipping the value of D from $\bot$ to $\top$ is not locally monotonic (see Definition 6.3.2), even though the behaviour in the root of the OBDD remains monotonic. This local non-monotonic behaviour is directly due to the fact that we can falsify $\phi_1$ by fixing $D := \top$, because of the second clause.*

The following example shows how we can construct a monotonic probability distribution that does yield a locally monotonic OBDD.

**Example 6.3.2** (A locally monotonic OBDD). *Taking the same example of computing the survival probability of a single power line, consider this alternative encoding:*

$$\phi_2 = (D \wedge S) \vee T, \tag{6.4}$$

*again with decision variable $D$, and stochastic variable $T$ corresponding to survival of a power line that is not reinforced, with corresponding probability $p$. We associate a probability $p_S = (p'-p)/(1-p)$ with stochastic variable $S$, which exists to encode the probabilistic survival of the power line if it is reinforced. Note how Equation* 6.4 *does not contain negation. If we fix $D = \top$, the corresponding models of the residual formula are $\{S = \top, T = \bot\}$ (with probability $p_S \cdot (1-p)$), $\{S = T = \top\}$ (with probability $p \cdot p_S$) and $\{S = \bot, T = \top\}$ (with probability $(1-p_S) \cdot p$), bringing the total survival probability to*

$$\begin{aligned} P(\phi_2|_{D=\top}) &= \frac{p'-p}{1-p} \cdot (1-p) + p \cdot \frac{p'-p}{1-p} + \left(1 - \frac{p'-p}{1-p}\right) \cdot p \\ &= (p'-p) + p \cdot \left(\frac{p'-p}{1-p} + 1 - \frac{p'-p}{1-p}\right) \\ &= p' - p + p = p' \end{aligned}$$

*Similarly, we can show that $P(\phi_2|_{D=\bot}) = p$. Again, since $p' > p$ by assumption, this distribution is monotonic, according to Definition 6.3.1.*

*An example of an OBDD encoding of this formula is shown in Figure 6.1b. Note that the decision node in this OBDD displays locally monotonic behaviour, and thus the OBDD itself is locally monotonic, according to Definition 6.3.2.*

We will use the notion of local monotonicity to define a global propagation algorithm for SCMDs that guarantees GAC by design, in Section 6.4 .

### 6.3.3 Stochastic constraint on monotonic distributions

Using the notion of local monotonicity, we now define a corresponding SCMD as follows:

**Definition 6.3.3.** *For a set of propositional formulae $\Phi$, threshold $\theta \in \mathbb{R}^+$ and utilities $\rho_\phi \in \mathbb{R}^+$, we call*

$$\sum_{\phi \in \Phi} \rho_\phi \cdot P(\phi|_\sigma) > \theta \tag{6.5}$$

*a* stochastic constraint on monotonic distributions *if, and only if, all $P(\phi|_\sigma)$ can be represented by locally monotonic OBDDs.*

Given a partial strategy $\sigma$, a GAC-guaranteeing propagator for the SCMD will, for each unbound decision variable $D \in \mathbf{D}$, remove value *false* from $dom(D)$ if, and only if,

$$\sum_{\phi \in \Phi} \rho_\phi \cdot P\left(\phi|_{\sigma'}\right) \leq \theta \tag{6.6}$$

holds for each possible extension of partial strategy $\sigma$ to a full strategy $\sigma'$ that includes $D = \perp$.

## 6.4 Global SCMD propagation

We propose three global SCMD propagation algorithms that operate on the OBDD representations of monotonic probability distributions directly, and guarantee GAC by design. First, we describe a naïve version of such an algorithm, which has a time complexity that is quadratic in the size of the OBDD on which it operates. We then show how to improve this algorithm to make its complexity linear, instead. Then, we propose some optimisations to make this algorithm potentially even faster in practice. We end this section with a brief overview of a corresponding SCPMD solving pipeline.

In the general case, different propositional formulae can be encoded in one OBDD with multiple roots (one for each formula), avoiding redundancy if they share sub formulae (as discussed in Section 2.4). For simplicity of discussion and notation, we will only consider constraints over one propositional formula $\phi$ in this section, and thus only single-rooted OBDDs. This makes the corresponding utility $\rho$ irrelevant in the discussion and limits the domain of threshold $\theta$ to $(0, 1)$, to which we compare a probability rather than an expectation.

### 6.4.1 Naïve SCMD propagation

For maintaining GAC, a key observation is that our *scoring function* (the expected utility in Equation 6.5) is monotonic; hence, the largest possible score is obtained by assigning the value *true* to all unbound decision variables. Given an OBDD representation of $\phi(\mathbf{D}, \mathbf{T})$, mapped to an AC, the following process for each unbound decision variable $D \in \mathbf{D}$ would be GAC:

**Step 1:** Fix variable $D$ to the value $\bot$.

**Step 2:** Fix all remaining unbound variables to the value $\top$.

**Step 3:** Calculate the root node score for the resulting assignment with Equation 2.11.

**Step 4:** If the score is lower than or equal to threshold $\theta$, remove $\bot$ from $dom(D)$.

Fixing all unbound variables to $\top$ in Step 2 ensures that we compute an upper bound on the score given the current partial assignment and $D := \bot$ in Step 3, because of the local monotonicity of the OBDD, as defined in Definition 6.3.2. Consequently, if that upper bound is lower than $\theta$, we know that extending the current partial assignment to decision variables with $D := \bot$, results in a partial assignment that cannot be extended with assignments to the unbound variables into a solution whose score exceeds $\theta$. Thus, we update $D$'s domain in Step 4 to guarantee GAC. This algorithm does not require us to put constraints on the variable order of the OBDD to obtain the strict bound in Step 3, in contrast to previous work using *sentential decision diagrams (SDDs)* and *deterministic decomposable negation normal forms (d-DNNFs)* [145].

Let $n$ be the number of unbound decision variables, and let $m$ be the size of the OBDD (the number of nodes in the OBDD). Then the complexity of this *naïve SCMD propagator* is $O(m \cdot n)$: for every unbound variable, we perform a bottom-up traversal of the OBDD. Since propagation is the most computationally intensive part of search algorithms under our constraint, it is important to obtain better performance. Therefore, will show now how to improve this complexity to $O(n + m)$.

### 6.4.2 A full-sweep SCMD propagator

The key idea behind improving the naïve propagator is that we calculate a partial derivative

$$\frac{\partial f(D, \sigma' \setminus \{D\})}{\partial D} = f(\sigma') - f(D = \bot, \sigma' \setminus \{D\}) \tag{6.7}$$

for each unbound decision variable $D$. The function $f$ represents the scoring function defined by Equation 2.11 on the root of the OBDD. The strategy $\sigma'$ represents an assignment to all decision variables obtained by taking a partial assignment $\sigma$ and extending it by assigning *true* to each unbound decision variable in $\mathbf{D}$.

We use the derivative to remove the value *false* from the domains of variables that do not meet the following condition:

$$f(\sigma') - \frac{\partial f(D, \sigma' \setminus \{D\})}{\partial D} > \theta. \tag{6.8}$$

The main question becomes how to calculate the partial derivative for all unbound variables efficiently. Here, we build on ideas introduced by Darwiche [42, 44] to build a linear algorithm that can furthermore maintain derivatives incrementally. We first need to define the concept of *path weight*:

**Definition 6.4.1.** *Let $r_m$ be a node labelled with variable $X_m$ in an OBDD with variable order $X_1 \prec \ldots \prec X_n$. We define the* path weight of $r_m$ with respect to root $r$ *as*

$$\pi(r_m) := \rho_r \sum_{\ell \in L_{r_m}} \prod_{r_i \in \ell} u(i), \tag{6.9}$$

*where $\ell$ is a path from the root of the OBDD to $r_m$, $\rho_r$ is the reward associated with the query at root $r$, and $L_{r_m}$ is the set of all such paths that are* valid. *A path is* valid *if it does not include the hi (respectively, lo) arc from a node labelled with a decision variable that is* false *(*true *or* unbound*, respectively).*

*We define $u(i)$ as follows. For the outgoing arcs of decision nodes that* can *be part of a valid path, we use $u(i) := 1$; for outgoing arcs that cannot be part of a valid path, we use $u(i) := 0$. For the outgoing arcs of stochastic nodes labelled with a stochastic variable $X_i$ that has weight $w(i)$, we use:*

$$u(i) := \begin{cases} w(i) & \text{if we take the hi arc of } r_i; \\ 1 - w(i) & \text{if we take the lo arc of } r_i. \end{cases} \tag{6.10}$$

The path weight $\pi(r_m)$ is expressed in terms of variables $X_i \prec X_m$ only. In the general case, the path weights are initialised at the roots of the diagram (one root for each query) using the corresponding utility $\rho$. Because, for simplicity, we assume the diagram to have just a single root in this section, $\rho$ is irrelevant and the path weight at the root is initialised to 1. In the case of a multi-rooted OBDD, we simply sum all the $\pi_r(r_m)$'s for each root $r$ that is an ancestor of $r_m$.

