Exact stochastic constraint optimisation with applications in network analysis

Latour A.L.D.; Babaki B.; Fokkinga D.; Anastacio M.I.A.; Hoos H.H.; Nijssen S.G.R.

Artificial Intelligence 304 (2022) 103650

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

Exact stochastic constraint optimisation with applications in network analysis

Anna L.D. Latour [a,*], Behrouz Babaki [b], Daniël Fokkinga [a], Marie Anastacio [a], Holger H. Hoos [a,c], Siegfried Nijssen [d,*]

[a] LIACS, Leiden University, P.O. Box 9512, 2300 RA Leiden, the Netherlands
[b] Polytechnique Montréal, Montréal (Québec), H3T 1J4, Canada
[c] University of British Columbia, Vancouver (British Columbia), V6T 1Z4, Canada
[d] ICTEAM, UCLouvain, Place Sainte-Barbe 2 bte L5.02.01, B-1348 Louvain-la-Neuve, Belgium

ARTICLE INFO

Article history:
Received 15 August 2020
Received in revised form 8 October 2021
Accepted 4 December 2021
Available online 10 December 2021

Keywords:
Constraint programming
Probabilistic inference
Stochastic constraints
Ordered binary decision diagrams
Monotonic probability distributions
Global constraints
Automated algorithm configuration
Probabilistic networks

ABSTRACT

We present an extensive study of methods for exactly solving stochastic constraint (optimisation) problems (SCPs) in network analysis. These problems are prevalent in science, governance and industry. The first method we study is generic and decomposes stochastic constraints into a multitude of smaller local constraints that are solved using a constraint programming (CP) or mixed-integer programming (MIP) solver. However, many SCPs are formulated on probability distributions with a monotonic property, meaning that adding a positive decision to a partial solution to the problem cannot cause a decrease in solution quality. The second method is specifically designed for solving global stochastic constraints on monotonic probability distributions (SCMDs) in CP. Both methods use knowledge compilation to obtain a decision diagram encoding of the relevant probability distributions, where we focus on ordered binary decision diagrams (OBDDs). We discuss theoretical advantages and disadvantages of these methods and evaluate them experimentally. We observed that global approaches to solving SCMDs outperform decomposition approaches from CP, and perform complementarily to MIP-based decomposition approaches, while scaling much more favourably with instance size. Both methods have many alternative design choices, as both knowledge compilation and constraint solvers are used in a single pipeline. To identify which configurations work best, we apply programming by optimisation. Specifically, we show how an automated algorithm configurator can be used to find optimised configurations of our pipeline. After configuration, our global SCMD solving pipeline outperforms its closest competitor (a MIP-based decomposition pipeline) on all test sets we considered by up to two orders of magnitude in terms of PAR10 scores.

1. Introduction

In business, governance, science as well as in our daily lives, we often have to solve problems that involve optimal decision making under constraints and uncertainty. Examples of these problems arise in planning and scheduling, but they

* Corresponding authors.
E-mail addresses: a.l.d.latour@liacs.leidenuniv.nl (A.L.D. Latour), behrouz.babaki@polymtl.ca (B. Babaki), d.b.fokkinga@umail.leidenuniv.nl (D. Fokkinga), m.i.a.anastacio@liacs.leidenuniv.nl (M. Anastacio), hh@liacs.nl (H.H. Hoos), siegfried.nijssen@uclouvain.be (S. Nijssen).

https://doi.org/10.1016/j.artint.2021.103650

also occur naturally in fields such as data mining and bioinformatics. Given the abundance of relational data in these areas, many problems also involve probabilistic network data.

Consider, for example, the following *spread of influence problem*, as studied in the data mining literature [26,41]. We are given a social network of people (vertices) that have stochastic influence relationships (edges). To promote a new product, we want to rely on word-of-mouth advertisement to turn acquaintances of people who buy our product into new customers. We start this process by 'infecting' a set of people in the network by giving them a free sample. The budget for this marketing campaign is limited (a constraint), so we can distribute at most k samples. Each person who is a customer can influence their friends to become a customer with a certain probability. Given the social network, which k people should we give free samples in order to maximise the expected number of eventual customers (our objective)? We note that adding a person to the set of people who receive a free product sample can never decrease the expected number of people that will eventually be infected by the viral campaign.

Another example is an optimisation variant of the *power transmission grid reliability problem* [29]. We are given a power transmission network of power lines, power producers, power consumers and transmission stations. Natural disasters, such as earthquakes and hurricanes, may cause power lines to break, and if too many such breaks occur, households lose power. We can reinforce power lines to make them less likely to break during a natural disaster, and we are given a budget b for these upgrades (a constraint). Which power lines do we reinforce such that we maximise the expected number of households that will still have power after a natural disaster (our objective), while not exceeding our budget? Note 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.

A third example is an enumeration problem, aimed at finding sets of members of a social network who are, in expectation, highly influential to certain communities in their distribution of fake news. Using again a probabilistic spread-of-influence framework similar to the one described above, we combine this framework with a *frequent itemset mining* (FIM) [1] approach and associated constraint. Here, we repeatedly solve a constraint satisfaction problem, to find all solutions to that problem, instead of solving an optimisation problem.

These problems are instances of a general class of problems, known as *stochastic constraint (optimisation) problems* (SCPs). SCPs have the following characteristics:

- They involve *decision* variables and *stochastic* (or *random*) variables.
- They involve reasoning over (typically) complex probability distributions.
- They involve possibly complex constraints that limit the decisions we can take.

The three problems described above feature a fourth characteristic:

- The probabilities and expectations are higher if more nodes or edges are selected, which makes these probability distributions *monotonic*.

We call these special cases of SCPs *stochastic constraint (optimisation) problems on monotonic distributions* (SCPMDs). While this fourth characteristic seems limiting, problems that have this property are plentiful in network analysis; examples include the applications mentioned above, but also a *signalling-regulatory pathway inference* problem described in the bioinformatics literature [24,52] and a variant on landscape connectivity [74]. We discuss the relation of SCPs and SCPMDs to other problems in Section 10.

Both SCPs and SCPMDs are difficult to solve exactly (*i.e.*, in a way that produces provably optimal solutions). Well-known instances of SCPMDs, such as spread of influence problems, are NP-hard [41]. Calculation of a probability in probabilistic networks requires solving a *counting* problem that is known to be #P-complete [62]. Exact solving requires finding an optimal solution in a search space that grows exponentially with problem size.

In this work we present two main approaches for solving SCPs exactly. The first is based on decomposing hard constraints on probability distributions into a multitude of local constraints, and is applicable to SCPs in general. The second is designed for solving SCPMDs (and can only be applied to those), exploiting structures that result from monotonicity to obtain a global constraint propagation algorithm for solving those stochastic constraints. We discuss SCPs, SCPMDs and how to solve them in Section 2.

The main algorithmic contributions in this work are as follows[1]:

1. We present new versions of a constraint decomposition method for solving SCPs [43], based on *ordered binary decision diagram* (OBDD) [14] representations of probability distributions (Section 4).

[1] We presented an earlier version of this work in a conference paper [44]. This article extends this earlier publication with a more elaborate description of the propagation algorithm from [44], new proofs and additional contributions 4–6. Preliminary versions of contributions 4 and 5 were presented in a workshop paper [31]. Here, we have extended that work by including earlier approaches from Latour et al. [43] to the configuration experiments and by significantly extending the design space of our methods.

2. We show that this decomposition approach is not *generalised arc consistent* (GAC), thus causing it to prune the search space insufficiently, and that a straightforward arc consistent modification of this approach does not significantly improve performance (Section 4.2).
3. To address this inefficiency in the search, we introduce a *global* constraint on OBDD representations of monotonic distributions, which we call the *stochastic constraint on monotonic distributions (SCMD)*, and introduce a GAC-by-design propagation algorithm for this constraint (Section 5).

In summary, the benefits of the decomposition methods are:

- They are applicable to the more generic SCPs (Section 4).
- Different types of decision diagrams can be used to represent the probability distributions (Section 3).
- The implementation is straightforward and compatible with different off-the-shelf *constraint programming (CP)* solvers or *mixed-integer programming (MIP)* solvers (Section 7).

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 5.2).
- Its space complexity is better than that of decomposition methods that do guarantee GAC (Sections 4.2 and 5.4).
- Its worst-case time complexity is $O(m+n)$ with OBDD size m and number of decision variables n (Section 5.4).
- It outperforms CP-based decomposition methods and complements MIP-based methods, while scaling better with OBDD size than MIP-based methods (Sections 7 and 9).

Different *design choices* made in the development of a solving method can have a big impact on the overall efficiency of an approach on different types of problem instances, especially for hard problems, such as SCPs and SCPMDs. We propose to address this problem by applying the *programming by optimisation* (PbO) paradigm [36] to our decomposition and global constraint optimisation methods and obtain the following additional contributions:

4. We develop several design alternatives for different parts of decomposition-based and global constraint optimisation pipelines and expose them as configurable parameters (Section 8.4).
5. We apply *automated algorithm configuration* (AAC) [35] to these configurable algorithms and demonstrate their effectiveness on benchmarks from two application domains (Section 9.3).
6. We then demonstrate how the optimised configurations of these methods generalise to harder problems and different problem settings (Section 9.4).

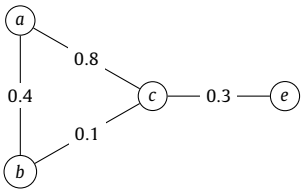
Note that in this work we compare methods that were specifically designed for solving SCPs to methods that use off-the-shelf solvers and their default settings. However, the performance of off-the-shelf solvers depends significantly on their configuration. Therefore, we also use AAC to help us to make a fairer comparison. To the best of our knowledge, this work represents the first time that AAC is applied to exact solving methods for probabilistic inference problems.

The remainder of this article is organised as follows. We start by describing SCPs and SCPMDs in Section 2. Then, in Section 3, we discuss how knowledge compilation can help us to represent probability distributions in a way that allows for tractable online probabilistic inference. We then show how we can use these representations to turn the constraints on probability distributions into CP and MIP models using a decomposition-based approach, in Section 4. Section 5 describes a global constraint for solving SCPMDs, whose performance we compare to the performance of CP-based decomposition methods in Section 7, on problem instances described in Section 6. We then briefly discuss PbO and AAC, as well as the parameter space of the decomposition and global stochastic constraint solving methods in Section 8. In Section 9 we use these expanded configuration spaces for our automated configuration experiments. Finally, we discuss related work in Section 10 and provide a summary of this work, as well as an outlook on future work, in Section 11.

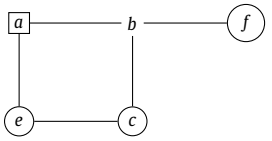
2. Stochastic constraint optimisation problems

We study *stochastic constraint (optimisation) problems (SCPs)* that are defined on Boolean *decision variables*. We are interested in finding one or more assignments of truth values to these decision variables, such that the given constraints are satisfied and, if provided, an optimisation criterion is optimised. We also refer to such assignments as *strategies*. The specific feature of *stochastic* constraint satisfaction problems is that the constraints and the optimisation criterion may be *stochastic*.

In this section we describe two hard constraints on probability distributions: one on generic distributions, the other on distributions with a monotonicity property. Then, we illustrate how to mathematically model the SCPs described in the introduction. Finally, we discuss why solving these kinds of problems requires some form of counting.



(a) A social network with vertices (a, b, c and e) representing people, and four undirected edges with mutually independent probabilities, representing their stochastic influence relationships.



(b) A power transmission grid with five vertices: one power producer (a), three power consumers (c, e and f) and one transition station (b). Edge probabilities depend on strategy and are therefore omitted from this figure.

Fig. 1. Two examples of probabilistic networks.

2.1. Stochastic constraints on probability distributions

The first stochastic constraint we study in this work is of the form

$$\sum_{\phi \in \Phi} \rho_\phi \cdot P(\phi \mid \sigma) > \theta. \tag{1}$$

The sum represents an *expected utility*, in which Φ is a set of stochastic events that are of interest to us, $P(\phi \mid \sigma)$ represents the probability of an event ϕ happening, given a strategy σ; and $\rho_\phi \in \mathbb{R}^+$ is a reward for this event. Specifically, we focus on stochastic constraints in which ϕ can be represented as a propositional formula on Boolean decision variables X, such that $\sigma : X \mapsto \{\top, \bot\}$, and Boolean stochastic variables T, such that each variable $t \in T$ takes value $\top$ with probability $P(t = \top)$ and value $\bot$ with probability $P(t = \bot) = 1 - P(t = \top)$.

A special case of the constraint in Equation (1) is one where we require each probability distribution $P(\phi \mid \sigma)$ to be *monotonic*, which we define as follows:

Definition 2.1. Let $\phi(X, T)$ be a propositional formula on Boolean decision variables X and Boolean stochastic variables T, as defined above. We call the probability distribution $P(\phi \mid \sigma)$ a *monotonic distribution* if, for all strategies σ and each $d \in X$, the following holds:

$$P(\phi \mid \sigma_{d=\bot}) \leq P(\phi \mid \sigma_{d=\top}), \tag{2}$$

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

We define a stricter notion of *local monotonicity* in Definition 3.1, which we use to formally define a *stochastic constraint on monotonic distributions* (SCMD) in Definition 5.1. We refer to SCPs whose relevant probability distributions all exhibit this local monotonicity as *stochastic constraint (optimisation) problems on monotonic distributions* (SCPMDs).

2.2. Modelling SCPs

Many problems on probabilistic networks can be seen as SCPs. We illustrate this with two running examples: a spread of influence problem [26,41] and a power grid reliability problem [29].

Example 2.1 *(Spread of influence: SCP).* Consider the network in Fig. 1a. Edges represent probabilistic mutual influence relationships, meaning that a person u influences another person v with probability p_{uv}, which labels edge (u, v). The probability that u does not influence v is $(1 - p_{uv})$. We distribute free samples to a subset of the individuals, who can probabilistically influence others to also become customers. The objective is to maximise the expected number of people who become our customer, given a limited number k of free samples to distribute to people in the network. We model this as follows:

- With each vertex i in the network we associate a Boolean decision variable $d_i \in \{\top, \bot\}$, representing whether person i receives a free sample.
- We are interested in the events $\Phi = \{\phi_a, \phi_b, \phi_c, \phi_e\}$, where ϕ_i denotes the event of person i being our customer.
- Our objective is to find a strategy σ that maximises the expected utility $\sum_{i \in \{a,b,c,e\}} \rho_i \cdot P(\phi_i \mid \sigma)$, where we fix $\rho_i := 1$.
- Constraint: $\sum_{i \in \{a,b,c,e\}} c_i \cdot d_i \leq k$ (threshold $k \in \mathbb{N}^+$), where we fix $c_i := 1$.

Example 2.2 *(Power grid reliability: SCP).* Consider the network in Fig. 1b. Here, each edge represents a power line (u, v) that has a probability p_{uv} of remaining intact during a natural disaster. This probability may depend on, *e.g.*, its length or the terrain in which it exists. If too many lines are damaged, consumers may lose power. By spending money to reinforce a power line, we can increase its probability of surviving a disaster. Our goal is to use our budget b for maintaining power

lines wisely, such that the expected number of consumers that will still have power after a natural disaster is maximised. We make the simplifying assumption that a consumer still has power if they are still connected to a power source, and we model this problem as follows:

- We distinguish three types of stations: power consumers $V_{cons} = \{c, e, f\}$, power producers $V_{prod} = \{a\}$ and transmission stations $V_{trans} = \{b\}$, such that $V_{cons} \cap V_{prod} \cap V_{trans} = \varnothing$ and $V_{cons} \cup V_{prod} \cup V_{trans} = V$ is the set of vertices in the network.
- With each power line $(u, v) \in L$ we associate a decision variable $d_{uv} \in \{\top, \bot\}$ that indicates whether or not a power line is reinforced.
- We are interested in events $\Phi = \{\phi_i : i \in V_{cons}\}$, where ϕ_i represents that consumer i is still connected to a power producer after a natural disaster.
- Our objective is to find a strategy σ that maximises the expected utility $\sum_{i \in V_{cons}} \rho_i \cdot P(\phi_i \mid \sigma)$, where we fix $\rho_i := 1$.
- Constraint: $\sum_{i \in L} c_i \cdot d_i \leq b$ (threshold $b \in \mathbb{N}^+$), where $c_i := 1$.

Note that in both examples, we fix $c_i = \rho_i = 1$ for reasons of simplicity, but it is straightforward to use alternative values.

Observe that each of the two examples above involves a stochastic *objective function*, rather than a stochastic *constraint*. We can straightforwardly employ constraints on probability distributions to solve maximisation problems over expected utilities (under other constraints). In the examples above, we want to maximise the expected utility without exceeding our budget. We can solve this problem by first writing it as a constraint *satisfaction* problem, and repeatedly solving that problem. Thus, we start with the two constraints

$$\sum_{0 \leq i < |\Phi|} \rho_i \cdot P(\phi_i \mid \sigma) > \theta \quad \text{and} \quad \sum_{0 \leq i < |X|} c_i \cdot d_i \leq b,$$

setting $\theta := 0$. When we find a solution $\sigma_{\text{sol}} : X \mapsto \{\top, \bot\}$ to this constraint with a utility $\rho_{\text{sol}} = \sum_{0 \leq i < |\Phi|} \rho_i P(\phi_i \mid \sigma_{\text{sol}})$, we update θ such that the stochastic constraint is no longer satisfied, setting $\theta := \rho_{\text{sol}}$. We continue searching until we find a new solution, which, if it exists, is guaranteed to have a higher utility than the previously found solution. We repeat this process until we cannot find a new solution, at which point θ represents the maximum possible expected utility.

In order to complete our models for the problems described in Examples 2.1 and 2.2, we must define the probability of events ϕ_i, given a strategy. While there are many formalisms for defining these conditional probabilities, we use a *weighted model counting* (WMC) approach in this work, where weighted propositional formulae define the distributions; this approach generalises many other well-known approaches and its use is common practice in the domains of probabilistic reasoning, planning and learning [5,17,22,25,27,29,30,64].

In order to take a WMC approach to modelling probability distributions, we express each event ϕ as a propositional formula $\phi(X, T)$ on Boolean decision variables X and Boolean stochastic variables T. A model is an assignment of truth values to the variables in $X \cup T$, such that $\phi(X, T)$ evaluates to *true*. We assign a weight w_t to a variable $t \in T$ reflecting its probability of being *true* when we sample a possible world: $w_t := p(t = \top)$ and $w_{\bar{t}} := p(t = \bot) = 1 - w_t$ [65].

We use $\phi|_\sigma(T)$ to denote the *residual formula* that is obtained by replacing all decision variables $d \in X$ in propositional formula $\phi(X, T)$ by their truth values as specified in strategy σ. This allows us to evaluate the probability of ϕ being *true*, given that strategy σ. Under the WMC approach, the following holds:

$$P(\phi \mid \sigma) = \sum_{\mu \in M} \prod_{t \in \mu} w(t), \tag{3}$$

where μ is a set of truth assignments to all stochastic variables in T, such that μ is a model of $\phi|_\sigma(T)$, M is the set of all models of $\phi|_\sigma(T)$, $t \in T$ is a stochastic variable, $w(t) := w_t$ if $t = \top$ in μ, and $w(t) := w_{\bar{t}}$ if $t = \bot$ in μ.

We now illustrate how weighted model counting can be used to formalise the probability distributions from our running examples.

Example 2.3 *(Spread of influence: WMC).* We model this problem under the following simplifying assumptions:

- Influence relationships are symmetric.
- Once someone gets a free product sample, they will become a customer.
- If u influences v, and u is a customer, then v becomes a customer.

The possible worlds in which the event ϕ_e takes place in Fig. 1a can then be modelled by the propositional formula $\phi_e = d_e \vee (d_c \wedge t_{ce}) \vee (d_b \wedge t_{bc} \wedge t_{ce}) \vee (d_a \wedge t_{ac} \wedge t_{ce}) \vee (d_b \wedge t_{ba} \wedge t_{ac} \wedge t_{ce}) \vee (d_a \wedge t_{ab} \wedge t_{bc} \wedge d_{ce})$. This formula represents all the different situations in which e becomes a customer. We use two types of variables: d_i are the decision variables of the SCP and t_{ij} are associated with each edge (i, j) in the network and represent influence. One possibility for event ϕ_e to happen is when person c gets a free sample and influences person e.

To define a distribution over the network, we associate a probability $p\left(t_{ij}\right)$ with each Boolean variable t_{ij} that this variable is *true*. We call t_{ij} a *stochastic variable*. The probability $P\left(\phi_e \mid \sigma\right)$ is then defined as the sum of the probabilities of all the (logical) models of this formula, given the strategy. Given strategy $\sigma = (d_a = \top, d_b = d_c = d_e = \bot)$, an example of a model is $t_{ac} = t_{ce} = \top, t_{ab} = t_{bc} = \bot$, with probability $0.8 \cdot 0.3 \cdot (1 - 0.4) \cdot (1 - 0.1) = 0.1296$.

Example 2.4 *(Power grid reliability: WMC).* For the sake of simplicity, we make the following assumptions:

- All power lines have the same survival probability p_{uv} if not reinforced and the same survival probability p'_{uv} if reinforced, with $p'_{uv} > p_{uv}$.
- With each power line $(u, v) \in L$ we associate a survival probability π_{uv} that takes the following values:

$$\pi_{uv} := \begin{cases} p_{uv} & \text{if } d_{uv} = \bot; \\ p'_{uv} & \text{if } d_{uv} = \top. \end{cases} \tag{4}$$

The possible worlds in which event ϕ_e takes place in Fig. 1b are defined by the propositional formula $\phi_e = (t_{ae} \vee (s_{ae} \wedge d_{ae})) \vee ((t_{ab} \vee (s_{ab} \wedge d_{ab})) \wedge (t_{bc} \vee (s_{bc} \wedge d_{bc})) \wedge (t_{ce} \vee (s_{ce} \wedge d_{ce})))$. This formula specifies all the situations in which consumer e will still be connected to a producer after a natural disaster. Again, we use two types of variables: d_{uv}, the decision variables of the SCP, and t_{uv} and s_{uv}, the stochastic variables associated with each edge (u, v) in the network, to represent the stochastic survival of the power line. In this model, we associate the following probabilities with variables t_{uv} and s_{uv}: $P\left(t_{uv} = \top\right) = p_{uv}$, and $P\left(s_{uv} = \top\right) = {}^{(p'_{uv} - p_{uv})}/_{(1 - p_{uv})}$. Note that these probabilities are defined such that they model π_{uv}.

Consequently, for $p_{uv} := 0.4$ and $p'_{uv} := 0.875$ (values from the literature [29]) and strategy $\sigma = \{d_{ae} = d_{bc} = \top, d_{ab} = d_{bf} = d_{ce} = \bot\}$, one example of a model for $\phi_e|_\sigma(T)$ is: $s_{ae} = t_{ab} = s_{ab} = t_{ce} = s_{bf} = \top, t_{ae} = t_{bc} = s_{bc} = s_{ce} = t_{bf} = \bot$, with probability $0.79167^3 \cdot 0.20833^2 \cdot 0.4^2 \cdot 0.6^3 = 7.44235 \cdot 10^{-4}$.

2.3. Complexity of SCP solving

Solving SCPs and SCPMDs exactly is NP-hard in the general case. There are two components to solving SCPs, and we now briefly address their computational complexity. To evaluate the quality of a strategy σ, we have to compute $P(\phi \mid \sigma)$, which is #P-complete in general [62]. We have to compute $P(\phi \mid \sigma)$ a potentially exponential number of times, since the number of possible strategies for n decision variables is 2^n. Thus, naïvely solving a SCP by enumerating all possible strategies and evaluating their score is typically impractical.

In the following section, we describe how we can use existing *knowledge compilation* techniques to make the problem of probabilistic inference more tractable during the search process, by means of preprocessing. In the next two sections, we present methods that use these techniques in combination with CP and MIP technology for efficient traversal of the search space.

3. Probabilistic inference

There are various methods for obtaining propositional formulae that model events ϕ. However, not all of them are equally suitable. For formulae in conjunctive normal form (CNF), weighted model counting is known to be #P-complete [62]. Therefore, computing the weighted model count of $\phi|_\sigma$ may involve solving an exponential-time search problem. We address this problem by limiting our attention to representations of propositional formulae from which a probability can be calculated in polynomial time. Hence, we separate the problem of finding a suitable representation of the distribution from the problem of finding a solution to the SCP. We use *knowledge compilation* to obtain these representations.

3.1. Knowledge compilation

Historically, knowledge compilation has been a popular method for making online WMC computation more tractable in the field of probabilistic inference and planning [17,22,25,30,37]. Knowledge compilation allows us to compactly represent the truth table of a propositional formula as a *decision diagram* (DD). In this work, we study the use of compiling propositional formulae to *ordered binary decision diagrams* (OBDDs) [14] to facilitate tractable WMC, in the context of solving SCPs. Once a formula $\phi(X, T)$ is compiled into a decision diagram, the time complexity of computing $P(\phi \mid \sigma)$ is linear in the size of the diagram. Note that, in the constraint programming literature, OBDDs and other types of decision diagrams are often employed as compact representations for the satisfying assignments of a constraint [32,33]. Here, we use these diagrams differently.

For the performance of algorithms that use DDs, the size of the DD matters. Note in particular that OBDD representations of probability distributions resulting from probabilistic networks may still be exponential in the size of those networks, which is particularly relevant for the SCPs considered in this work. The size and shape of an OBDD are determined by its variable order. Unfortunately, finding a variable order that produces a minimal-size OBDD, is NP-hard [10]. However, a key

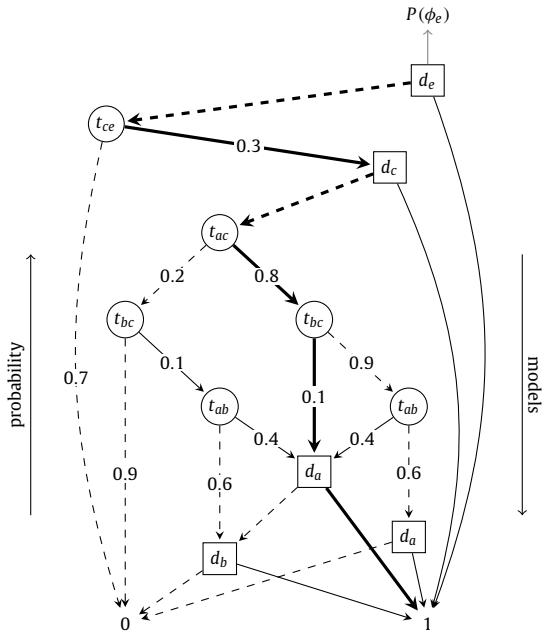


Fig. 2. An OBDD representation of ϕ_e. Circular nodes represent stochastic variables, square nodes represent decision variables. This OBDD has variable order $d_e \prec t_{ce} \prec d_c \prec t_{ac} \prec t_{bc} \prec t_{ab} \prec d_a \prec d_b$.

advantage of using knowledge compilation in an SCP solving pipeline is that the resulting approach is modular, and that part of the model counting problem can be solved in a preprocessing phase by the knowledge compiler, which enables tractable online inference.

There are decision diagrams that can be used to produce more compact representations of probability distributions, such as *sentential decision diagrams (SDDs)* [12,56] and *weighted positive binary decision diagrams (WPBDDs)* [19]. However, OBDDs have properties that we can exploit for a propagation algorithm for stochastic constraints on monotonic distributions, as we show in Section 5. Therefore our focus is on OBDDs.

To construct OBDDs for networks, we use an approach implemented in the ProbLog system [25,30]. This system provides a convenient way of modelling paths in probabilistic networks and uses knowledge compilers to create OBDD representations of propositional formulae.

3.2. Performing WMC with OBDDs

Fig. 2 demonstrates how we can compute $P(\phi \mid \sigma)$ in time that is linear in the size of the OBDD representation of the probability [20,22]. The outgoing arcs of OBDD nodes represent the truth values of the variables labelling those nodes. Specifically, the *hi* (solid) arc represents *true*, while the *lo* (dashed) arc represents *false*. A path from the root of an OBDD to the leaf labelled 1 corresponds to a (sub)set of the set of (variable, truth value) pairs that form a model for the formula encoded by the OBDD. Here, the value of the variable is determined by which outgoing arc of the node labelled with that variable is taken in the path. This subset satisfies the formula, and its weight equals the sum of the weights of all models that are its supersets. Each model of the formula is defined by exactly one such path/(sub)set.

We map this OBDD to an *arithmetic circuit* (AC) to compute the probability that ϕ_e in Example 2.3 evaluates to *true* under a strategy σ as follows. The weights on the outgoing arcs of nodes that represent stochastic variables (those labelled with t_{ij}) correspond to the probability that a variable is *true* (*hi* arcs) or *false* (*lo* arcs). We add a strategy σ on the OBDD by adding weights of 0 and 1 to the appropriate outgoing arcs of the nodes labelled with decision variables.