Our global SCMD propagation algorithm is based on the following:

**Theorem 6.4.1.** *The partial derivative of the OBDD with respect to an unbound decision variable D can be calculated as follows:*

$$\frac{\partial f(D, \sigma' \setminus \{D\})}{\partial D} = \sum_{r_D \in OBDD_D} \pi(r_D) \left( v(r_D^+) - v(r_D^-) \right), \tag{6.11}$$

*where $OBDD_D$ is the set of OBDD nodes labelled with decision variable D.*

*Proof.* Observe that Equation 6.7 can be read as:

$$\frac{\partial f(D, \sigma' \setminus \{D\})}{\partial D} = v|_{\sigma' \setminus \{D\}, D=\top}(r) - v|_{\sigma' \setminus \{D\}, D=\bot}(r) \tag{6.12}$$

$$= \left( u_{r \to r^+} \cdot v|_{\sigma' \setminus \{D\}, D=\top}(r^+) + u_{r \to r^-} \cdot v|_{\sigma' \setminus \{D\}, D=\top}(r^-) \right) - \left( u_{r \to r^+} \cdot v|_{\sigma' \setminus \{D\}, D=\bot}(r^+) + u_{r \to r^-} \cdot v|_{\sigma' \setminus \{D\}, D=\bot}(r^-) \right), \tag{6.13}$$

where $r$ denotes the root of the OBDD and $v|_{\sigma' \setminus \{D\}, D=\bot}(r)$ the score at root $r$ (calculated using Equation 2.11), conditioned on partial strategy $\sigma$, extended by fixing $D$ to $\bot$ and all other unbound decision variables to $\top$.

The expression in Equation 6.12 states that the partial derivative of $f$ equals the difference of the score of Equation 2.11 taken at the root of the OBDD, conditioned on $\sigma'$ and either $D = \top$ or $D = \bot$. In Equation 6.13, we have expanded the expressions on each side of the '$-$'-sign according to Equation 2.11. Here, $r^+$ and $r^-$ represent the hi and lo children of the OBDD root $r$, respectively, and $u_{r \to r^+}$ and $u_{r \to r^-}$ are the corresponding weights of the outgoing arcs, according to the definition above.

We can continue this expansion recursively, until we find the $v|_{\sigma' \setminus \{D\}, D=\top}(r_D)$ or $v|_{\sigma' \setminus \{D\}, D=\bot}(r_D)$ terms, where we are computing Equation 2.11 in a node $r_D$ labelled with the unbound decision variable $D$ for which we are computing the derivative. The result is an expression that contains the following types of terms:

1. Constant terms, where we have expanded until we found either the '0' or the '1' leaf of the OBDD and replace the corresponding term accordingly.
2. Terms with $v(r_D^+)$ (from expansions of the $v|_{\sigma' \setminus \{D\}, D=\top}(r)$ term in Equation 6.12).
3. Terms with $v(r_D^-)$ (from expansions of the $v|_{\sigma' \setminus \{D\}, D=\bot}(r)$ term in Equation 6.12).

The terms of type 1 correspond to paths from the OBDD root to a leaf that do not contain an OBDD node labelled with $D$. Therefore, the terms on the right of the '$-$'-sign in Equation 6.13 cancel out those on the left.

Given a particular node $r_D$ in the OBDD, we have at least two terms for this node in the remaining expression: $u_{r \to r^+} \cdot \ldots \cdot u_{r_D \to r_D^+} \cdot v(r_D^+)$ and $-u_{r \to r^-} \cdot \ldots \cdot u_{r_D \to r_D^-} \cdot v(r_D^-)$. These two terms correspond to the same node $r_D$ and the same path $\ell$ from the root $r$ to $r_D$. Hence, we can rewrite these terms as follows:

$$\begin{aligned} u_{r \to r^+} \cdot \ldots \cdot u_{r_D \to r_D^+} \cdot v(r_D^+) - u_{r \to r^-} \cdot \ldots \cdot u_{r_D \to r_D^-} \cdot v(r_D^-) \\ = u_{r \to r^+} \cdot \ldots \cdot u_{r_D \to r_D^+} \cdot \left(v(r_D^+) - v(r_D^-)\right), \end{aligned} \tag{6.14}$$

where we use that $u_{r_D \to r_D^+} = u_{r_D \to r_D^-} = 1$ for outgoing arcs of unbound decision nodes. Note that for all valid paths from the root to nodes labelled with $D$, we find at least one such term in the expanded expression for $\partial f(D, \sigma' \setminus \{d\})/\partial D$. Hence, for a particular node $r_D$, we can group all terms together, obtaining $\pi(r_D) \cdot \left(v(r_D^+) - v(r_D^-)\right)$. Summing over all particular nodes $r_D$ yields Equation 6.11. □

We use the observation above to create an $O(m+n)$ algorithm for calculating all derivatives in two stages:

1. A top-down pass over the complete OBDD for calculating all path weights.
2. A bottom-up pass for calculating the values for all nodes in the complete OBDD, calculating the derivatives for each variable in the process.

The *top-down* pass operates as follows. We initialise the path weight $\pi(r)$ of each internal node with 0, and the path weights of the roots are initialised with the utilities of the corresponding queries. We update the path weight of its children $r^+$ and $r^-$ as follows if $r$ is labelled with a decision variable $d$:

$$\begin{aligned} \pi\left(r^+\right) &:= \pi\left(r^+\right) + \pi(r) \text{ if } D \text{ is unbound or } \textit{true}; \\ \pi\left(r^-\right) &:= \pi\left(r^-\right) + \pi(r) \text{ if } D \text{ is } \textit{false}; \end{aligned} \tag{6.15}$$

If $r$ is labelled with a stochastic variable with weight $w$, we assign $\pi\left(r^+\right) + w \cdot \pi(r)$ to $\pi\left(r^+\right)$ and $\pi\left(r^-\right) + (1-w) \cdot \pi(r)$ to $\pi\left(r^-\right)$.

We compute the node values in a *bottom-up* pass, using Equation 2.11 with $w(r) = 0$ if $r$ corresponds to a decision variable that is *false*, and $w(r) = 1$ otherwise.

During this bottom-up pass, we can recompute the derivatives for all decision variables that are still unbound using Equation 6.11, and evaluate Equation 6.8 for each of those to see if we can remove *false* from their domain.

Clearly, the overall calculation can be completed in time $O(n+m)$.

### 6.4.3 A partial-sweep SCMD propagator

We now explore whether we can further reduce the empirical running time of the algorithm above, by avoiding the unnecessary traversal of parts of the OBDD. The following observations allow for potentially more efficient propagation:

**O1** As noted before, the expression for the path weight of an OBDD node labelled with variable $X_m$ (Equation 6.9) only contains variables $X_i < X_m$. We conclude that fixing a decision variable $D$ can only affect the *path weights* of nodes *below* the nodes labelled with that variable $D$.

**O2** Path weights below unbound decision nodes are not changed when we fix an unbound decision node to *true*. Therefore, our propagator only needs to update path weights if we fix a decision variable to *false*.

**O3** Similarly, fixing a variable can only affect the *scores* of the nodes labelled with that variable, and of those *above* them in the OBDD. Again, only fixing a variable to *false* requires the propagator to update scores.

**O4** We do not need to maintain the scores for any of the ancestors of the decision nodes that are closest to the root of the OBDD. For each of these ancestors $r$, it holds that there is no path from the root to $r$ that passes through a decision node. Therefore, we will never need to calculate the derivative for any variable in that part of the diagram.

**O5** Similarly, we do not need to maintain path weights for the descendants of the decision nodes closest to the leaves. For each of these descendants, it holds that there is not path from it to one of the leaves that passes through a decision node. Therefore, we will never need to calculate the derivative for any variable in that part of the diagram.

It can be shown that by only maintaining the part of the OBDD between two *borders* of unbound decision variables (the *active* part of the OBDD), one can calculate the derivatives exactly, as well as calculate the score of the solution.

**O6** Some parts of the OBDD will no longer be connected to the root as a consequence of partial assignments. We thus do not need to update those parts of the OBDD.

**O7** We can exploit partial derivatives as well as **O4** and **O5** in branching heuristics to guide the search. For example: if we always branch on the variable with the largest derivative, we are likely to find failing partial strategies quickly. Alternatively, by branching on the highest or lowest decision variable (applying **O4** and **O5**, respectively), we reduce the size of the active part of the OBDD.

We improve the *full-sweep OBDD propagation algorithm* by addressing these observations. Here, we give a short overview of how we do so; we refer the reader to Chapter A for the pseudocode of the resulting partial-sweep algorithm.

**O1** to **O3** are addressed by using priority queues; we initialise and update them such that we start traversing the OBDD downwards (upwards) at the places where path weights (scores) may change due to decision variable assignments.

In our implementation of the partial-sweep algorithm, we maintain for each OBDD node $r$ three counters, addressing **O4** to **O6**. Maintaining these counters requires two extra passes through part of the OBDD each time the propagator is called. However, they allow us to traverse an ever-decreasing part of the OBDD in each pass.

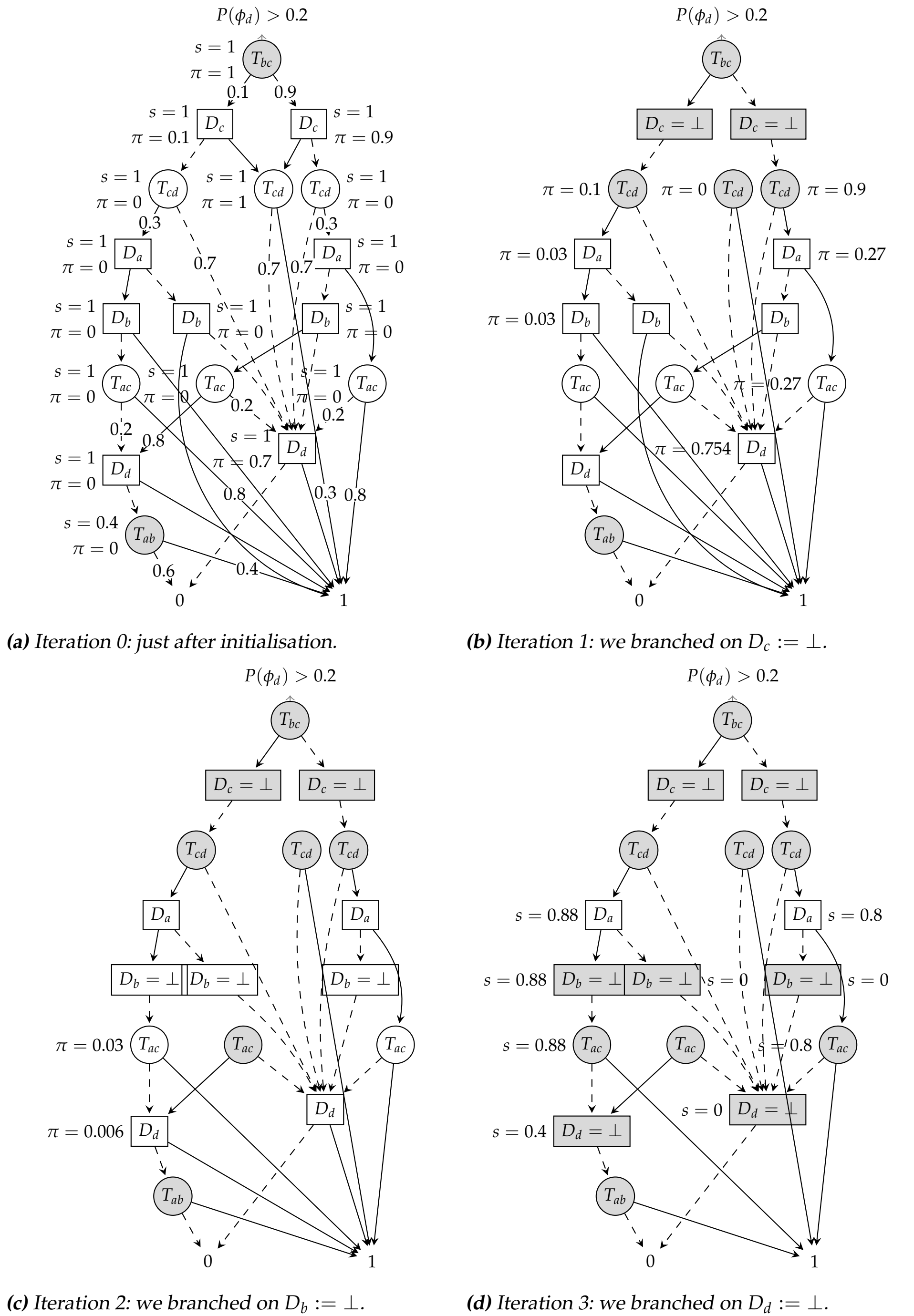
We call the first counter `FreeIn[r]`. It indicates the number of parents $r'$ of $r$ for which there is at least one valid path from an unbound decision node above $r'$ to $r'$. If `FreeIn[r]`=0, we need not update scores of nodes above $r$ if the score of $r$ changes (**O4**).

The second indicates the number of children $r'$ for which there is at least one valid path from $r'$ to an unbound decision node below $r'$; we call this counter `FreeOut[r]`. If its value is 0, any changes in $r$'s path weight need not be propagated down from $r$, because of **O5**.

Because of **O6**, we use a third counter, which we call `Reachable[r]`.

It counts the number of parents of $r$ through which there is a valid path from the root to $r$, thus counting through how many of its parents $r$ is reachable from the root. If there is no valid path from the root to $r$, $r$'s path weight is 0, and changes in its score need not be propagated. Note that we need the `Reachable[r]` counter despite the fact that we have the `FreeIn[r]` counter, because it can happen that a part of the OBDD becomes disconnected from the root while there are still free decision nodes in that part. If those decision nodes are ancestors of $r$, we would keep updating their scores if we only rely on the `FreeIn[r]` counter to stop that upwards traversal through the OBDD for that part. Note that for a multi-rooted OBDD, `Reachable[r]` counts the number of parents of $r$ through which $r$ is reachable from *at least one* of the roots.

Note that, as observed in **O6**, some decision nodes labelled with free decision variables may become disconnected from the root. Consequently, we do not use their scores to compute the derivative in Equation 6.11. If the OBDD representation of the probability distribution is *not* locally monotonic (Definition 6.3.2), it may happen that the contributions of the active nodes cause us to compute a negative partial derivative. Consequently, the algorithm is no longer be able to guarantee GAC, and may even compute a wrong score for the optimal solution.




***(a)** Iteration 0: just after initialisation.*

***(b)** Iteration 1: we branched on $D_c := \bot$.*

***(c)** Iteration 2: we branched on $D_b := \bot$.*

***(d)** Iteration 3: we branched on $D_d := \bot$.*

***Figure 6.2:** Illustration of the partial-sweep propagator on an OBDD representation of Equation 2.10.*

**Example 6.4.1** (Partial-sweep propagation). *Figure 6.2 shows an example of an execution of the partial-sweep algorithm on an OBDD representation of Equation* 2.10. *When unbound decision nodes become fixed, we remove one of their outgoing arcs to indicate their truth assignment. This may cause nodes to become no longer reachable from the root. Nodes that are inactive because of this, or because they are not on a path from one unbound decision node to another, are coloured grey. Note that there is no need to update the scores and path weights of nodes that have been greyed out, because of* ***O4*** *to* ***O6****. In each iteration of the algorithm, we use the partial derivatives to check if the upper bound on the score in the root of the diagram is still high enough to satisfy the constraint on the probability. Next to each node, we indicate its current score s and current path weight* $\pi$*. We only show the scores and path weights that change in an iteration. The* `Reachable`, `FreeIn` *and* `FreeOut` *counters are not shown in the figure. Suppose we have to find a strategy* $\sigma$*, such that* $P(\phi_d|_\sigma) > 0.2$ *holds.*

*Figure 6.2a shows the state of this OBDD just after initialisation. The current partial strategy is* $\sigma' = \varnothing$*, since no assignments to decision variables have been made. The partial derivatives are:* $\partial f/\partial D_a = 0$*,* $\partial f/\partial D_b = 0$*,* $\partial f/\partial D_c = 0$ *and* $\partial f/\partial D_d = 0.7$*. Since Equation* 6.7 *holds for all derivatives, we cannot fix any decision variables to* true*. The upper bound on the score of the diagram is* $s(root) = 1$*.*

*Suppose we now branch on* $D_c := \bot$ *in Figure 6.2b. Because there are no active nodes above the nodes labelled with* $D_c$*, no scores will change in this iteration. However, there are active nodes below the nodes labelled with* $D_c$*, causing some of the path weights to change. Note that the middle node labelled with* $T_{cd}$ *becomes unreachable. While its* `Reachable` *counter was* 2 *in Figure 6.2a, now it equals 0. Similarly, the* `FreeIn` *counters of all nodes labelled with* $T_{cd}$ *or* $D_c$ *become 0.*

*We now update the upper bound on the score of the diagram to* $P\left(\phi_e|_{\{D_c=\bot\}}\right) = P\left(\phi_e|_\varnothing\right) - \partial f/\partial D_c = 1 - 0 = 1$*, and compute new partial derivatives:* $\partial f/\partial D_a = 0$*,* $\partial f/\partial D_b = 0$ *and* $\partial f/\partial D_d = 0.754$*. Again, this is not enough to infer that a specific decision variable must be fixed to* true*.*