We can now compute $P(\phi_e \mid \sigma)$ as follows. In a bottom-up traversal of the OBDD, each node r is assigned the *score*

$$v(r) := w(r) \cdot v\left(r^+\right) + (1 - w(r)) \cdot v\left(r^-\right), \tag{5}$$

where $0 \leq w(r) \leq 1$ represents the weight of the variable that labels r, and r^+ (r^-) is the *hi* (*lo*) child of r, *i.e.*, the child connected through the solid (dashed) outgoing arc of r. In the base cases we have $v(r) := 0$ for the negative leaf and $v(r) := 1$ for the positive leaf. Observe that $v(root) = P(\phi \mid \sigma)$. Note that it takes one *bottom-up* traversal of this AC to compute the score of the root.

In the interest of brevity, in the remainder of this work, we will sometimes abuse terminology and refer to the OBDD when we actually mean the AC that the OBDD is mapped onto.

Example 3.1 *(WMC on an OBDD).* Consider the OBDD in Fig. 2. Suppose we want to compute $P(\phi_e \mid \{d_e = d_c = \bot, d_a = d_b = \top\})$. We label the dashed outgoing arcs of nodes labelled with d_e and d_c, as well as the solid outgoing arcs of nodes labelled with d_a and d_b with the value 1. Similarly, we label the solid outgoing arcs of nodes labelled with d_e and d_c, as well as the dashed outgoing arcs of nodes labelled with d_a and d_b with the value 0. Then, we perform a bottom-up sweep of the diagram to compute the score of each node, by computing the weighted sum of its children, as per Equation (5).

This yields a score of 1 for the nodes labelled with d_a, d_b and t_{ab}, and for the right node labelled with t_{bc}. The left t_{bc} node has a score of 0.1. The t_{ac} and d_c nodes each have a score of 0.82, and the nodes labelled with t_{ce} and d_e each have a score of 0.246. Because the node labelled with d_e is the root node of the diagram, we conclude that $P(\phi_e \mid \{d_e = d_c = \bot, d_a = d_b = \top\}) = 0.246$.

3.3. OBDDs and monotonicity

In this work, we consider two types of probability distributions: those that are *monotonic* and those that are not (Section 2.1). 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; likewise, 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.

Recall our definition of a monotonic probability distribution in Definition 2.1. For OBDDs, we also define the concept of local monotonicity:

Definition 3.1. Let $\phi(X, T)$, σ and $P(\phi \mid \sigma)$ be defined as in Section 2.1. We call an OBDD representation of a probability distribution whose score at the root equals $P(\phi \mid \sigma)$ *locally monotonic*, iff the following property holds for any projected σ (see Section 3.2):

$$v\left(r_d^-\right) \leq v\left(r_d^+\right), \tag{6}$$

for each OBDD node r_d labelled with a decision variable $d \in X$, using Equation (5) to compute $v\left(r_d^-\right)$ and $v\left(r_d^+\right)$.

Theorem 3.1. *If a probability distribution $P(\phi \mid \sigma)$ can be represented by a locally monotonic OBDD as defined in Definition 3.1, then it is a monotonic distribution, as defined in Definition 2.1.*

Proof. In this proof, we use $v(r \mid \sigma_{d=\bot})$ to denote the score of an OBDD node r, computed using Equation (5), for an OBDD with strategy σ, in which decision variable $d = \bot$, and analogously for $d = \top$. Since the root of the OBDD is an OBDD node, and its score represents $P(\phi \mid \sigma)$, the task is to prove that, for any node r in a locally monotonic OBDD, it holds that $v(r \mid \sigma_{d=\bot}) \leq v(r \mid \sigma_{d=\top})$. 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.

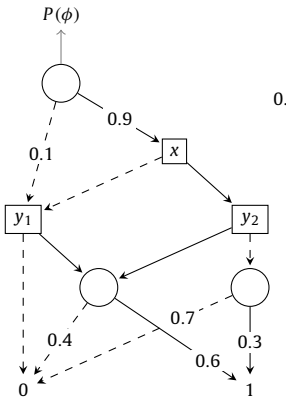
For the first case, the inequality holds by Definition 3.1. For the second case, $v(r)$ is determined by only one child, because σ assigns a truth value to each decision variable, which fixes $w(r)$ to either 0 or 1 in Equation (5). Since the inequality holds for this one child, it also holds for r. For the third case, the inequality holds for the two children. Since $v(r)$ is the weighted sum of those two children, the inequality also holds for r. □

It is yet unknown if the reverse of Theorem 3.1 holds. Ensuring that distributions are monotonic is relatively easy in the WMC approach: for any program written in the ProbLog system [25,30] without negation, the resulting OBDD distribution 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. An illustration of why distributions encoded using negation do not always yield locally monotonic OBDDs is provided in Appendix A. We will use the notion of local monotonicity to define a global propagation algorithm for SCMDs in Section 5.

4. Solving SCPs with decomposition

In Section 2.3, we discussed the two elements that we need for efficient SCP solving: tractable WMC and an efficient way of traversing the search space. As described above, we propose to use knowledge compilation for the first element.

We address the second element in this section. We propose leveraging *constraint programming (CP)* and *mixed-integer programming (MIP)* technology to efficiently traverse the search space. We assume familiarity with these paradigms and refer readers who would like to learn more about them to the literature [13,61].



$$
\begin{aligned}
P(\phi \mid \sigma) &\geq 0.4 \\
0.1 \cdot z_{y_1} + 0.9 \cdot z_x &\geq 0.4 \\
z_x &:= (1-x) \cdot z_{y_1} + x \cdot z_{y_2} \\
z_{y_1} &:= 0.6 \cdot y \\
z_{y_2} &:= 0.6 \cdot y + 0.3 \cdot (1-y) \\
x, y &\in \{0, 1\} \\
0 \leq P(\phi \mid \sigma) &\leq 0.6 \\
0 \leq z_x &\leq 0.6 \\
0 \leq z_{y_1} &\leq 0.6 \\
0.3 \leq z_{y_2} &\leq 0.6
\end{aligned}
$$

Fig. 3. A small OBDD (left) with three stochastic variables (circles) and two decision variables x and y (squares). The two nodes corresponding to decision variable y are indexed for clarity. The decomposition on the right is constructed using Equation (5).

In the following, we recall a method for combining knowledge compilation and CP and MIP technology for SCP solving that we first proposed in 2017. This method decomposes a global constraint on a DD representation of a probability distribution into a multitude of local constraints [43]. We also show that this method does not guarantee generalised arc consistency (GAC) and propose a straightforward way to turn this decomposition method into one that does guarantee GAC.

Recall from Section 3.1 that we can represent probability distributions using OBDDs. We now explain how OBDDs can be converted into a *linear program* (LP) that encodes a constraint on a probability distribution (Equation (1)), and identify a problem with this approach.

Note that, in our earlier work, we also showed how SDDs can be used to obtain linear programs from stochastic constraints such as Equation (1), formulated on SDD-representations of probability distributions [43]. However, the discussion of that method is outside the scope of this work.

4.1. Decomposing a constraint on an OBDD

Recall that we represent a specific strategy by labelling the outgoing arcs of OBDD nodes labelled with decision variables with the values 0 and 1. Our aim is to solve Equation (1), which we interpret as a constraint on the values we can use to label those arcs. We can thus interpret Equation (1) as a constraint on the AC induced by the OBDD encoding of the probability distribution of an SCP.

Fig. 3 shows an example of an OBDD representation of a formula ϕ. We impose the constraint $P(\phi \mid \sigma) \geq 0.4$. Fig. 3 also shows a decomposition of this constraint on the whole OBDD into a multitude of smaller constraints, adding auxiliary variables z_{y_1}, z_{y_2} and z_x, whose domains include real numbers.

The next step to solving $P(\phi \mid \sigma) \geq 0.4$ is to simply feed this set of smaller, local constraints to a CP or MIP solver [43]. The decomposition in Fig. 3 contains quadratic constraints involving real-valued variables (the auxiliary variables z in the decomposition in Fig. 3), which are hard to solve for MIP solvers and CP solvers. We therefore linearise this decomposition using the big-M approach [49] (with $M \leq 1$), for easier solving.

For example, the quadratic constraint $z_x := (1-x) \cdot z_{y_1} + x \cdot z_{y_2}$ can be linearised as follows:

$$
\begin{array}{lll}
z_x := z_{x_\top} + z_{x_\bot} & z_{x_\top} \leq x & z_{x_\bot} \leq 1 - x \\
0 \leq z_{x_\top} \leq 1 & z_{x_\top} \leq z_{y_2} + (1-x) & z_{x_\bot} \leq z_{y_1} + x \\
0 \leq z_{x_\bot} \leq 1 & z_{x_\top} \geq z_{y_2} - (1-x) & z_{x_\bot} \geq z_{y_1} - x
\end{array}
\tag{7}
$$

We linearise the program by repeating this method for all quadratic constraints.

4.2. GAC guarantees of decomposition

We briefly recall the well-known concept of generalised arc consistency (GAC) [61]. We note that a CP solver works by fixing unbound variables to values during branching, and updating the domains of the remaining unbound variables while it builds a partial solution through branching and propagation. Let x be a variable and $dom(x)$ its domain. A (variable, value) pair (x, a) with $a \in dom(x)$ is considered GAC with respect to a constraint c iff there exists an assignment in the current domains of the variables in the scope of c that satisfies c and in which $x = a$. Propagation establishes GAC for a constraint c if all remaining values of all variables in the scope of c are GAC.

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

Theorem 4.1. *Propagation on the decomposed representation of the SCP as described in [43] does not guarantee GAC.*

Proof. We prove this by contradiction. Assume that propagation in the decomposition method from [43] is GAC. Then, the following counterexample leads to a contradiction.

Consider the OBDD in Fig. 3 and associated constraint $P(\phi \mid \sigma) \geq 0.4$. Observe that the four possible strategies yield these conditional probabilities:

$$P(\phi \mid x = y = 0) = 0 \qquad P(\phi \mid x = 1, y = 0) = 0.3$$
$$P(\phi \mid x = y = 1) = 0.6 \qquad P(\phi \mid x = 0, y = 1) = 0.6$$

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 Fig. 3, 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,

$$v(r_d) := \begin{cases} v\left(r_d^+\right) & \text{if } d := \top \\ v\left(r_d^-\right) & \text{if } d := \bot \end{cases} \quad \text{with} \quad v(r_d) := \max\left(d \cdot v\left(r_d^+\right), (1-d) \cdot v\left(r_d^-\right)\right),$$

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 (5), conditioned on $d = \bot$. Each of these constraints encodes explicitly what is required to maintain GAC: if setting a decision variable d to 0 causes the score at the root of the OBDD to be too low to satisfy Equation (1), then this decision variable must be fixed to 1, instead.

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 $O(|X| \cdot |OBDD| \cdot \tau)$ for this approach, where X is the set of decision variables, $|OBDD|$ the size of the OBDD, and τ the depth of the search tree. We demonstrate the practical inferiority of this approach in Section 7.

5. Solving SCPMDs with global propagation

For the special case of constraints on monotonic distributions, we intend to improve upon the OBDD decomposition approach described in Section 4.1 by developing a *global* constraint on an OBDD representation of such distributions. We first define this constraint and then introduce a corresponding propagator that guarantees GAC. We refer to this propagator as the *global SCMD propagator*. Our propagator relies on Theorem 5.1 for efficient constraint propagation. It also relies on the local monotonicity of the OBDD representation of the probability distribution (Definition 3.1) to be able to guarantee GAC.

In the following, we refer to OBDD nodes labelled with a decision variable as *decision nodes* and to OBDD nodes labelled with a stochastic variable as *stochastic nodes*.

5.1. Stochastic constraint on monotonic distributions

Recall the definition of locally monotonic OBDDs, Definition 3.1. We now define a corresponding *stochastic constraint on monotonic distributions* (SCMD) as follows:

Definition 5.1. For a set of propositional formulae Φ, threshold $\theta \in \mathbb{R}^+$ and utilities $\rho_\phi \in \mathbb{R}^+$, we call

$$\sum_{\phi \in \Phi} \rho_\phi \cdot P(\phi \mid \sigma) > \theta \tag{8}$$

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

Given a partial strategy σ, a GAC-guaranteeing propagator for the SCMD will, for each unbound decision variable $d \in X$, remove value $\bot$ from $dom(d)$ if, and only if,

$$\sum_{\phi \in \Phi} \rho_\phi \cdot P\left(\phi \mid \sigma'\right) \leq \theta \tag{9}$$

holds for each possible extension of partial strategy σ to a full strategy σ' that includes $d = \bot$.

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 subformulae. For simplicity of discussion and notation, we only consider constraints over one propositional formula ϕ in this section, and thus only single-rooted OBDDs, but the approach can be easily extended to a more general situation, as indeed we have done in our implementation. This makes the corresponding utility ρ irrelevant in the discussion and limits the domain of threshold θ to $(0, 1)$, to which we compare a probability rather than an expectation.

5.2. Naïve SCMD propagation

For maintaining GAC, a key observation is that our *scoring function* (the expected utility in Equation (8)) is monotonic; hence, the largest possible score is obtained by assigning the value *true* to all unbound decision variables. Given an OBDD representation of $\phi(X, T)$, mapped to an AC, the following process for each unbound decision variable $d \in X$ would be GAC:

1. Fix variable d to the value $\bot$.
2. Fix all remaining unbound variables to the value $\top$.
3. Calculate the root node score for the resulting assignment with Equation (5).
4. If the score is lower than or equal to θ, remove value $\bot$ 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 3.1. Consequently, if that upper bound is lower than θ, 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 θ. 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 SDDs and d-DNNFs [57].

Let n be the number of unbound decision variables, and let m be the size of the OBDD. Then the complexity of this *naïve SCMD propagator* is $O(n \cdot m)$: for every unbound variable, we perform a bottom-up traversal of the OBDD. Since this propagation algorithm must be called in each node of the search tree, and the size of the OBDD can be exponential in the size of the input problem, this complexity may be prohibitively large. Therefore, will now show how to improve the complexity of this propagator to $O(n + m)$.

5.3. A full-sweep SCMD propagator

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

$$\frac{\partial f(d, \sigma' \setminus \{d\})}{\partial d} = f\left(\sigma'\right) - f\left(d = \bot, \sigma' \setminus \{d\}\right) \tag{10}$$

for every unbound decision variable d. Here, function f represents the scoring function defined by Equation (5) on the root of the OBDD. Strategy σ' represents an assignment to all decision variables obtained by taking partial assignment σ and extending it by assigning *true* to each unbound decision variable in X.

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

$$f(\sigma') - \frac{\partial f(d, \sigma' \setminus \{d\})}{\partial d} > \theta. \tag{11}$$

The main question becomes how to calculate the partial derivative for all unbound variables efficiently. Here, we build on ideas introduced by Darwiche [20,21] to build a linear algorithm that can furthermore maintain derivatives incrementally. We first need to define the concept of *path weight*:

Definition 5.2. Let r_m be a node labelled with variable v_m in an OBDD with variable order $x_1 \prec \ldots \prec x_n$. We define the *path weight of* r_m as

$$\pi(r_m) := \sum_{\ell \in L_{r_m}} \prod_{r_i \in \ell} u(i), \tag{12}$$

where ℓ is a path from the root of the OBDD to r_m, 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* (respectively, *true* or *unbound*).