*In Figure 6.2c we branch on* $D_b := \bot$ *next. Note that the nodes labelled with* $D_b$ *remain active, as they are on paths from unbound decision nodes to other unbound decision nodes, and therefore, their* `FreeIn` *and* `FreeOut` *counters remain larger than 0. Since the scores of the nodes labelled with* $D_b$ *happen to not change in this case, we do not need to update the scores of the active nodes above them in the OBDD. However, we do need to update the path weight of one of the nodes below them. We can update the upper bound on the score of the diagram to* $P\left(\phi_d|_{\{D_b=D_c=\bot\}}\right) = P\left(\phi_e|_{\{D_c=\bot\}}\right) - \partial f/\partial D_b = 1 - 0 = 1$*. We compute the new partial derivatives:* $\partial f/\partial D_a = 0$ *and* $\partial f/\partial D_d = 0.7576$*. Again, this does not give us reason to fix any remaining decision variables to* true*.*

*Next, we branch on* $D_d := \bot$ *(see Figure 6.2d). There are no active nodes below those*

*nodes labelled with $D_d$ (their* `FreeOut` *counters equal 0), so no path weights are updated in this iteration. However, many scores do change. The upper bound on the score for the root of the diagram also changes:* $P\left(\phi_d|_{\{D_b=D_c=D_d=\bot\}}\right) = P\left(\phi_e|_{\{D_b=D_c=\bot\}}\right) - \partial f/\partial D_d = 1 - 0.7576 = 0.2424$.

*The last remaining partial derivative is:* $\partial f/\partial D_a = 0.2424$. *Now we know that this decision variable must be fixed to* true, *because* $P\left(\phi_e|_{\{D_b=D_c=D_d=\bot\}}\right) - \partial f/\partial D_a = (0.2424 - 0.2424) < 0.2$. *We therefore fix* $D_a = \top$ *and conclude that* $\sigma = \{D_a = \top, D_b = D_c = D_d = \bot\}$ *is a solution to the constraint* $P(\phi_d|_\sigma) > 0.2$, *and one with value* $P(\phi_d|_\sigma) = 0.2424$.

Note that in the example above we fix only one decision variable per iteration. Our implementation also allows multiple decision variables to be fixed at the same time, for example by another constraint, such as a linear constraint on the cardinality of the solution, as would be the case in Examples 4.2.1 and 4.2.2.

Finally, we address **O7** by implementing different *variable branching heuristics*: *Top*, which always branches on the unbound variable highest in the OBDD, and its counterpart, *Bottom*. Each can be combined with a *value branching heuristics*: either branch first on value 0, or on value 1. These heuristic are static during the search and depend on the variable order underlying the OBDD. We also implement two regret-based [27] branching heuristics that use the calculated derivatives: *Derivative-1* and *Derivative-0*. The former (latter) selects the unbound decision variable with the largest (smallest) absolute derivative and first branches on 1 (0). These heuristics are dynamically computed during the search, but do not present much overhead, since we need to compute the derivatives anyway.

Note that the space complexity of this approach is only $O(|OBDD| \cdot \tau)$, where $\tau$ is the depth of the search tree. This is less than that of the GAC-guaranteeing decomposition method from Section 6.2.

**Relation to cost-*multi-valued decision diagram (MDD)* propagators.** Our partial-sweep propagation algorithm bears some resemblance to cost-MDDs propagation algorithms in general [55, 69], and a recently proposed optimisation to such an algorithm in particular [142]. Both algorithms have a notion of "up" and "down" scores that they update and use for maintaining arc consistency. Both algorithms' implementations avoid unnecessary work by being smart about which nodes really need their scores updated. However, there are some important differences. Cost-MDDs are used to encode constraints, while we use DDs to encode probability distributions *on which we formulate a constraint*. In cost-MDDs, each path from the root to the "*true*" leaf corresponds to a valid solution to the constraint. However, in this dissertation, a valid solution consists of vari-

able assignments such that a weighted sum computed over several paths in the OBDD exceeds a certain threshold value. This is reflected in the scores that are being maintained for each node. In cost-MDDs propagators, these scores are sums over single paths, whereas in our propagator, these scores are weighted sums over multiple paths. A second difference is that in our algorithm, a node that becomes inactive (such that its scores are no longer updated) remains inactive until it might be reactivated due to backtracking, and due to backtracking only. In cost-MDDs propagators, on the other hand, as the branch-and-bound algorithm traverses deeper into the search tree, the values of a node may not be updated in one iteration, and then be updated again in the next. The final difference is in how nodes are selected for having their values updated. In the cost-MDD propagator, node values are updated if they may change due to arc removal. In our propagator there are cases in which we do not update node values, even though they change, because we do not need their values to compute the partial derivatives.

### 6.4.4 A global constraint SCPMD solving pipeline

In the previous two sections we described global SCMD solving algorithms. Recalling Section 5.2.3, we now give a brief summary of how these algorithms fit into a pipeline for solving SCPs.

Figure 6.3 shows this pipeline. The first three steps are exactly the same as the pipeline described in Section 5.2.3, except that we have the extra requirement on the input problems that the probability distributions involved in the stochastic constraint or optimisation criterion, must be locally monotonic (Definition 6.3.2):

**Step 1:** Model the problem using a probabilistic network and (stochastic) constraint(s) or optimisation criterion that involves a monotonic probability distribution.

**Step 2:** Model the problem using SC-ProbLog, without using negation.

**Step 3:** Ground the program for the queries present in the optimisation criterion of the SCP into a set of propositional formulae $\Phi$.

**Step 4:** Compile a multi-rooted OBDD, such that each root encodes the conditional success probability $P(\phi|_\sigma)$ of one of the queries $\phi \in \Phi$.

The pipeline differs from our earlier one in the next steps:

**Step 5:** Create a global stochastic constraint or stochastic optimisation criterion based on the OBDD encoding of the probability distribution, using either the full-sweep or partial-sweep implementation of the SCMD propagator.

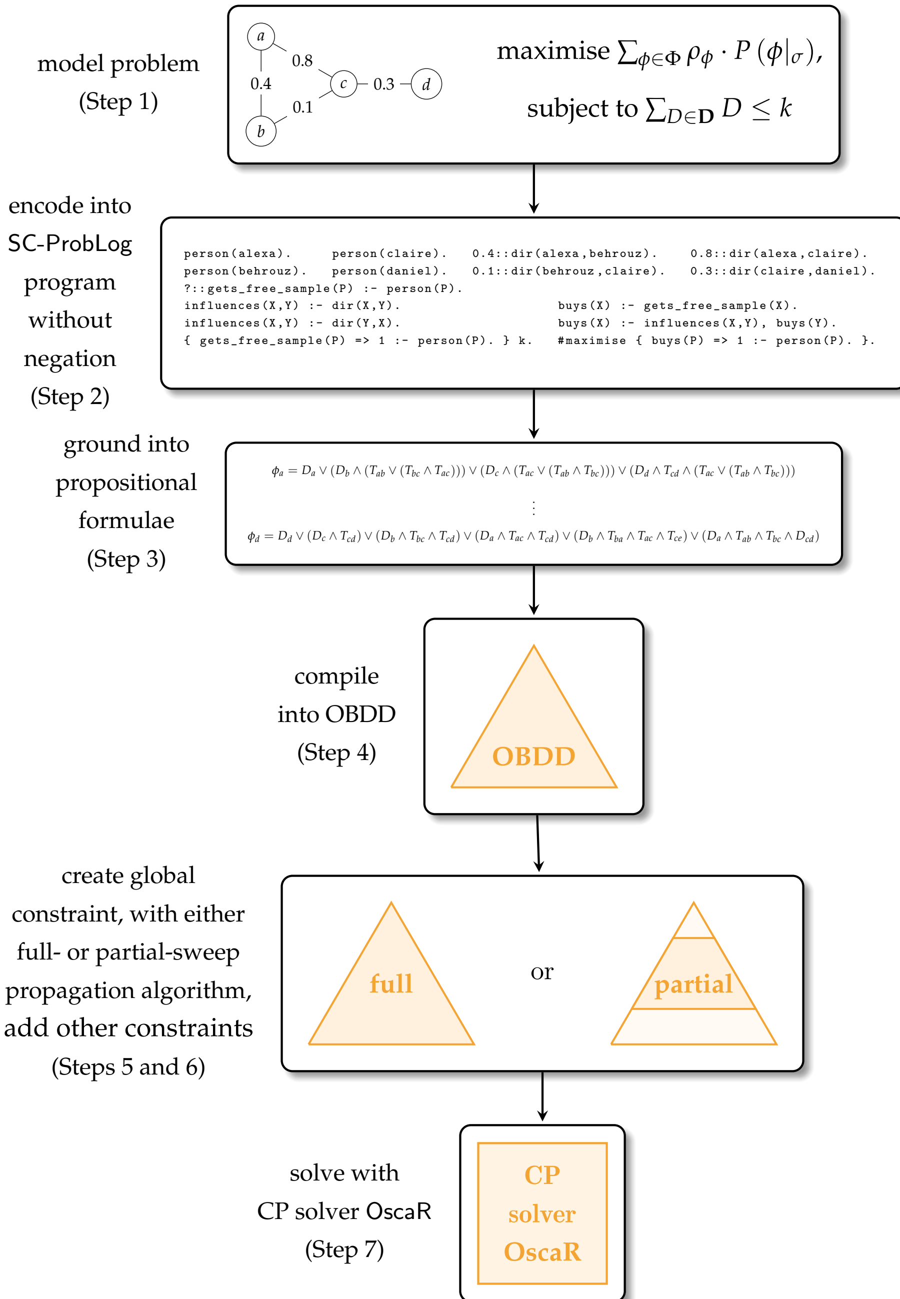



***Figure 6.3:*** *Overview of the global constraint SCP solving pipeline we propose in this chapter.*

**Step 6:** Add any other constraints and optionally an optimisation criterion to the CP model.

**Step 7:** Solve using a CP solver.

## 6.5 Experimental evaluation

We experimentally evaluate the performance of CP-based and MIP-based OBDD decomposition methods (described in Sections 5.2.1 and 6.2), as well as the full-sweep and partial-sweep global SCMD propagators on OBDDs (described in Sections 6.4.2 and 6.4.3).

The remainder of this section is organised as follows. First, we formulate our research questions. We then provide details on experimental setup, hardware and software we use in Section 6.5.2, as well as an overview of the different pipelines evaluated. Finally, we report and analyse the results we obtained in our experiments, answering our questions in Section 6.5.3.

### 6.5.1 Research questions

In the work presented in this chapter, we leverage techniques from knowledge compilation (with a focus on OBDDs) in combination with readily available CP and MIP solvers for efficient SCP solving. The experiments in this section are designed to evaluate the efficiency of decomposition methods and global constraint methods that use these elements. Specifically, we aim to answer the following questions:

**Q1** How do solving times depend on the CP encoding of the constraint (decomposed versus our new global constraint)?

**Q2** How do branching heuristics (Section 6.4.3) affect solving times for the global constraint?

**Q3** How do solving times for the global SCMD constraint compare to those of a decomposed constraint solved with a MIP solver?

**Q4** How do the performances of decomposed and global approaches depend on OBDD size?

**Q5** How effective is the partial-sweep propagation algorithm compared to the full-sweep algorithm in practice?

**Q6** How does our propagator perform in combination with other constraints?

### 6.5.2 Experimental setup

The implementations of our propagation algorithms are available at `github.com/latower/SCMD-solving`.

#### Software and hardware

For modelling the probability distributions, we used the SC-ProbLog language proposed in Section 4.3, which is based on ProbLog 2.1 [59] and DT-ProbLog [178], running in Python 3.6.9.[1] We use the Cython binding of the dd 0.5.4 library to CUDD 3.0.0 [168] for OBDD compilation, and use its implementation of the Sifting algorithm [156] for dynamic minimisation.[2]

We implemented the MIP-based decomposition methods by building and solving MIP models with Gurobi 9.0.0, because it is freely available to academics and provides a convenient modelling interface through gurobipy.[3] The CP-based decomposition was implemented in Gecode 6.0.1, because it is a well-known, well-performing, open source solver that is used by industry.[4]

We implemented the global OBDD propagators proposed in Sections 6.4.2 and 6.4.3 in the Scala 2.12 library OscaR 4.0.0 [132]. This library contains a state-of-the-art implementation of the CoverSize constraint [164], which we needed to answer **Q6**.[5] Since OscaR does not support floating point variables, we could not implement the decomposition methods in OscaR.

Our experiments in Section 6.5.3 were run on two different machines, for reasons of availability. The first, which we refer to as PASCALINE, is equipped with 24GB of RAM and eight Intel Xeon E5540 CPUs, each with four cores and 8192 KB of cache, running at 2.53 GHz, under CentOS Linux 7.4.1708. The second, GRACE, is a cluster with 32 nodes, each equipped with 94GB of RAM and two Intel Xeon E5-2683 CPUs with 16 cores, a cache size of 40MB, running at 2.10 GHz under CentOS Linux 7.7.1908. Unless indicated otherwise, all experiments in Section 6.5.3 were run on PASCALINE. Note that whenever running times need to be compared directly, they were obtained from experiments that ran on the same machine. Running times were measured in wall clock time, using the solver's reports on their running times, which exclude time for reading in or constructing the models and thus measure solving time alone.

[1] Available at `github.com/ML-KULeuven/problog/tree/sc-problog`.
[2] Available at `pypi.org/project/dd`.
[3] Available at www.gurobi.com.
[4] Available at www.gecode.org.
[5] Available at `sites.uclouvain.be/cp4dm/fim`.

**Overview of solving methods**

We briefly outline the different solving methods evaluated in this section. As described in Sections 5.2.3 and 6.4.4, a full SCPMD solving pipeline starts with modelling the problem in SC-ProbLog (as demonstrated in Section 4.3) and grounding the resulting logical program into propositional formulae $\phi_i$ on Boolean decision variables and Boolean stochastic variables (see examples in Section 4.2). We then use knowledge compilation to compile these formulae into OBDDs and impose the stochastic constraint of Equation 1.1 on the probability distributions encoded by those OBDDs. We then either decompose the resulting constraint on the OBDDs into a set of smaller, local constraints, or keep it as a global constraint. We then solve this model with a CP or MIP solver. In the experiments in this section we *only* evaluate the solving part of the pipelines, which starts after the model has been loaded into the solver. We evaluate the entire pipeline in the experiments presented in Chapter 6.

**CP-based decomposition.** In Sections 5.2.1 and 5.2.2, we described how constraints on probability distributions modelled by OBDDs and SDDs can be decomposed into linear programs. Using this decomposition, we proposed a method that uses Gecode to solve a CP encoding of the stochastic constraint on a multi-rooted OBDD that does not guarantee GAC (Sections 3.3 and 5.2.1). We therefore refer to the solving step in this pipeline as *no-GAC CP decomposition*. In Section 6.2, we briefly discussed how we can turn this CP encoding into one that does guarantee GAC. We also solve the resulting CP programs from this encoding with Gecode, and refer to this pipeline as *GAC CP decomposition*.

**MIP-based decomposition.** Since MIP solvers have been shown to be very effective in solving linear programs, we also evaluate an OBDD variant of the MIP-based pipeline described in Chapter 5. The *OBDD-to-MIP* pipeline converts the propositional formulae $\phi_i$ into a multi-rooted OBDD, and converts this OBDD, and the stochastic constraint imposed on it, into a linear program that is then solved using Gurobi.

**Global SCMD propagation.** Finally, we evaluate the two variants of the new global SCMD propagator on probability distributions represented by OBDDs, implemented in OscaR: *full-sweep* (Section 6.4.2) and *partial-sweep* (Section 6.4.3).

***Table 6.1:*** *Characteristics of our set of 52 problem instances, including their range of size of the set of interest $|\Phi|$, number of stochastic variables $|\mathbf{T}|$, number of decision variables $|\mathbf{D}|$ and the number of test instances of each type.*

| name | problem type | $\lvert\Phi\rvert$ | $\lvert\mathbf{T}\rvert$ | $\lvert\mathbf{D}\rvert$ | # instances |
|---|---|---|---|---|---|
| **spine** | *sparsification* | 13–23 | 33–60 | 33–60 | 3 |
| **hep-th** | *spread of influence* | 20–33 | 51–90 | 20–33 | 2 |
| **facebook** | *spread of influence* | 10–18 | 40–98 | 20–30 | 11 |
| **high-voltage** | *power grid reliability* | 2–20 | 32–154 | 15–45 | 36 |

### Parameter settings

While, in the next chapter, we will do a thorough analysis of the influence of parameter settings on the performance of our methods, in these first experiments, we use the default settings for all software, unless indicated otherwise. In the experiments to answer **Q1**, we constrain both CP solvers to branch on the variables in lexicographical order, branching first on *false* and then on *true*. In doing so, we fix the branching order in an attempt to take the influence of branching heuristics out of the equation, and thus to compare only the speed and effect of propagation. For the other experiments, the global SCMD propagators use the branching heuristic *Derivative-1* (Section 6.4.3), because it seems to outperform the other branching heuristics, as is shown in Table 6.3.