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:

$$u(i) := \begin{cases} w(i) & \text{if we take the hi arc of } r_i; \\ 1 - w(i) & \text{if we take the lo arc of } r_i. \end{cases} \tag{13}$$

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 ρ. Because, for simplicity, we assume the diagram to have just a single root in this section, ρ is irrelevant and the path weight at the root is initialised to 1.

Our global SCMD propagation algorithm is based on the following:

Theorem 5.1. *The partial derivative of the OBDD with respect to an unbound decision variable d can be calculated as follows:*

$$\frac{\partial f(d, \sigma' \setminus \{d\})}{\partial d} = \sum_{r_d \in OBDD_d} \pi(r_d) \cdot \left(v(r_d^+) - v(r_d^-)\right), \tag{14}$$

where $OBDD_d$ is the set of OBDD nodes labelled with decision variable d.

Proof. Observe that Equation (10) can be read as:

$$\frac{\partial f(d, \sigma' \setminus \{d\})}{\partial d} = v|_{\sigma' \setminus \{d\}, d=\top}(r) - v|_{\sigma' \setminus \{d\}, d=\bot}(r) \tag{15}$$

$$\begin{aligned} &= \left(u_{r \to r^+} \cdot v|_{\sigma' \setminus \{d\}, d=\top}(r^+) + u_{r \to r^-} \cdot v|_{\sigma' \setminus \{d\}, d=\top}(r^-)\right) - \\ &\quad \left(u_{r \to r^+} \cdot v|_{\sigma' \setminus \{d\}, d=\bot}(r^+) + u_{r \to r^-} \cdot v|_{\sigma' \setminus \{d\}, d=\bot}(r^-)\right), \end{aligned} \tag{16}$$

where r denotes the root of the OBDD and $v|_{\sigma' \setminus \{d\}, d=\bot}(r)$ the score at root r (calculated using Equation (5)), conditioned on partial strategy σ, extended by fixing d to $\bot$ and all other unbound decision variables to $\top$.

The expression in Equation (15) states that the partial derivative of f equals the difference of the score of Equation (5) taken at the root of the OBDD, conditioned on σ' and either $d = \top$ or $d = \bot$. In Equation (16), we have expanded the expressions on each side of the '$-$'-sign according to Equation (5). 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 (5) 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 expanding $v|_{\sigma' \setminus \{d\}, d=\top}(r)$ in Equation (15)).
3. Terms with $v(r_d^-)$ (from expanding $v|_{\sigma' \setminus \{d\}, d=\bot}(r)$ in Equation (15)).

The terms of type 1 correspond to paths from OBDD root to leaf that do not contain an OBDD node labelled with d. Therefore, the terms on the right of the '$-$'-sign in Equation (16) 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 ℓ from the root r to r_d. Hence, we can rewrite these terms as follows:

$$\begin{aligned} &u_{r \to r^+} \cdot \ldots \cdot u_{r_d \to r_d^+} \cdot v(r_d^+) - u_{r \to r^-} \cdot \ldots \cdot u_{r_d \to r_d^-} \cdot v(r_d^-) \\ &\qquad = u_{r \to r^+} \cdot \ldots \cdot u_{r_d \to r_d^+} \cdot \left(v(r_d^+) - v(r_d^-)\right), \end{aligned} \tag{17}$$

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 (14). □

We use the observation above to create an $O(n + m)$ 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 node with 0. We update the path weight of its children r^+ and r^- as follows if r is labelled with a decision variable d:

$$\begin{aligned} \pi\left(r^+\right) &:= \pi\left(r^+\right) + \pi(r) \text{ if } d \text{ is unbound or } \textit{true}; \\ \pi\left(r^-\right) &:= \pi\left(r^-\right) + \pi(r) \text{ if } d \text{ is } \textit{false}; \end{aligned} \tag{18}$$

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 (5) 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 (14), and evaluate Equation (11) for each of those to see if we can remove *false* from their domain.

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

5.4. A partial-sweep SCMD propagator

We now explore whether we can further reduce the empirical running time of the algorithm above, by avoiding 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 (12)) only contains variables $x_i \prec x_m$. 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 nodes in the OBDD above the decision nodes closest to the root, as 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 variable closest to the leaves. 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 Appendix B for the pseudocode of the resulting partial-sweep algorithm.

O1–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.

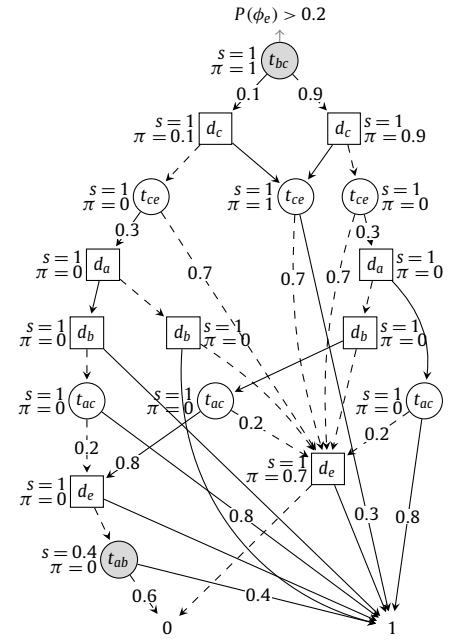
In our implementation of the partial-sweep algorithm, we maintain for each OBDD node r three counters, addressing observations O4–O6. Maintaining these counters requires two extra passes through part of the OBDD each time the propagator is called. However, they allow us to traverse an ever-decreasing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

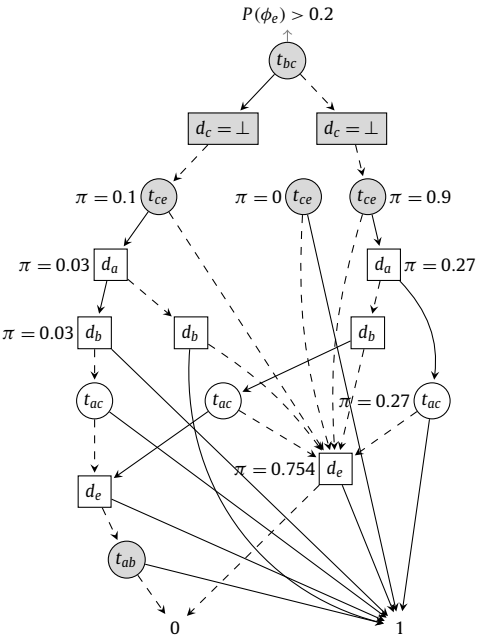
The second indicates the number of children r' for which there is at least one valid path from r' to an unbound decision node below r'; we call this counter `FreeOut[r]`. If its value is 0, any changes in r's path weight need not be propagated down from r, because of O5.

Because of O6, we use a third counter, which we call `Reachable[r]`. It indicates the number of parents r' for which there is at least one valid path from an OBDD root to r'. If not, r's path weight is 0, and changes in its score need not be propagated.

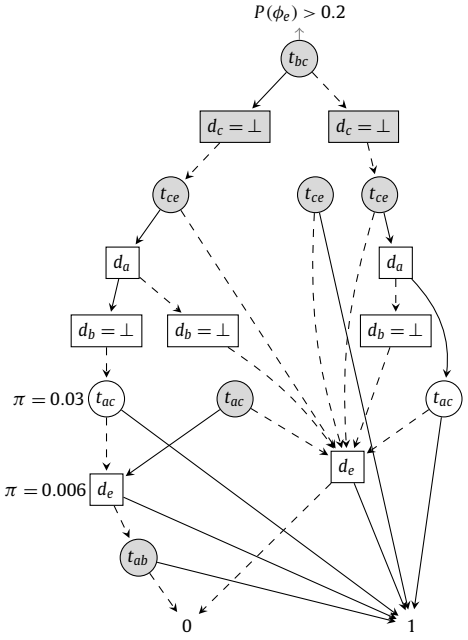
Example 5.1 *(Partial-sweep propagation).* Fig. 4 shows an example execution of the partial-sweep algorithm on an OBDD representation of ϕ_e from Example 2.1. When unbound decision nodes become fixed, we remove one of their outgoing arcs to indicate their truth value. This may cause nodes to become no 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. Next to each node, we indicate its current score s and current path weight π. We only indicate the scores and path weights that



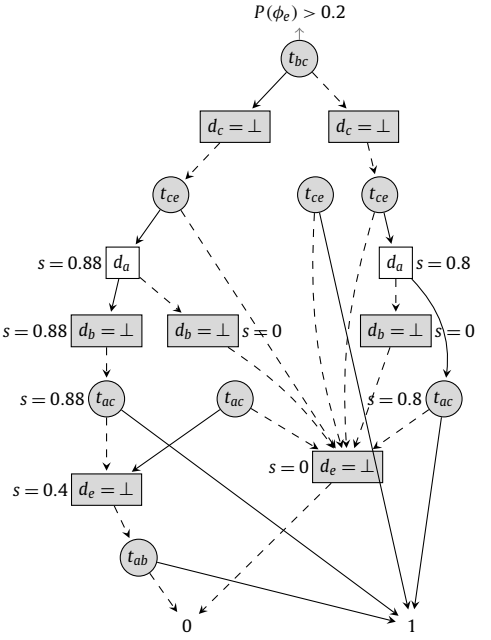
(a) Iteration 0: just after initialisation.



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



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



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

Fig. 4. Illustration of the partial-sweep propagator on an OBDD of ϕ_e from Example 2.1.

change in an iteration. The `Reachable`, `FreeIn` and `FreeOut` counters are not shown in the figure. Suppose we have to find a strategy σ, such that $P(\phi_e \mid \sigma) > 0.2$ holds.

Fig. 4a shows the state of this OBDD right after initialisation, with current partial strategy $\sigma' = \varnothing$. The partial derivatives are: $\partial f/\partial d_a = 0$, $\partial f/\partial d_b = 0$, $\partial f/\partial d_c = 0$ and $\partial f/\partial d_e = 0.7$. Since Equation (10) 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 Fig. 4b. 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_{ce} becomes unreachable. While its `Reachable` counter was 2 in Fig. 4a, now it equals 0. Similarly, the `FreeIn` counters of all nodes labelled with t_{ce} or d_c become 0.

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

In Fig. 4c 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 $P(\phi_e \mid \{d_b = d_c = \bot\}) = P(\phi_e \mid \{d_c = \bot\}) - {}^{\partial f}/_{\partial d_b} = 1 - 0 = 1$. We compute the new partial derivatives: ${}^{\partial f}/_{\partial d_a} = 0$ and ${}^{\partial f}/_{\partial d_e} = 0.7576$. Again, this does not give us reason to fix any remaining decision variables to *true*.

Next, we branch on $d_e := \bot$ (see Fig. 4d). There are no active nodes below those nodes labelled with d_e (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: $P(\phi_e \mid \{d_b = d_c = d_e = \bot\}) = P(\phi_e \mid \{d_b = d_c = \bot\}) - {}^{\partial f}/_{\partial d_e} = 1 - 0.7576 = 0.2424$.

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 $P(\phi_e \mid \{d_b = d_c = d_e = \bot\}) - {}^{\partial f}/_{\partial d_a} = (0.2424 - 0.2424) < 0.2$. We therefore fix $d_a = \top$ and conclude that $\sigma = \{d_a = \top, d_b = d_c = d_e = \bot\}$ is a solution to the constraint $P(\phi_e \mid \sigma) > 0.2$, and one with value $P(\phi_e \mid \sigma) = 0.2424$.

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 2.1 and 2.2.

Finally, we address observation 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 heuristics are static during the search and depend on the variable order underlying the OBDD. We also implement two regret-based [15] 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 τ is the depth of the search tree. This is less than that of the GAC-guaranteeing decomposition method from Section 4.2.

6. Problem settings and data sets

In this section, we introduce the problem types that we consider in the experiments in Sections 7 and 9. Then we describe the data on which we formulate these problems. We make a distinction between the data used for the experiments in Section 7, which aim to provide insight in how a number of different algorithms compare, and those used for the experiments in Section 9, which focus on automated algorithm configuration.

6.1. Problem types

We describe the problem settings on which we have evaluated the solving methods described in Sections 4 and 5. All of the following problem settings involve probability distributions that are monotonic.

Theory compression or graph sparsification. We take this setting from the literature [24]. The problem is the following: given a probabilistic network, a set of pairs of vertices that are of interest, and a maximum number of edges k, extract a subset of the given network, induced on at most k edges, that models the strength of the interaction between the vertex pairs of interest in the original network as well as possible. The goal is to obtain a sparser network that preserves the pairwise interactions we are most interested in. Here, we associate a decision variable *and* a stochastic variable with each edge in the network. For details, we refer to our earlier work [43].

Spread of influence. This is the problem setting described in Example 2.1, and is known from the data mining literature [26, 41]. For our experiments, we relax some of the simplifying assumptions made in Example 2.1. In particular, we set the probability that a person turns into a customer when they receive a free product sample to 0.2. Similarly, if an existing customer influences a person, this person has a probability of 0.2 to turn into a customer themselves. We also apply this setting to the spreading of ideas, research interests or even research styles within a scientific community, by means of citation. In this problem setting, we associate the decision variables with the vertices of the network, and stochastic variables with both vertices and edges. For more details, we refer the reader to our earlier work [43].

Power grid reliability. This is the problem described in Example 2.2; we note that it is somewhat similar to the *theory compression* or *sparsification* problem described above. Again, we associate stochastic variables and decision variables with the edges of the network.

However, in this problem we are not given a set of paired vertices, but two sets of vertices, the power sources and the power consumers. These vertices are not paired. We wish to maximise the expected number of consumers that can

Table 1
Some characteristics of the test instances we use in Section 7. In particular: what entities the decision variables are associated with, the OBDD size without minimisation ($|\text{OBDD}_{nm}|$) and with dynamic minimisation ($|\text{OBDD}_{dm}|$) [63] during the compilation, the difference in compilation times Δt for these two compilation methods, the size of the set of interest $|\Phi|$, and the number $|T|$ ($|X|$) of stochastic (decision) variables.

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

be reached from at least one of the producers. Moreover, whereas in the sparsification problem setting a variable that represents an edge to *false* is interpreted as removing that edge from the graph, in the power grid reliability problem the connection probability modelled by the edge becomes lower, but not zero, when the corresponding variable is set to *false*. Finally, the graphs in the power grid reliability problem are undirected rather than directed.