### Problem instances

For our experiments we consider a total of 52 instances from the **spine** [133], **hepth** [131], **facebook** [180] and **powergrid** [183] data sets described in Section 4.5. For all these instances, we choose a problem setting of **Variant 1** (see Section 4.5). We summarise some characteristics of these instances in Table 6.1.

For presentation purposes we selected a representative subset of ten instances from this set, for which we will show our results in this work. We provide some characteristics of the instances in this subset in Table 6.2.

For each problem instance, we select a constraint threshold in the form of an upper bound on the cardinality of the solution $k$. Specifically, we run each example for nine values of $k$, based on the number of decision variables in the problem instance. For sparsification problems, $k$ represents an upper bound on the size of the network that we extract. For spread-of-influence problems, $k$ represents an upper bound on the number of people to whom we can give a free sample of the product. Finally, for power grid reliability problems, we make the simplifying as-

***Table 6.2:** Some characteristics of the test instances we use in Section 6.5.3. In particular: what entities the decision variables are associated with, the size of the set of interest $|\Phi|$, the number of stochastic variables $|\mathbf{T}|$ and the number of decision variables $|\mathbf{D}|$, the OBDD size without minimisation ($|OBDD_{nm}|$) and with dynamic minimisation ($|OBDD_{dm}|$) [156] during the compilation, the OBDD compilation time without minimisation $t_{nm}$, the difference in compilation times $\Delta t$ for these two compilation methods (compilation with dynamic minimisation always takes longer than compilation without minimisation).*

| instance | $\|\Phi\|$ | $\|\mathbf{T}\|$ | $\|\mathbf{D}\|$ | $\|\text{OBDD}_{nm}\|$ | $\|\text{OBDD}_{dm}\|$ | $t_{nm}$ [s] | $\Delta t$ [s] |
|---|---|---|---|---|---|---|---|
| *sparsification* | | | | | | | |
| **spine16** | 23 | 33 | 33 | 80 | 80 | 0.09 | 0.01 |
| **spine27a** | 13 | 60 | 60 | 1 898 | 266 | 0.08 | 0.03 |
| **spine27b** | 13 | 55 | 55 | 9 350 | 476 | 0.08 | 1.13 |
| *spread of influence* | | | | | | | |
| **hep-th47** | 20 | 51 | 20 | 10 815 | 3 658 | 0.12 | 0.59 |
| **hep-th5** | 33 | 90 | 33 | 14 555 | 8 865 | 0.43 | 13.25 |
| **facebook12** | 12 | 61 | 23 | 7 836 | 794 | 0.09 | 0.07 |
| **facebook25** | 25 | 72 | 25 | 6 981 | 2 198 | 0.10 | 0.22 |
| *power grid reliability* | | | | | | | |
| **croatia** | 6 | 66 | 21 | 4 873 | 429 | 0.20 | 0.13 |
| **illinois** | 20 | 96 | 32 | 68 019 | 3 040 | 0.37 | 0.60 |
| **russia** | 16 | 94 | 34 | 1 616 | 947 | 0.42 | 0.55 |

sumption that the cost of reinforcing power lines is uniform, such that we can replace the budget $\beta$ by an upper bound on the number of power lines we can reinforce, $k$. We do this to avoid overcomplicating the experiments.

For our experiments on the *frequent itemset mining (FIM)* problem setting, in which we aim to detect top fake news distributors, we used communities in the **facebook** dataset. We generated 25 OBDDs, which we combined with different minimum expectation thresholds $\theta$ and different minimum support thresholds $\kappa$ to create FIM problem instances. The problems have sets of interest of size 50–65 and the same numbers of decision variables. The numbers of stochastic variables range from 151–225, and the databases contain 33–52 transactions. Finally, the OBDD sizes range from roughly 20 thousand to 2.5 million nodes.

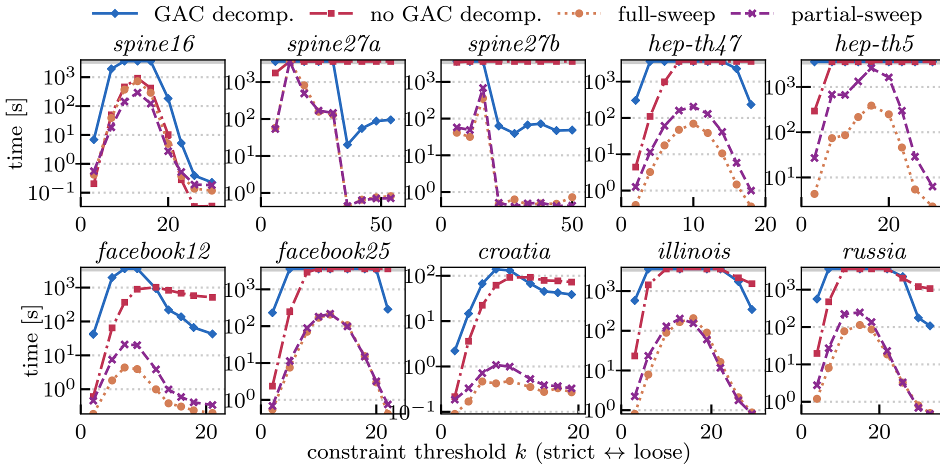



***Figure 6.4:*** *Solving times of CP-decomposition methods and global SCMD methods. Cut-off time is 3 600 s (1 hour). Vertical axes are log scale.*

### 6.5.3 Results

We study how the decomposition methods (Sections 5.2.1 and 5.2.2) compare to the global SCMD propagators (Sections 6.4.2 and 6.4.3) in terms of solving time, which we measure by using the wall-clock time reported by the different solvers as the time actually spent on solving (and not on I/O). We aggregate some of our results by computing the *penalised average runtime with penalty factor 10 (PAR10)* values of solving times.

**Comparison of CP solvers.** We address **Q1** by comparing the solver search times of the implementations of the full-sweep (Section 6.4.2) and partial-sweep (Section 6.4.3 versions of our propagator with two decomposed approaches in Gecode: the GAC CP composition method and the no GAC CP decomposition method (Sections 5.2.1 and 5.2.2). We keep the branching order for the search process fixed to a lexicographical one, branching first on *false* and then on *true*. This allows us to directly compare the propagation strength and speed on these CP methods, because the ones that guarantee GAC have the same search trees. The constraint threshold $k$ indicates the maximum allowed cardinality of the solution: from small (strict) to large (loose). We run these propagators on OBDDs that are obtained using dynamic minimisation. Figure 6.4 shows that the global SCMD propagators outperform both decomposition methods on our set of test instances. While the full-sweep version of the SCMD propagator outperforms the partial-sweep version, this difference is less pronounced.

***Table 6.3:*** *PAR10 values in seconds for six branching heuristics used by the full-sweep propagation algorithm on 52 test instances. Cutoff time is 3 600 s.*

| *Top-0* | *Top-1* | *Bottom-0* | *Bottom-1* | *Derivative-0* | *Derivative-1* |
|---|---|---|---|---|---|
| 1 502 | 1 575 | 1 526 | 1 385 | 2 412 | 27 |

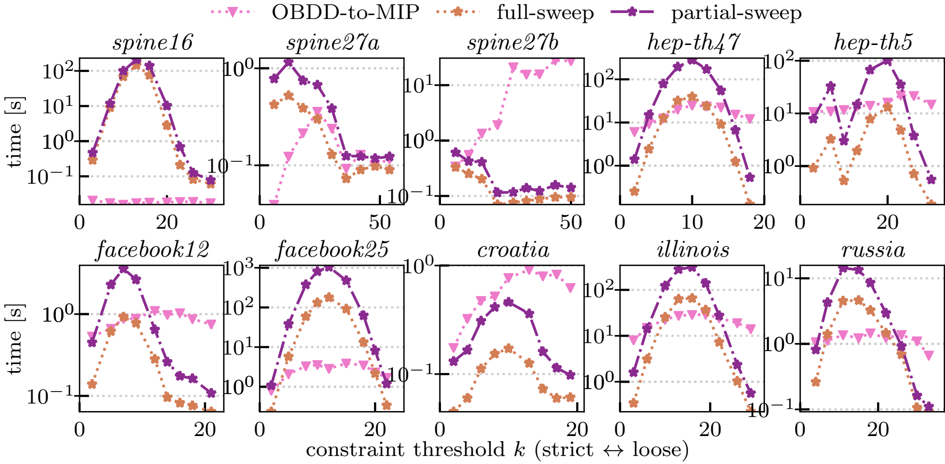



***Figure 6.5:*** *Solving times of MIP-based OBDD decomposition method, compared to the full-sweep and partial-sweep methods. Cutoff time is* 3 600 s *(*1 *hour). Vertical axes are log scale.*

**Branching heuristics.** We answer **Q2** by evaluating the performance of the six branching heuristics described in Section 6.4.3. We ran the full-sweep and partial-sweep propagation algorithm on our set of 52 instances described in Table 6.1, using an upper bound of $k = \lfloor |\mathbf{D}|/2 \rceil$ on the cardinality of the solution, where $|\mathbf{D}|$ denotes the number of decision variables of the given instance. We repeated this for the six branching heuristics from Section 6.4.3, using a cutoff time of 3 600 s, and compute the PAR10. We present the results in Table 6.3. Clearly, *Derivative-1* seems to be the most efficient branching heuristic for the full-sweep propagator on our set of test instances.