Top fake news distributors. To investigate the interaction of the stochastic constraint with constraints other than cardinality constraints, we also consider a *frequent itemset mining (FIM)* problem, based on the spread of influence problem as described above. A challenging problem of our times is the spread of fake news. Oftentimes, fake news is released into specific 'bubbles', where it can then spread. It may therefore be interesting to identify not necessarily which fake news distributors are most influential, but which fake news distributors are most influential *to the same set of people*. We can model this as a FIM problem as follows.

Given a social network, we aim to enumerate *all* sets of users $U \subseteq V$ for which the following holds. First, the selected users U are influential, directly or indirectly, as determined by spread of influence: $\sum_{i \in V} P(\phi_i \mid \sigma_U) \geq \theta$, where ϕ_i represents the event that a user i believes or adopts a piece of fake news, and σ_U represents the 'strategy' in which all users in U are considered to be the initial distributors of that news. In other words: we require the *collective* influence of the users in U to be at least θ. Second, the selected users all directly influence the same large group of other users: with each set of users U we can associate another set $W \subseteq V$ of users of size at least κ, such that there is an edge (u, w) in the network for each user $u \in U$ and for each user $w \in W$, meaning that u directly tries to influence w.

This second constraint corresponds to a minimum support constraint over a transaction database in FIM (see, *e.g.*, [1]). We create this *transaction database* by putting in it one transaction τ per user $i \in V$. Here, τ represents the set of other users who influence v directly. We then aim to identify the sets of users U (itemsets) such that $U \subseteq \tau$ for at least κ individual transactions τ (making them frequent). Hence, we combine the stochastic constraint from Equation (1) with a minimum support constraint, known from the FIM literature [67].

6.2. Individual problem instances

The problem instances used for our initial experiments in Section 7 are shown in Table 1. These experiments are focused on showing the performance of our methods on a large range of different problems. We selected five problem instances from the literature. In particular, *spine16*, *spine27a*, *spine27b* are *theory compression* or *sparsification* problems, formulated on directed DNA-protein interaction networks [52].[2] Similarly, we take communities from the high energy physics collaboration undirected network [50] (*hep-th47* and *hep-th5*) and formulate a *spread of influence* problem [26,41] on them. We refer the reader to the literature [43] for a more detailed description of these five problem instances.

To expand our range of test cases to graphs of different sizes and complexities, we created additional instances for the spread of influence and power grid reliability problems. A number of these instances are used in our initial comparison and are included in Table 1.

The *facebook* instances are created by extracting communities of different sizes using the Louvain algorithm [8] from Facebook data [71]. We converted these communities to probabilistic networks using the *independent cascade model* for spread of influence [26,41]. In particular, when a user u posts n_{uv} messages on the wall of another user v, we collapse the corresponding n_{uv} parallel edges into one edge with weight $1 - (1 - p)^{n_{uv}}$, with $p = 0.1$. This weight represents the probability that user u influences user v. As described in Section 6.1, we associate two stochastic variables, each with weight 0.2, with each vertex in the network. Note that these networks are directed. In *facebook25*, all people in the network are in our set of interest. In *facebook12*, we have selected 50% of the people in the network uniformly at random for the set of interest.

[2] The dataset *spine27a* in this work corresponds to *spine27* in [43]. *spine27b* is a problem on the same network, but formulated on a different set of interest.

Table 2
Characteristics of our set of 52 test instances, including their range of size of the set of interest $|\Phi|$, number of stochastic variables $|T|$, number of decision variables $|X|$ and the number of test instances of each type.

name	problem type	$\|\Phi\|$	$\|T\|$	$\|X\|$	# 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

Table 3
Summary of characteristics of the benchmark sets we use in Section 9.4. We provide the range of sizes of the set of interest $|\Phi|$, numbers of stochastic variables $|T|$, numbers of decision variables $|X|$, OBDD sizes |OBDD| and the sizes of the training and test sets.

name	problem type	$\|\Phi\|$	$\|T\|$	$\|X\|$	# train	# test
facebook	spread of influence	15–30	16–107	15–30	412	411
high-voltage	power grid reliability	6–39	30–300	15–150	51	50

The *power grid reliability* instances are based on network models of the European and North-American *high-voltage* power grids [73], extracted using `GridKit`.[3] We have taken connected components of networks in specific geographical areas (countries in Europe and states in North America) that contain at least one power producer and at least one power consumer as our test networks. Probabilities on the edges are as presented in Example 2.2. These networks are all undirected. Here, the set of interest consists of all power consumers in the network, and we wish to maximise the expected number of power consumers that continue to receive power.

To complete each problem instance, we need to associate with each of them an upper bound on the cardinality of the solution k. In our experiments in Section 7, we run each example for nine values of k, based on the number of decision variables in the problem instance. For the spread of influence problem, k represents an upper bound on the number of people to whom we can give a free sample of the product. For the sparsification problem, k represents an upper bound on the size of the network that we extract. Finally, for the power grid reliability problem, we make the simplifying assumption that the cost of reinforcing power lines is uniform, such that we can replace the budget b by an upper bound on the number of power lines we can reinforce, k.

We used a total of 52 instances to evaluate our method in more detail in Section 7; these are summarised in Table 2.

For our experiments on the 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 E and different minimum support thresholds κ 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 20 000 to 2.5 million nodes.

6.3. Algorithm configuration data sets

For automated algorithm configuration, we require an even larger set of instances. This is because we need disjoint training and testing of sufficient size for the configurator to learn from different instances (training) and then validate its performance on a sufficiently varied set of instances (testing).

We created these instances using the process described in the previous section and summarise them in Table 3. Here, *facebook* refers to a set of spread of influence problem instances, generated from Facebook [71] data, by extracting communities and adding weights, as described in Section 6.2. We select all vertices in a problem as our set of interest. We choose the upper bound on the cardinality of the solution to be constant in these examples. Specifically, we use $k = 10$, because it can be expected to yield challenging problems, as seen in our results in Section 7. Additionally, fixing this threshold to one value, even for problems with different sizes, is a realistic choice for real-life applications in this setting. After all, companies likely have a marketing budget that does not depend very directly on the size of the social network data they have access to.

Similarly, we generated a set of power grid reliability problem instances from the European and North-American high-voltage transmission grids [73], also as described above. For these examples, we use $k = {}^{|X|}/_{2}$, such that we can reinforce at most half of the total number of power lines in any given problem instance. We believe this to be realistic for real-life applications, assuming the maintenance budgets for power grids to be roughly proportional to their sizes.

7. Experimental evaluation of OBDD-based SCMD solving methods

We experimentally evaluated the performance of CP-based and MIP-based OBDD decomposition methods (described in Section 4), as well as the full-sweep and partial-sweep global SCMD propagators on OBDDs (described in Sections 5.3

[3] Available from https://github.com/bdw/GridKit.

and 5.4). In our experiments, we used the problem instances described in Section 6.2. 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 7.2, 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 7.3.

7.1. Research questions

The experiments in this section 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 5.4) 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?

7.2. Experimental setup

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

7.2.1. Hardware and software

For modelling the probability distributions, we used a `SC-ProbLog` [43] version based on `ProbLog 2.1` [28], running in `Python 3.6.9`.[4] We used the `Cython` binding of the `dd 0.5.4` library to `CUDD 3.0.0` [68] for OBDD compilation, and used its implementation of the Sifting algorithm [63] for dynamic minimisation.[5] We implemented the MIP-based decomposition method by modelling and solving it with `Gurobi 9.0.0`, because it is freely available to academics and provides a convenient modelling interface through `gurobipy`.[6] 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 in industry.[7] We implemented the global OBDD propagators proposed in Sections 5.3 and 5.4 in the `Scala 2.12` library `OscaR 4.0.0` [51]. This library contains a state-of-the-art implementation of the `CoverSize` constraint [67], which we needed to answer question Q6.[8] Since OscaR does not support floating point variables, we could not implement the decomposition methods in OscaR.

Our experiments in Section 7.3 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 40 MB, running at 2.10 GHz under CentOS Linux 7.7.1908. Unless indicated otherwise, all experiments in Section 7.3 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.

7.2.2. Overview of pipelines

We briefly outline the different pipelines evaluated in this section. They all start with modelling the problem in SC-ProbLog [25,43] and grounding the resulting logical program into propositional formulae ϕ_i on Boolean decision variables and Boolean stochastic variables (see examples in Section 2.2), on which we impose the stochastic constraint of Equation (1). In the experiments in this section, we only evaluated the solving part of the pipelines, while later, in Section 9, we also take into account the knowledge compilation step.

CP-based decomposition. In Section 4, 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 (Section 4.2) [43]. We therefore refer to this pipeline as *no GAC CP decomposition.* In Section 4.2, 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 evaluated an OBDD variant of the MIP-based pipeline described in our previous work [43]. The *OBDD-to-MIP* pipeline converts

[4] Available at https://github.com/ML-KULeuven/problog/tree/sc-problog.
[5] Available at https://pypi.org/project/dd.
[6] Available at https://www.gurobi.com.
[7] Available at https://www.gecode.org.
[8] Available at https://sites.uclouvain.be/cp4dm/fim.

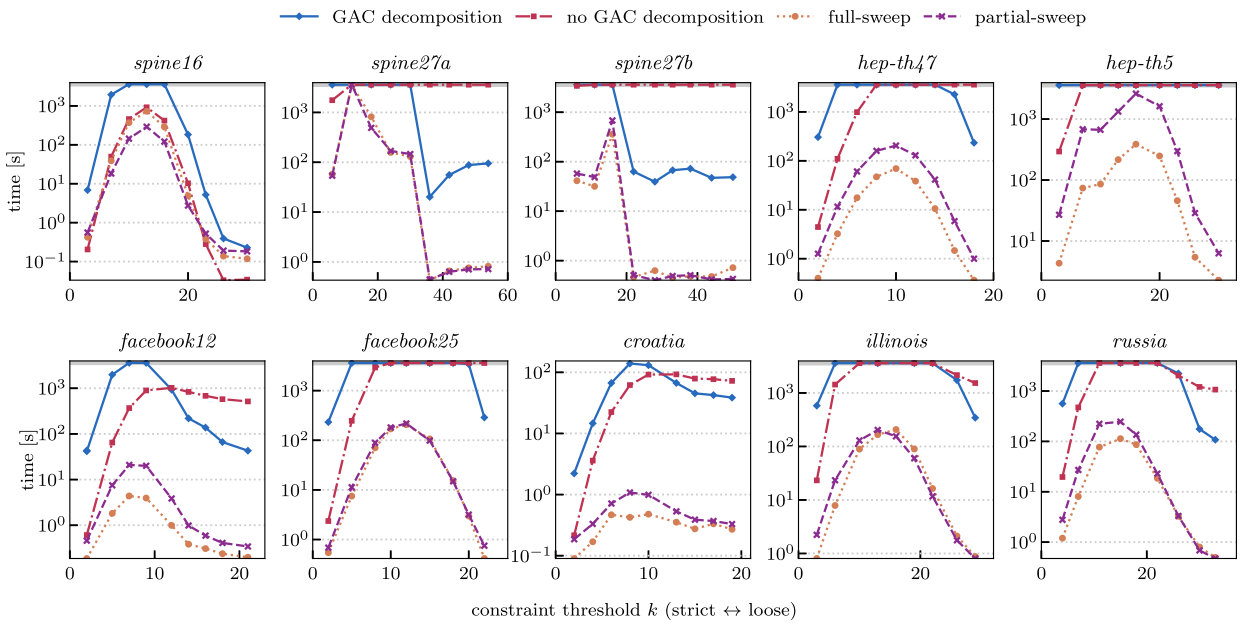


Fig. 5. Solving times of CP-decomposition methods and global SCMD methods. Cutoff time is 3 600 s (1 hour). Vertical axes are log scale. (For interpretation of the colours in the figure(s), the reader is referred to the web version of this article.)

Table 4
PAR10 values in seconds for six branching heuristics used by the full-sweep propagation algorithm on 52 test instances. Cutoff time is 3 600s.

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

the propositional formulae ϕ_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 evaluated the two variants of the new global SCMD propagator on probability distributions represented by OBDDs, implemented in OscaR: *full-sweep* (Section 5.3) and *partial-sweep* (Section 5.4).

7.2.3. Parameter settings

Unless indicated otherwise, we used the default settings for all software in the experiments presented in Section 7.3. In the experiments to answer Q1, we constrained both CP solvers to branch on the variables in lexicographical order, branching first on *false* and then on *true*. In doing so, we fixed 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 used the branching heuristic *Derivative-1* (Section 5.4), because it seems to outperform the other branching heuristics, as is shown in Table 4.

7.3. Results

We study how the decomposition methods (Section 4) compare to the global SCMD propagators (Sections 5.3 and 5.4) in terms of solving time, which we measured by using the wall-clock time reported by the different solvers as the time actually spent on solving (and not on I/O).

Comparison of CP solvers. We address Q1 by comparing the solver search times of the implementations of the full-sweep (Section 5.3) and partial-sweep (Section 5.4) versions of our propagator with two decomposed approaches in Gecode: the GAC CP composition method and the no GAC CP decomposition method (Section 7.2.2). We kept 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 ran these propagators on OBDDs that were obtained using dynamic minimisation. Fig. 5 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.

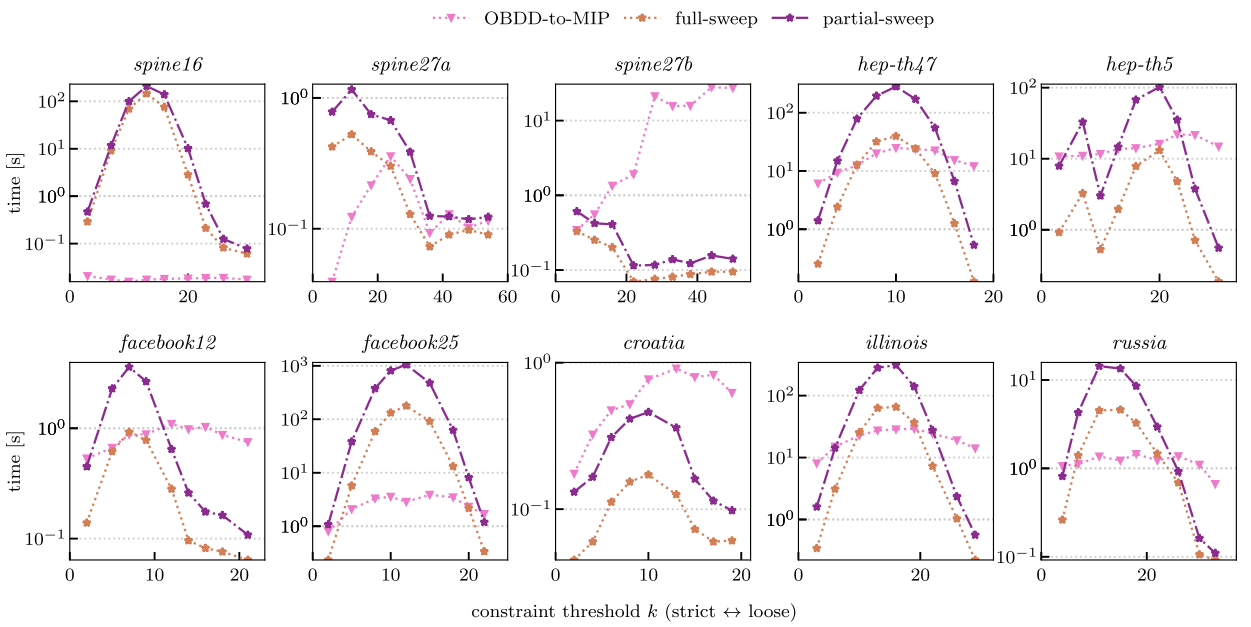


Fig. 6. Solving times of MIP-based OBDD decomposition method, compared to the full-sweep and partial-sweep methods. Cutoff time is 3 600 s (1 hour). Vertical axes are log scale.