**Comparison of global CP and decomposed MIP encoding.** Figure 6.5 compares the performance of the full-sweep and partial-sweep OBDD propagators to that of the OBDD decomposition method using Gurobi for solving the problem (OBDD-to-MIP). For the global propagators, we have used branching heuristic *Derivative-1*. We observe that the global SCMD propagators perform comparably,

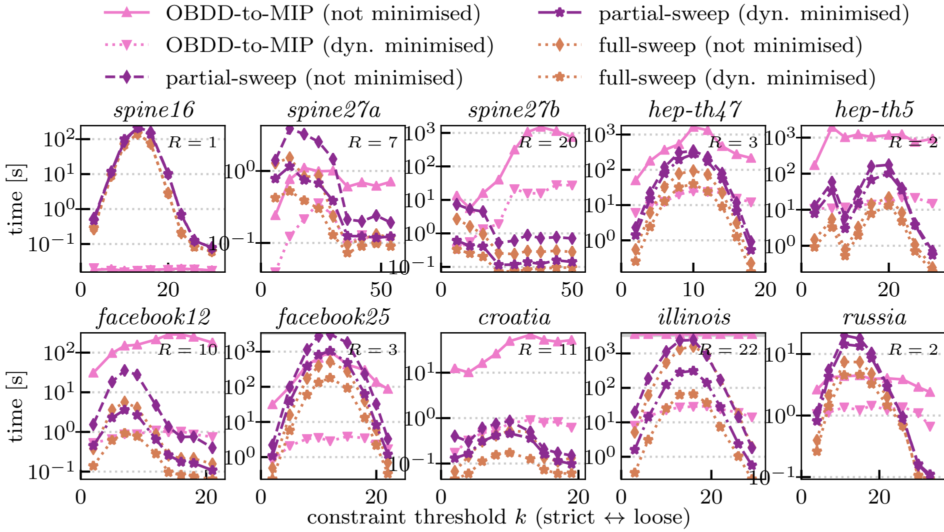



***Figure 6.6:*** *Solving times of OBDD-to-MIP and global SCMD propagators on OBDDs of different sizes: either obtained without minimisation, or by using dynamic minimisation during compilation. Cutoff time is 3 600 s (1 hour). Vertical axes are log scale. R indicates the size ratio* $|OBDD_{nm}|/|OBDD_{dm}|$.

and often complementarily, to the OBDD-to-MIP method, answering **Q3**.

**Scaling.** We address **Q4** in Figure 6.6, where we show how the full and partial-sweep SCMD propagators scale for OBDDs of different size, obtained by running an OBDD compiler with and without minimisation for the same set of problems. Note that the SCMD propagators have the same search tree regardless of the shape and size of the OBDD they operate on. We observe that the global SCMD propagators seem to scale much more favourably with OBDD size than the OBDD-to-MIP decomposition method. For example, on *facebook12*, the minimised OBDD is one order of magnitude smaller than the non-minimised OBDD. Both full-sweep and partial-sweep propagators seem to indeed scale linearly with that difference in size. However, the solving times for the OBDD-to-MIP decomposition method increase by over two orders of magnitude when the OBDD size increases by one order of magnitude.

**Full-sweep versus partial-sweep.** Recall that the full-sweep algorithm traverses the entire OBDD twice per iteration of the propagator, while the partial-sweep algorithm is designed to traverse only part of the OBDD in each propagation. This renders the partial-sweep algorithm potentially more efficient than

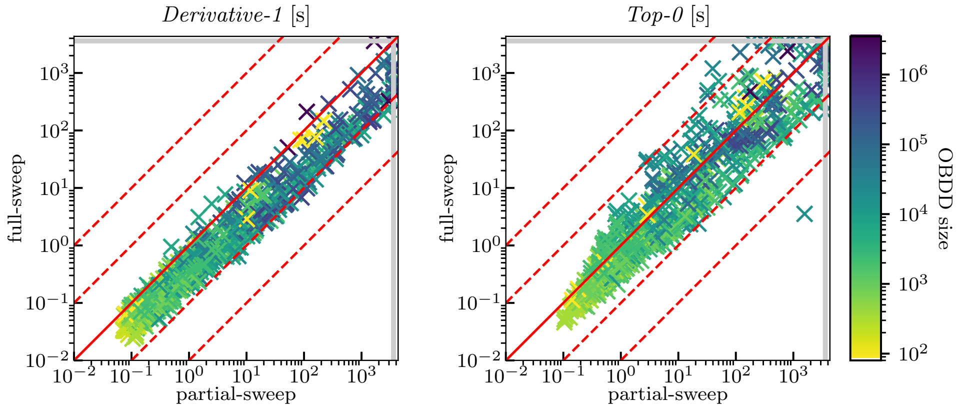



***Figure 6.7:*** *Solving times of the full-sweep and partial-sweep SCMD propagators on dynamically minimised and non-minimised OBDDs, for 52 instances (Table 6.1). We compare two branching heuristics:* Derivative-1 *and* Top-0. *Cutoff time is 3 600 s (1 hour).*

the full-sweep version, especially for branching strategies that work to reduce the active part of the OBDD. While this comes at the price of some overhead, we expect the overhead to become less important as OBDD size increases, since for larger OBDDs, the benefits of not traversing the entire OBDD become more pronounced.

To answer **Q5**, we therefore ran both propagators on the dynamically minimised and non-minimised OBDDs of all 52 test instances from Table 6.1. We ran each solver on each OBDD, using nine constraint thresholds $k$. We performed this experiment using two different branching heuristics: *Derivative-1* and *Top-0*. Figure 6.7 and Table 6.4 summarise our results.

Looking at the left plot in Figure 6.7, we observe that the full-sweep propagator tends to solve instances faster than partial-sweep when using the *Derivative-1* branching heuristic. This is also reflected in the results in Table 6.4, where we see that the PAR10 value for the partial-sweep propagator is 1.6 times that of the full-sweep propagator. However, when we look at the top 5% largest OBDDs only, these PAR10 values are more similar.

This effect is stronger when we use the *Top-0* branching heuristic. Recall that this heuristic attempts to reduce the size of the active part of the OBDD during the search, by always branching on the free decision variable that is highest in the OBDD. The right plot in Figure 6.7 shows that, when using the *Top-0* branching heuristic, the partial-sweep propagator outperforms the full-sweep algorithm on many instances. This is also reflected in the PAR10 values in Table 6.4: on the full set of OBDDs, the PAR10 value of partial-sweep is now only 1.2 times higher

***Table 6.4:*** *PAR10 values (cutoff time of 3 600 s) for full-sweep and partial-sweep SCMD propagators on dynamically minimised and not minimised OBDDs, for 52 instances (Table 6.1), and the top 5% largest OBDDs in this set. We ran them on nine values of threshold k per OBDD, and compare two branching heuristics:* Derivative-1 *and* Top-0*. We indicate in parentheses the total number of instances, and how many times out of the total number of instances a solver timed out. We also indicate the partial/full ratio of the PAR10 values.*

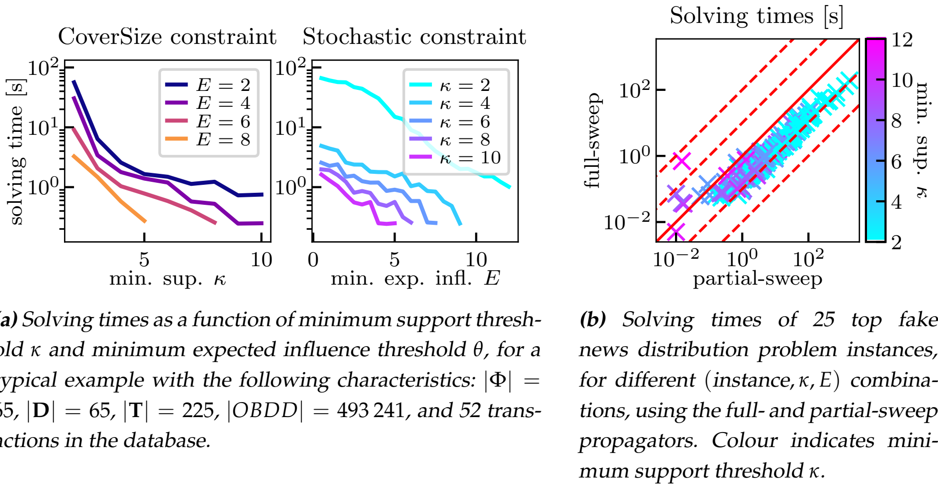
| | All OBDDs (936) | | Top 5% largest OBDDs (45) | |
|---|---|---|---|---|
| | *Derivative-1* | *Top-0* | *Derivative-1* | *Top-0* |
| full | 847 s (21) | 2 366 s (59) | 13 817 s (17) | 21 028 s (26) |
| partial | 1 373 s (33) | 2 470 s (61) | 16 389 s (20) | 19 585 s (24) |
| partial/full | 1.6 | 1.0 | 1.2 | 0.9 |

than that of full-sweep. Again, partial-sweep has a smaller PAR10 value than full-sweep on the largest 5% of OBDDs.