Branching heuristics. To answer Q2, we evaluated the performance of the six branching heuristics described in Section 5.4. We ran the full-sweep propagation algorithm on our set of 52 instances described in Table 2, using an upper bound of $k = \lfloor |X|/2 \rfloor$ on the cardinality of the solution, where $|X|$ denotes the number of decision variables of the given instance. We repeated this for the six branching heuristics from Section 5.4, using a cutoff time of 3 600 s, and computed the PAR10 (penalised average run time with penalty factor 10). We present the results in Table 4. 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. Fig. 6 compares the performance of the full- and partial-sweep SCMD propagators to that observed for the OBDD decomposition method using Gurobi for solving the problem (OBDD-to-MIP). For the global propagators, we used branching heuristic *Derivative-1*. We observe that the global propagators perform comparably, and often complementarily, to the OBDD-to-MIP method, answering Q3.

Scaling. We address Q4 in Fig. 7, 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 iteration. This renders the partial-sweep algorithm potentially more efficient than the full-sweep version, especially for branching strategies that aim to reduce the size of the active part of the OBDD. While this comes at the price of some overhead, we expect the benefits of traversing a smaller part of the OBDD to outweigh the costs as OBDD size increases.

To answer Q5, we therefore ran both propagators on the dynamically minimised and non-minimised OBDDs of all 52 test instances from Table 2. 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*. Fig. 8 and Table 5 summarise our results.

Looking at the left plot in Fig. 8, 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 5, 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 Fig. 8 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 5: on the full set of OBDDs,

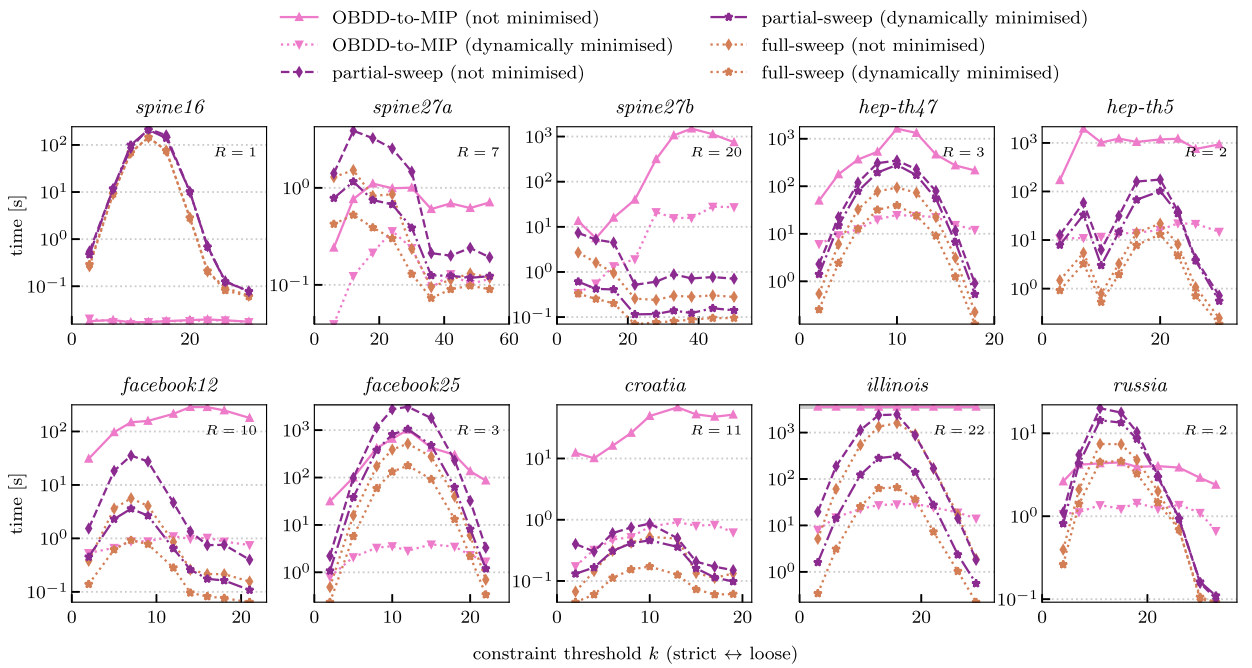


Fig. 7. Solving times of OBDD-to-MIP and global SCMD propagators on OBDDs of different sizes: either obtained without minimisation, or by using dynamic minimisation during compilation. Cutoff time is 3 600 s (1 hour). Vertical axes are log scale. R indicates the size ratio $|\text{OBDD}_{\text{nm}}|/|\text{OBDD}_{\text{dm}}|$.

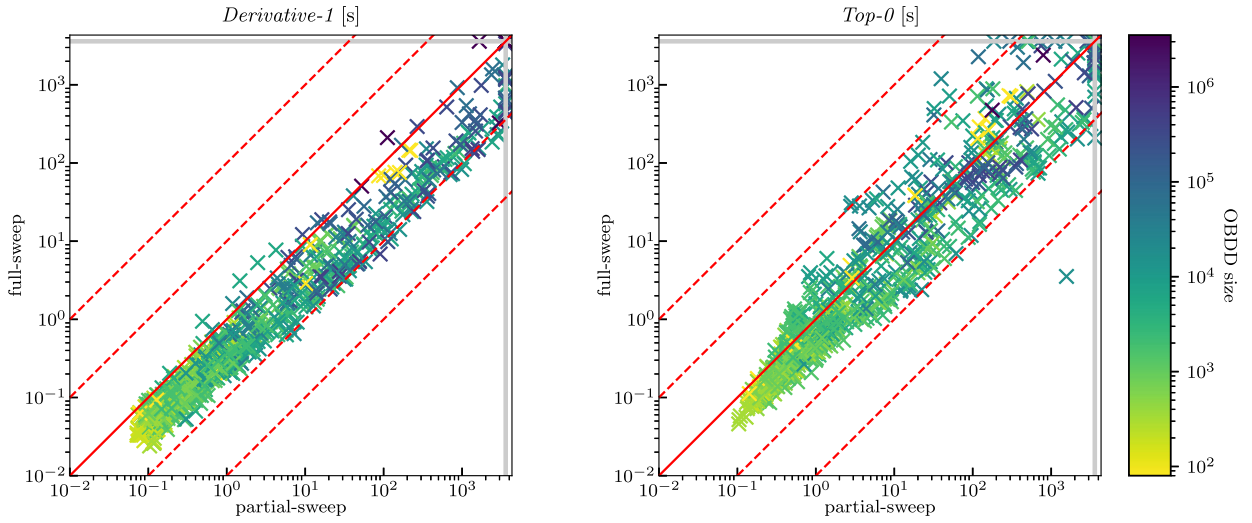


Fig. 8. Solving times of the full- and partial-sweep SCMD propagators on dynamically minimised and non-minimised OBDDs, for 52 instances (Table 2). We compare two branching heuristics: *Derivative-1* and *Top-0*. Cutoff time is 3 600 s (1 hour).

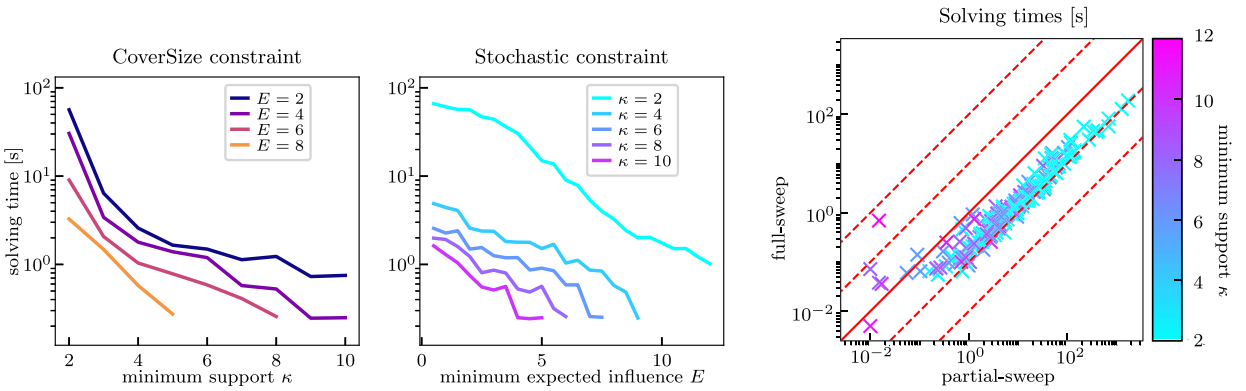
the PAR10 value of partial-sweep is now only 1.2 times higher than that of full-sweep. Again, partial-sweep has a smaller PAR10 value than full-sweep on the largest 5% of OBDDs.

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 propaga-

Table 5
PAR10 values (cutoff time of 3 600 s) for full-sweep and partial-sweep SCMD propagators on dynamically minimised OBDDs and OBDDs that are not minimised, for 52 instances (Table 2), 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 how many times out of the total number of instances a solver timed out.

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



(a) Solving times as a function of minimum support threshold κ and minimum expected influence threshold E, for a typical example with the following characteristics: $|\Phi| = 65$, $|X| = 65$, $|T| = 225$, $|OBDD| = 493\,241$, and 52 transactions in the database.

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

Fig. 9. Experimental results on the top fake news distributors problem setting (Section 6). (For interpretation of the colours in the figure(s), the reader is referred to the web version of this article.)

tor 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 κ and minimum expected influence thresholds E. We present the results for a typical example problem instance in Fig. 9a, which shows the running time for the full-sweep propagator with branching heuristic *Derivative-1* for different combinations of E and κ. Lower values of E and κ correspond to looser constraints. As expected, we see that solving times decrease as the constraints become stricter.

In Fig. 9b, 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 (κ, E) combinations. The colour indicates the minimum support threshold κ. We observe that the full-sweep propagator outperforms the partial-sweep version on almost all instances. For large values of κ, 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 κ, 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.

8. Solving SCMDs more efficiently using programming by optimisation

As the results in Section 7 show, different variants of solving methods behave differently on different problem instances. Based on this, we cannot decide what the optimal configuration is for each pipeline, or accurately predict how these or alternative configurations will behave on new problem types. Additionally, we largely relied on default parameter settings, with some minimal exploration of alternatives. Since generic solvers like Gurobi have many parameters, their defaults are unlikely to be optimal for a specific type of problem. While this might give us an indication of how well the decomposition method works 'out of the box', to assess its true potential, we need to tune its parameters. Finally, by learning which parameter settings yield shorter solving times for specific problems, we may also learn more about those problems and how to solve them more efficiently, potentially sparking interesting ideas for future research.

To address these observations, we leverage the *programming by optimisation* (PbO) paradigm [36]. We provide a brief introduction to this concept in Section 8.1, and then briefly discuss *automated algorithm configuration* (AAC) [35], which critically enables PbO. Finally, we describe how we have applied the PbO paradigm to different elements of the pipelines described in Section 7.2.2.

8.1. Programming by optimisation

During algorithm or software development, there are typically multiple ways of achieving various subtasks. For example, for solving SCPMDs, we could either use the full-sweep (Section 5.3) or partial-sweep (Section 5.4) global SCMD propagation algorithm. However, often only one of these *design choices* is implemented in the final version of an algorithm or software system. The choice is often made based on limited experimentation, with a specific application in mind. These *design choices* have no effect on correctness, but can affect performance, especially for computationally challenging problems, such as SCPs.

The paradigm of PbO introduces a different approach to design choices. When taking a PbO-based approach to software or algorithm design, developers implement multiple alternatives for many elements or components of their software. They provide the end user with the choice between these options, by exposing them as configurable parameters. Consequently, developers focus on exploring alternatives for design choices instead of determining the best instantiations for specific applications. Moreover, expanding the design space of a given software system and exposing choices as configurable parameters provides the basis for using automated algorithm configuration techniques for performance optimisation. Hence, the existence of effective AAC procedures critically enables PbO-based algorithm design.

8.2. Automated algorithm configuration

PbO-based algorithm design results in algorithms that come with a set of parameters. The parameter settings of such an algorithm (its *configuration*) can have a substantial impact on its performance, and the optimal choice may vary for different sets of problem instances. This also applies to many state-of-the-art algorithms that naturally come with many parameters. Using suitable parameter settings is then critical for reaching state-of-the-art performance — even more so for NP-hard problems, such as the ones studied in this work.

We use AAC to automatically find an optimised configuration for a problem set. After applying AAC to a target algorithm A with parameters $q_1, \ldots, q_n$ on a set of problem instances I, we obtain a configuration c^* that is expected to perform well, according to a given performance metric m, on new instances that are similar to those in I [35].

8.3. Automated algorithm configurators

There are several state-of-the-art configurators available [2,4,40]. In particular, SMAC [40] and GGA++ [2] are *model-based*: They build a model that captures the dependency of the performance of the target algorithm on its configuration. This model is used to predict the performance of configurations on multiple instances and to select promising candidate configurations. It supports different types of parameters, including conditional parameters, whose activation depends on the values assigned to other parameters.

8.4. Design space of SCP pipelines

In Section 7.2.2 we described three SCP solving pipelines: a CP-based decomposition method, a MIP-based decomposition method, and a global SCMD propagation method. Each consists of a *compilation phase*, in which we compile a propositional formula $\phi(X, T)$ to an OBDD, followed by a *solving phase*, in which we encode the constraint on the compiled diagram and solve it with a CP or MIP solver. We now briefly discuss the design spaces for these phases. We refer the reader to Appendix C for more details.

Note that we differ from earlier approaches by separating 'special values' of parameters from their regular domains. Parameters may have, *e.g.*, the domain of $\mathbb{N}^+$, but are turned off or tuned automatically when they take value 0. Contrary to earlier approaches, we split these parameters into a *switch parameter* and the *normal parameter*; the former turns the latter on or off, and the latter is only configured if the switch is on.

8.4.1. Compilation phase

In Section 7, we limited ourselves to the study of CP and MIP methods that use OBDDs to encode probability distributions. However, as mentioned in Section 3, we can also use SDDs [23] to encode those probability distributions, and decompose constraints on those SDDs [43]. Since SDDs can be more succinct than OBDDs [12] and may therefore yield smaller linear programs, we also consider SDD encodings in our configuration experiments.

For OBDD compilation, we exposed a number of parameters related to minimisation in the CUDD compiler. In particular, we open up the possibility to exploit different minimisation algorithms. These can be applied either to dynamic minimisation during the OBDD compilation or after compiling the OBDD. This results in six Boolean parameters, four categorical parameters, two integer-valued parameters and one real-valued parameter.

We use the SDD minimisation algorithm proposed in our earlier work [43] as the only option for SDD compilation, as we had to make sure that the resulting decomposed constraints can be linearised for Gurobi to be able to find a solution in all cases. We exposed all other available parameters for configuration, resulting in two categorical parameters, eight Boolean parameters, five integer- and four real-valued parameters that can be tuned for the SDD compilation.

Finally, we added a Boolean parameter for the configurator to choose between an SDD encoding or an OBDD encoding. We note that this parameter can only be configured for the decomposition approach.

8.4.2. Solving phase: Gecode and Gurobi

For the pipelines that make use of Gecode and Gurobi to solve a linear program obtained by decomposing a stochastic constraint on a probability distribution, we enabled the configuration of all parameters that are relevant for the speed of solving the problem exactly. We based the choices for domains and default values on earlier work on the automated configuration of Gurobi [39] and Gecode [42]. Considering the fact that Gecode and Gurobi already offer a wide range of branching heuristics, we refrained from exploring additional heuristics for these solvers.

For the CP-based decomposition method, we also introduced a Boolean parameter `GuaranteeGAC` that indicates whether we use the GAC-guaranteeing decomposition method described in Section 4.2, or the CP-decomposition that does not guarantee GAC [43]. Since one of the goals in this work is to develop an efficient SCMD propagation algorithm that guarantees GAC and uses an OBDD for the probability distribution encoding, we consider developing a GAC-guaranteeing CP encoding of stochastic constraints on probability distributions represented by SDDs to be outside the scope of this work. Therefore, the `GuaranteeGAC` parameter was made unavailable to the SDD-to-MIP pipeline, and we only used an encoding that does not guarantee GAC.

The resulting configuration space for solving linear program encodings of SCPs with Gecode consists of 2 Boolean parameters, 3 categorical parameters, 3 integer- and 1 real-valued parameter. The configuration space of solving linear program encodings of SCPs with Gurobi consists of 11 Boolean parameters, 10 categorical parameters, 37 integer-valued parameters, 5 real-valued parameters and 49 switch parameters.

8.4.3. Solving phase: global SCMD propagator

Our experiments in Section 7 showed that branching order has an important impact on search efficiency. Because we study a variety of problems with different properties (Section 6.1), we decided to add a range of problem-specific branching heuristics to explore this result in more detail.

For the global SCMD solving algorithm, we exposed a parameter to choose between using the full- or partial-sweep algorithm. Since a good branching strategy can dramatically limit the size of the search space, all other parameters that we have introduced are directly related to choosing which variable to branch on next, and which value to assign to that variable first.

For the variable selection heuristic, we exposed the *Top*, *Bottom* and *Derivative* variable branching heuristics (Section 5.4). These heuristics are derived from the topology of the OBDD (*Top* and *Bottom*) or dynamically determined during the search (*Derivative*). We propose seven new heuristics that take a different approach: they are derived directly from the probabilistic network on which the SCPMD is defined. Two reflect topological features of the network, namely the degree of vertices, or the degrees of the two endpoints of an edge, and the betweenness centrality of vertices or edges. Another is inspired by work on social influence [11]. We also introduced four heuristics that are inspired by work on graph sparsification [45,66]. An eighth new heuristic branches on variables that are selected uniformly at random.

Some branching heuristics incur preprocessing time. The computational costs of preprocessing steps as well as the quality of the resulting heuristic may again depend on additional parameters, which we have also exposed. Finally, we have a parameter that chooses whether to always branch on *true* first, or to always branch on *false* first.

The resulting parameter space of the global SCMD propagator consists of two Boolean parameters, one categorical parameter, two integer-valued parameters and two real-valued parameters. We provide a table with details on these parameters and their values in Appendix C.

9. Automated configuration of SCMD solving pipelines

In this section, we report on experiments using AAC to determine which pipeline outperforms the others on the sets of problem instances described in Section 6.3, and to gauge how much each pipeline benefits from being automatically configured for these specific sets of problem instances. Again, we first discuss the specific research questions that we are trying to answer. Next, we provide some details on the experimental setup in Section 9.2. Finally, we analyse the results of our experiments in Section 9.3, to answer the questions.

9.1. Research questions

The experiments in this section aim to answer the following questions:

(Q7) How much can we improve the performance of the decomposition methods and the global SCMD method on different real-world problems by automatically configuring these methods for those specific instance sets?

Table 6
PAR10 values in CPU seconds for the default and optimised configurations of the three solving methods, for both the training set and the test set. We indicate in brackets the number of examples that hit our cutoff time (600 CPU s). We highlight the smallest PAR10 values on the test sets in **bold**.

	CP-decomposition		MIP-decomposition		global SCMD	
	train	**test**	**train**	**test**	**train**	**test**
facebook (412 training instances, 411 test instances)						
default	4 338 (295)	4 270 (289)	1 888 (124)	1 664 (108)	797 (52)	782 (51)
optimised	2 518 (168)	2 615 (174)	594 (39)	**627** (41)	751 (49)	682 (44)
high-voltage (51 training instances, 50 test instances)						
default	4 386 (37)	4 351 (36)	3 686 (31)	3 989 (33)	2 379 (20)	2 782 (23)
optimised	4 379 (37)	4 452 (37)	3 188 (27)	3 031 (25)	2 260 (19)	**2 669** (22)

(Q8) Which automatically configured method solves these problems best?
(Q9) What can we learn about these solvers from the configuration results?
(Q10) How do our optimised configurations generalise to harder instances of the same problem type and to instances of a different problem type?

9.2. Experimental setup

For our configuration experiments, we mostly used the software as described in Section 7.2.1. We used the `NetworkX 2.2` and `NetworKit 5.0.1` Python toolkits for computing the scores used for variable branching heuristics, as described in Section 8.4.3.[9] SDDs were compiled using a version of a package [18] we adapted to generate SDDs that can be decomposed into linear programs [43].[10]

Because of the nature of the parameters described in Section 8.4, we expect that a model-based search process for optimal configurations will yield the best results. For our configuration experiments, we chose the general-purpose configurator `SMAC v3` [40], because it is one of the best-performing configurators that are model-based and freely available.

In our experiments, we chose default values for compilation, CUDD, Gecode and Gurobi based on the literature [39,42,43] and on their own default settings. The default settings for OscaR were chosen based on the experiments in Section 7.3.

All experiments in this section were performed on *Grace* (Section 7.2).

9.3. Configuration results

To address Q7–9, we performed fifteen independent 48-hour runs of SMAC on each solving pipeline (Section 7.2.2), on the two training sets in Table 3, minimising the PAR10 (penalised average running time with penalty factor 10) and using a cutoff time of 600 CPU seconds. Then, for each method and each dataset, we evaluated the final incumbent (the configuration with the smallest PAR10 value) on the appropriate test set.

The results in Table 6 show that the MIP-decomposition method shows the largest relative improvement after configuration, which answers Q7. We explain this by noting that Gurobi has a relatively large configuration space (which gives many options for improvement), compared to Gecode and OscaR, and by noting that we used default settings for Gurobi as our default configuration, while we chose our default configuration of OscaR based on our results in Section 7.

We also observe that, even with configuration, the CP-decomposition method is not competitive with the MIP-decomposition method and global SCMD method, similar to what we see in Fig. 5. Once more, it appears that, for CP encodings of SCMDs, a global encoding is more favourable than a decomposed one. However, we see that the performances of the MIP-decomposition method and global SCMD method are comparable and complementary after configuration, similar to what we see in Fig. 6: on the Facebook instances, MIP works better, while on the high-voltage datasets, the global constraint works better; this answers Q8.

We provide the optimised configurations obtained from these experiments online at https://github.com/latower/SCPMD-solving. To answer Q9, we first note that for the CP-decomposition and MIP-decomposition pipelines, SMAC always chooses to encode the probability distributions as OBDDs, rather than SDDs. Furthermore, here and in the global SCMD propagation pipeline, SMAC tends to favour the group sifting algorithm for OBDD minimisation, which is CUDD's default minimisation algorithm. Remarkably, the optimised configurations for the *facebook* and *high-voltage* sets agree on all parameter choices for OscaR: SMAC chooses to use the full-sweep version of the propagator, combined with the *Derivative-1* branching heuristic. We believe that further, detailed analysis of these and similar results of configuration experiments could provide useful directions for improvements to SCMD solving pipelines and see this as a promising direction for future work.

Finally, we note that the improvement in running time on the *high-voltage* benchmark set is less impressive (and even negative, in the case of CP-decomposition) than on the *facebook* benchmark set. We explain this by noting that the *high-*

[9] Available at https://networkx.github.io and https://networkit.github.io.
[10] Available at http://reasoning.cs.ucla.edu/sdd/ and https://github.com/ML-KULeuven/problog/tree/sc-problog.

Table 7
PAR10 values in CPU seconds for the default and optimised configurations on two test sets. We indicate in brackets the total number of (problem, k) combinations in the first column. In the other columns, we indicate in brackets how many of those combinations reached the cutoff time of 3 600 CPU seconds. For each problem set, we highlight the lowest PAR10 value for the optimised configurations in **bold**.

(a) Results for the problems in the benchmark sets in Table 3, that were not solved by any of the solvers in the configuration experiment of Section 9.3.

		CP-decomposition	MIP-decomposition	global SCMD
facebook (558)	default	35 398 (548)	28 780 (441)	11 330 (168)
	optimised	32 607 (504)	18 528 (278)	**10 716** (158)
high-voltage (351)	default	34 325 (334)	33 523 (326)	29 300 (285)
	optimised	32 597 (317)	31 302 (304)	**29 186** (284)

(b) Results on the full set of 52 examples in Table 2.

		CP-decomposition	MIP-decomposition	global SCMD
spine (27)	default	12 308 (9)	569 (0)	17 (0)
	optimised	16 220 (12)	35 (0)	**17** (0)
hep-th (18)	default	30 177 (15)	6 493 (3)	65 (0)
	optimised	26 254 (13)	568 (0)	**68** (0)
facebook (99)	default	19 100 (52)	4 428 (11)	58 (0)
	optimised	15 482 (42)	791 (2)	**51** (0)
high-voltage (324)	default	8 808 (77)	8 410 (74)	55 (0)
	optimised	8 538 (75)	4 447 (39)	**52** (0)

voltage example set is much smaller than the *facebook* set (and thus has fewer examples to learn from), while the problems tend to be larger (see Table 3), causing relatively many examples to hit the cutoff time.

9.4. Generalisation of automated configuration results

We addressed Q10 by running the default and optimised configurations obtained from Section 9.3 on the examples in Table 3 that were not solved by any solver during the experiment in Section 9.3 and therefore represent the hardest instances in the training and test sets. In this new experiment, however, we used a cutoff time of 3 600 CPU seconds instead of 600. Rather than using just one threshold k per problem instance, we ran each configuration with nine different thresholds per example, like we did for the experiments in Section 7.

Similarly, we ran the optimised configuration obtained on the *facebook* dataset on the *hep-th* and *facebook* examples described in Table 2, since these are spread of influence problems. Finally, we ran the optimised configurations obtained on the *high-voltage* dataset on the *spine* and *high-voltage* examples, because of their similarity. Note that, for practical reasons, there is a small overlap (at most 5%) in the instances used for training in Section 9.3 and the *facebook* and *high-voltage* test sets we are using in this experiment. We present the results in Table 7 and observe patterns similar to those in Table 6.

From Table 7a, we see that the results for the harder examples with the larger cutoff time are very similar to the ones shown in Table 6 and conclude that our configuration results translate predictably to harder instances of the same problem type, answering part of Q10.

Table 7b shows very similar results for the *facebook* and *high-voltage* problem instances. This is unsurprising, since they are taken from the same datasets and represent the same problem types as the ones used for the configuration experiments. For the *spine* and *hep-th* examples, we see dramatic improvement in the performance of the MIP-decomposition pipeline, but not in the global SCMD pipeline, with negative results for *hep-th*. Still, the global SCMD pipeline outperforms the MIP-decomposition pipeline on these examples, for both the default and the optimised configurations. We conclude that the results obtained in Section 9.3 translate reasonably to problem instances of different types, with a small advantage for the global SCMD approach, answering the remainder of Q10.

10. Related work

Solving a stochastic constraint like the one in Equation (1) and maximising a stochastic optimisation criterion can both be seen as instances of the *stochastic satisfiability problem* (SSAT) defined in the Handbook of Satisfiability [6]. An SSAT instance is defined over a given propositional formula ϕ and a *prefix* in which each variable is either existentially or 'randomly' quantified (meaning that a variable t takes the value *true* with a given probability p_t, independently of other variables), in some order. If the existential variables come first in the prefix, *i.e.*, $\exists x_1 \cdots \exists x_{|X|} \text{R} t_1 \cdots \text{R} t_{|T|} \phi(X, T)$, we can think of the variables in X as the decision variables that are assigned a value in a strategy σ, and the variables in T as the stochastic variables. In this setting, proving that there is a σ for which Equation (1) is satisfied, is an instance of the SSAT problem.

For the case in which $X = \varnothing$ and $p_t = 1/2$ for all variables in T, the *majority SAT* (MAJSAT) problem [6,53] asks if the probability that $\text{R} t_1 \cdots \text{R} t_{|T|} \phi(X, T)$ evaluates to *true*, exceeds 1/2. The slightly more general version of this problem, *"exists"*

MAJSAT (E-MAJSAT) [46], takes as input a prefix and formula of the form $\exists x_1 \cdots \exists x_{|X|} \text{R} t_1 \cdots \text{R} t_{|T|} \phi(X, T)$ (with $x_i \in X$, $t_j \in T$, and $p_t = 1/2$ for all variables $t \in T$), and aims to find a strategy σ that assigns truth values to the variables $x \in X$ and *maximises* the probability that $\phi|_\sigma$ evaluates to *true* [6].