These results confirm that branching heuristics that aim to minimise the size of the active part of the OBDD can indeed give partial-sweep the edge over full-sweep for large OBDDs. *Derivative-1*, on the other hand, leads to smaller search trees. As the active part of OBDDs in this case does not get much smaller, the partial sweep algorithm entails an overhead compared to the full-sweep approach, and partial sweep does not offer substantial benefits. Nevertheless, even with a branching heuristic that aims to minimise the size of the search tree (*Derivative-1*), we see an indication that partial-sweep becomes competitive with full-sweep, as OBDD size increases.

**Interaction with other constraints.** We conclude our evaluation of the global constraint propagation algorithm with experiments on FIM instances, which we performed on GRACE. In all earlier experiments, we combined the stochastic constraint with a cardinality constraint. To answer **Q6**, in this experiment, we have evaluated the interaction of the global propagator with a CoverSize constraint. We note that, in contrast with the other experiments reported in this section, this is a constraint *solving* rather than a constraint *optimisation* setting. Note also that we *enumerate* all solutions to the constraint solving problem.

We looked at the solving time of the top fake news distributors problem instances for different minimum support thresholds $\kappa$ and minimum expected influence thresholds $\theta$. We present the results for a typical example problem instance in Figure 6.8a, which shows the running time for the full-sweep propagator with branching heuristic *Derivative-1* for different combinations of $\theta$ and $\kappa$.




*(a) Solving times as a function of minimum support threshold $\kappa$ and minimum expected influence threshold $\theta$, for a typical example with the following characteristics: $|\Phi| = 65$, $|\mathbf{D}| = 65$, $|\mathbf{T}| = 225$, $|OBDD| = 493\,241$, and 52 transactions in the database.*

*(b) Solving times of 25 top fake news distribution problem instances, for different $(\text{instance}, \kappa, E)$ combinations, using the full- and partial-sweep propagators. Colour indicates minimum support threshold $\kappa$.*

***Figure 6.8:*** *Experimental results on the top fake news distributors problem setting (Section 4.5).*

Lower values of $\theta$ and $\kappa$ correspond to looser constraints. As expected, we see that solving times decrease as the constraints become stricter.

In Figure 6.8b, we compare the solving times of the full-sweep and partial-sweep propagators (using branching heuristic *Derivative-1*) on the full set of 25 FIM problem instances, each combined with different ($\kappa$, $\theta$) combinations. The colour indicates the minimum support threshold $\kappa$. We observe that the full-sweep propagator outperforms the partial-sweep version on almost all instances. However, for large values of $\kappa$, we see that the partial-sweep propagator becomes more competitive with, and in some cases even faster than, the full-sweep propagator. We explain this by observing that for large values of $\kappa$, itemsets whose support meet the threshold will likely be small. In the CoverSize algorithm, this means that most decision variables will be fixed early in the search. Since in the partial-sweep algorithm, the size of the active part of the OBDD tends to decrease when decision variables are fixed, the active part likely shrinks dramatically early on in the search. The benefits of the reduction in size then start to outweigh the larger overhead.

## 6.6 Conclusion

In this chapter, we identified a problem with the decomposition approach to solving stochastic constraint (optimisation) problems (SCPs) as described in Chap-

ter 5: it does not guarantee generalised arc consistency (GAC) and may therefore traverse the search space inefficiently. Instead, we proposed a new method that guarantees GAC by design and is specifically suited for solving global stochastic constraints on monotonic distributions (SCMDs). It operates on OBDD encodings of probability distributions, leveraging the monotonicity of the underlying probability distributions.

We gave an extensive description of two implementations of this global SCMD constraint propagator, and showed that their incremental way of propagating results in linear time complexity. The benefit of the *partial* sweep implementation is that it does not need traverse the complete OBDD in all cases; however, additional data structures are required to make this possible; the *full* sweep propagator always considers the full OBDD, but incurs a smaller overhead in its passes through the OBDD.

We implemented both versions in the CP solver OscaR [132]. In an initial set of experiments on a set of 52 examples problems from four different domains, we demonstrated that our global SCMD propagation method is superior to a CP-decomposition method. However, when comparing the global SCMD approach in its two variations with the MIP-decomposition approach, we found that the approaches perform complementarily, with none of the approaches consistently outperforming the other. Small trends can however be observed.

Specifically, the global SCMD propagators scale better with the size of the OBDD than the MIP-decomposition method. For smaller OBDDs, the full-sweep implementation of the global SCMD propagator outperforms the partial-sweep version, while this is less pronounced for larger OBDDs. The branching heuristics in CP are important; a branching order that focuses on reducing the size of the active part of the OBDD, leads to more efficient propagation for the partial-sweep implementation, but also to larger search trees. Overall, the choice of parameter settings is important to obtain good performance for both the SCMD and MIP methods.

We also presented results that suggest that for larger OBDDs, the partial-sweep algorithm becomes competitive with and might even outperform the full-sweep algorithm. The bottleneck for creating larger OBDDs for our experiments was ProbLog's speed in grounding the probabilistic programs. Perhaps with different tools, we could create larger monotonic OBDDs. Recently, promising efforts have been made towards opening up that grounding bottleneck [175], which opens up a concrete avenue for future work on exploring the performances of the full-sweep and partial-sweep algorithms further.

In this dissertation in general and this chapter in particular, we have limited

our attention to applications in network analysis, because such problems are complex and have interesting monotonicity properties, and to applications that require maximisation of an expected value. It would be interesting to study how well these approaches work in other types of problems. For example, we believe our constraint propagation algorithm to be easily modifiable such that it can be applied to problems where we require a lower bound on an expected value and minimise the cardinality of the solution. We also expect our methods to find possible applications in the domains of FIM and in scheduling and vehicle routing problems. Here we can exploit the fact that it is easy to combine our constraint with other constraints in CP.

Naturally, some questions remain. While the theoretical asymptotic worst-case time complexity of the partial-sweep propagator is the same as that of the full-sweep propagator, in practice we find that the overhead of this propagator is large. Based on our experiments, the cost of the overhead does not outweigh the benefits of traversing a smaller part of the diagram, except for sufficiently large instances and branching orders that reduce the size of the active part of the diagrams. Even so, whether an alternative approach can be developed with a smaller overhead remains an open question.

A first step towards answering that question may be to take a careful look at the observations presented in Section 6.4.2. Notice that addressing the first three observations does not require the addition of much extra overhead; they could all be dealt with by using priority queues. An obvious next step may therefore be to implement a version of the SCMD propagator that is somewhat in between the full-sweep and the partial-sweep propagator. For example: one that only implements the optimisations implied by **O1** to **O3**, or a subset thereof.

Another direction of interest is to generalise the concept of monotonicity to SDDs, and to develop a corresponding propagation algorithm. The partial-sweep OBDD propagation algorithm that we presented in this chapter heavily relies on the fact that OBDD nodes split on variable values. In SDDs however, nodes split on how entire sub formulae, and thus on the values of sets of variables. Hence the generalisation of our partial-sweep propagation algorithm to SDDs is not trivial. Since SDDs can be made more succinct than OBDDs, we do think that this could be an interesting line of future research.

We implemented our SCMD propagation algorithm in OscaR, so it is available to any OscaR user. However, for the benefit of those unfamiliar with OscaR or those unwilling or unable to use OscaR, it would be good to implement versions of this propagation algorithm in other CP solvers as well. A key feature of OscaR is its use of reversible data structures, providing convenient and efficient

support for backtracking. It would be interesting to know if and how our SCMD propagation algorithms can be implemented efficiently in other CP systems.

A different question is whether we can develop stochastic constraint propagators that do *not* require that the probability distribution be monotonic. These constraints may either be more general, or also especially designed to work on probability distributions with specific properties. Another interesting line of possible future work is to ask if we can develop propagation algorithms that operate on SDDs rather than OBDDs. After all, SDDs can be more compact than OBDDs and thus maybe yield propagation algorithms that are more efficient in practice. We believe that the performance of the SCMD propagator, as presented in this chapter, should provide sufficient encouragement to a future researcher to explore the research directions outlined above.