Finally, *functional E-MAJSAT* [57] generalises this further by allowing p_t to be an arbitrary, rational probability for each stochastic variable $t \in T$. Clearly, this last setting corresponds to maximising a stochastic optimisation criterion.

In the context of stochastic constraint *optimisation*, the problems studied in this work are also related to the *maximum a posteriori (MAP)* inference problem. In the context of propositional logic, the MAP problem can be stated as follows. Given a propositional formula $\phi(X, T)$ on decision variables X and stochastic variables T, and an assignment of truth values s to the variables in T, which assignment of truth values to the variables in X maximises the probability that $\phi(X, T)$ evaluates to *true*? This task is also known in the probabilistic inference literature as the *most probable explanation (MPE)* task [38,54] and the *maximum probability assignment (MPA)* task [9]. It has been shown that the MAP problem can be seen as a constraint optimisation task [60].

SCPs themselves are related to *chance constraint optimisation* [16] and *probabilistic CP* [69]. In particular, the problems we consider in this work can be framed as single-stage *stochastic constraint satisfaction problems* (SCSPs) [72]. Our work differs from earlier, more generic, approaches to solving these problems in that we explicitly use the structure of the DD representation of the probability distributions to speed up the solving process.

While SCPs are certainly related to constraint optimisation under *soft constraints* [7], we impose *hard constraints* on probability distributions, and thus refrain from a further discussion of soft constraints. Mixed networks essentially combine probabilistic graphical models and constraint networks [48]. Since we use a probabilistic propositional framework to represent our models, we also consider probabilistic graphical models, although conceptually somewhat related, to be outside of the scope of this discussion.

In this work, we pay special attention to a particular subclass of SCPs: SCPMDs. The main difference between SCPMDs and SCPs is that SCPMDs are defined on *monotonic* distributions. We exploit this property to optimise the constraint propagation process for SCMDs, which distinguishes our method from more general approaches [72]. While our global propagator can only be applied to monotonic distributions, it does allow us to obtain strict bounds during constraint propagation. These strict bounds can only be achieved in alternative methods [57] by constraining the underlying variable order of the decision diagrams. The variable order of a diagram determines its size, and the efficiency of constraint propagators depends on the size of the diagrams they are operating on. Therefore, having extra constraints on the variable order that limit possibilities of obtaining a sufficiently small diagram, is often disadvantageous.

Note that in the SDD decomposition method we use in the experiments of Section 9, the underlying variable order of the SDDs is also constrained, to obtain a linear program [43]. However, none of our methods that use OBDDs to represent probability distributions require any constraints on the variable order of the OBDD, and our automated configuration experiments also indicate that OBDDs are preferable over SDDs in this context.

We limited ourselves in this work to single-stage optimisation problems. In multi-stage SCPs, after a first set of decisions, the value of stochastic variables is revealed. This prompts another set of decisions to be made, after which the value of another set of stochastic variables is revealed, and so on. The goal is to either make an optimal first decision (with respect to a given objective function), before the values of the stochastic variables of the first stage are even revealed, or to develop a *policy* that allows the users to choose their decisions in the following stages, based on what unfolds as the values of the stochastic variables are revealed. Multi-stage SCPs are typically used to model planning and scheduling problems [3,47], and can be modelled as special cases of the SSAT problem [6], where blocks of existentially quantified and randomly quantified variables alternate in the prefix of the propositional formula.

The authors of Stochastic MiniZinc [59] implemented a generic framework to encode generic multi-stage SCPs in a solver-agnostic manner. Since multi-stage problems typically do not yield SCPs on monotonic distributions, we did not study them in this work, but instead focused on leveraging the structure of single-stage SCPMDs, aiming to solve those faster.

Our approach of keeping constraints (such as a linear constraint on cardinality) separated avoids the complexity of encoding this combination of constraints into one diagram, which is the approach taken by Pipatsrisawat & Darwiche [57]. Consequently, we avoid the blow-up of the diagram and exploit the structure of these constraints, thus leveraging the power of dedicated constraint propagators. Finally, modelling constraints separately allows users to add constraints that cannot be (trivially) encoded in CNF, allowing larger expressiveness than the method by Pipatsrisawat & Darwiche [57].

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. The majority of existing methods sample scenarios from a distribution, and hence ignore such structures [34].

In the CP literature, OBDDs and the similar *multi-valued decision diagrams* (MDDs) are often used to encode all solutions for a constraint, and efficient propagation algorithms for these data structures have been developed [32,33,70]. By associating MDD arcs in such encodings with probabilities, one can sample solutions to a constraint [55]. Note that, while this data structure is similar to OBDDs, it is used to solve a fundamentally different problem than the one we address in this work.

Automated optimisation techniques and tools such as SMAC [40] have been used to solve optimisation problems in approximate probabilistic inference [58]. However, to the best of our knowledge, they have not yet been applied to the optimisation of the configuration of exact probabilistic inference methods.

11. Conclusion and outlook

We introduced two main approaches to exactly solving stochastic constraint (optimisation) problems on monotonic distributions (SCPMDs). The first is a more general, decomposition-based approach that is applicable to stochastic constraint (optimisation) problems (SCPs) on generic probability distributions and leverages either existing constraint programming (CP) or mixed-integer programming (MIP) technology [43]. The second is specifically suited for solving global stochastic constraints on monotonic distributions (SCMDs). It operates on ordered binary decision diagram (OBDD) encodings of probability distributions, and we have implemented it in the CP solver OscaR [51].

In this work we demonstrated the benefits and drawbacks of both methods. In summary: the decomposition method is applicable to a wider range of problems, but less efficient in its traversal of the search space than the global SCMD constraint, because it does not guarantee generalised arc consistency (GAC). We showed that a straightforward modification of this method that does guarantee GAC, does not significantly improve solving speed.

We gave an extensive description of two implementations of the 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 pass through the OBDD.

In an initial set of experiments on a set of 52 problem instances 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 approach, we found that the approaches are complementary, with none of the approaches consistently outperforming the other. Small trends can however be observed.

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 the global SCMD propagation and MIP methods.

For a fairer comparison, we applied programming by optimisation (PbO) [36] to three solving pipelines: a CP-decomposition pipeline, a MIP-decomposition pipeline and a global SCMD pipeline. We implemented different design choices for the compilation phase of these pipelines, and for their solving phases. We opened up as many parameters as possible for automated configuration and used SMAC [40] to optimise each of these pipelines on benchmarks from two different domains. This approach ensured that we could compare methods once their configurations were optimised for the specific problem settings that we tested on.

These experiments demonstrated that both the default and optimised configurations generalise to larger and different types of problems. Both in the default and optimised configurations, the global SCMD approach outperforms both the CP-decomposition method and the MIP-decomposition method on these more complex problem sets in terms of solving time.

We also showed how our SCMD propagator can effectively be combined with other constraints, such as linear constraints and a CoverSize constraint.

We presented a number of ideas that can be useful in other contexts as well:

- The idea of solving SCPs by means of a modular approach that decouples knowledge compilation from search.
- The creation of global constraints to search more efficiently over distributions, by guaranteeing GAC.
- The idea of applying algorithm configuration to simultaneously configure different stages of a pipeline of rather complex tools.

A number of 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 such an approach would be the development of a propagation algorithm that only implements the priority queues that address observations O1–O3 from Section 5.4. These require little overhead, but may already benefit from shrinking the active part of the OBDD that needs to be sweeped during propagation.

In this work, 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 scheduling and vehicle routing problems. Here we can exploit the fact that it is easy to combine our constraint with other constraints in CP.

We applied automated algorithm configuration in the current study with a focus on running time minimisation. Other criteria can be of interest as well. For example, the memory use of knowledge compilers can be prohibitively large, so optimising solving methods to use less memory could increase the applicability of those methods. Furthermore, to the best of our knowledge, this work represents the first use of automated algorithm configuration in exact probabilistic inference. The configuration results we presented in this work encourage us to expect automated solver configuration to also be beneficial for optimisation solvers for other exact probabilistic inference tasks than the ones discussed in this work.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

We thank Hélène Verhaeghe for her input and suggestions. We thank Roger Paredes and Leonardo Dueñas-Osorio for their advice regarding the formulation of the power grid reliability problem and the interpretation of the high-voltage data.

Funding: This work was supported by the Netherlands Organisation for Scientific Research (NWO), and by TAILOR, a project funded by EU Horizon 2020 research and innovation programme under GA No 952215. Behrouz Babaki was supported by a postdoctoral scholarship from IVADO through the Canada First Research Excellence Fund (CFREF) grant.

Appendix A. Monotonic distributions and locally monotonic OBDDs

Some monotonic distributions can be encoded into OBDDs that exhibit local non-monotonic behaviour. Here is an example of such a distribution.

Example A.1 *(An OBDD that is not locally monotonic).* Recall the power grid reliability example of Example 2.4, and let us focus on encoding the survival probability of a single power line. The decision whether to reinforce this power line is denoted by Boolean decision variable d. Following the notation used in Example 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', respectively, which take the value $\top$ if the line survives and the value $\bot$ if it does not. We then encode the event ϕ_1 of survival of this power line with the following formula:

$$\phi_1 = (d \wedge t) \vee \left(\neg d \wedge t'\right). \tag{A.1}$$

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 $P(\phi_1 \mid d = \top) = p$. Similarly, we can show that $P(\phi_1 \mid d = \bot) = p'$. 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 2.1.

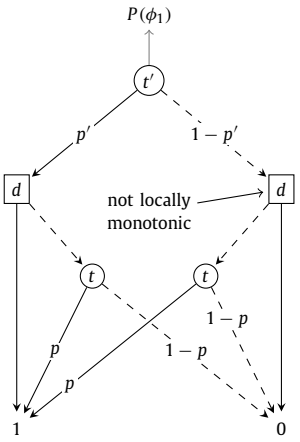
An example OBDD encoding of this formula is shown in Fig. A.10a. Consider the decision node on the right. If we set $d := \bot$, the score of that node is p. However, if we set $d := \top$, then the score in that node is 0. Consequently, flipping the value of d from $\bot$ to $\top$ is not locally monotonic (see Definition 3.1), 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 ϕ_1 by fixing $d := \top$, because of the second clause.

This becomes relevant in the partial-sweep version of the global propagation algorithm. Here, it can happen that, *e.g.*, the left decision node in Fig. A.10a becomes inactive, and we have to compute the partial derivative with only the right decision node. The local non-monotonic behaviour may cause us to compute a negative partial derivative (Equation (14)), which causes us to no longer be able to guarantee GAC, and to possibly compute an incorrect score for the optimal solution. The following example shows how we can construct a monotonic probability distribution that does yield a locally monotonic OBDD.

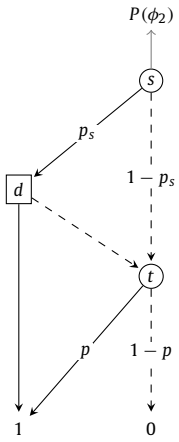
Example A.2 *(A locally monotonic OBDD).* Taking the same example of computing the survival probability of a single power line, consider this alternative encoding:

$$\phi_2 = (d \wedge s) \vee t, \tag{A.2}$$

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. Note how Equation (A.2) does not contain any negations. 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 $P(\phi_2 \mid d = \top) = {}^{(p'-p)}/_{(1-p)} \cdot (1-p) + p \cdot {}^{(p'-p)}/_{(1-p)} + (1 - {}^{(p'-p)}/_{(1-p)}) \cdot p =$



(a) An OBDD representation of $\phi_1 = (d \wedge t') \vee (\neg d \wedge t)$, with variable order $t' \prec d \prec t$. Probabilities are such that $0 < p < p' < 1$.



(b) An OBDD representation of $\phi_2 = (d \wedge s) \vee t$, with variable order $s \prec d \prec t$. Here, $p_s = {}^{(p'-p)}/_{(1-p)}$, and $0 < p < p_s < p' < 1$.

Fig. A.10. Two different OBDD representations of the same probability distribution.

$(p' - p) + p \cdot ({}^{(p'-p)}/_{(1-p)} + 1 - {}^{(p'-p)}/_{(1-p)}) = p' - p + p = p'$. Similarly, we can show that $P(\phi_2 \mid d = \bot) = p$. Again, since $p' > p$ by assumption, this distribution is monotonic, according to Definition 2.1.

An example of an OBDD encoding of this formula is shown in Fig. A.10b. Note that the decision node in this OBDD displays locally monotonic behaviour, and thus the OBDD itself is locally monotonic, according to Definition 3.1.

Appendix B. Pseudocode of the partial-sweep propagation algorithm

Because the pseudo code for our partial-sweep SCMD propagation algorithm is too lengthy to include in the main part of this paper, we provide it in this appendix.

Note that OscaR [51] uses *reversible data structures* that provide very convenient support for backtracking. We do not include any 'undo' operations for backtracking in our algorithm, as those mechanisms are already provided by the reversible data structures implemented in OscaR.

B.1. Notation and terminology

We use r to refer to a node in the OBDD, and r^- and r^+ to its lo and hi child, respectively. We use $var(r)$ to indicate *variable* that labels a node r, and we use $w(r)$ to indicate its weight in case $var(r)$ is a stochastic variable. The path weight of r is denoted by $\pi(r)$, and its score according to Equation (5) by $s(r)$.

We assume that the nodes of the OBDD are indexed in a topological way, such that any path from a root to a leaf corresponds to a series of increasing indices. In most of the top-down and bottom-up sweep algorithms we use queues to limit the number of nodes we visit during the sweep. In our pseudo code, a queue corresponding to a downward sweep is represented by $\mathcal{Q}$ (such that elements in the queue are sorted in increasing order of OBDD node index), while a queue used for an upward sweep is denoted with $\mathcal{U}$ (with elements in the queue sorted in decreasing order of OBDD node index). Note that we treat these queues as sets: they only contain unique elements.

We often iterate over OBDD nodes that are labelled with a particular decision variable d. We denote this set of particular decision nodes with OBDD_d.

For compactness, we refer to a node labelled with a stochastic variable as a *stochastic node*. We use similar shorthands for *free* or *unbound decision nodes*, *bound decision nodes*, *true decision nodes* and *false decision nodes*.

In the case of decision nodes, we define the *active child* of a node as follows. A child of a decision node is active if it is the hi child of a free or true decision node, or if it is the lo child of a false decision node (see Algorithm B.9).

We think of propagation as the act of *removing* outgoing arcs of decision nodes when we fix the corresponding decision variable (recall Fig. 4 in Section 5.4). Specifically, we remove an OBDD arc (p, c) from a parent p to child c if we fix $var(p)$ to *true* and c is p's lo child, or if we fix $var(p)$ to *false* and c is p's hi child (see Algorithm B.9).

Through this process of removing arcs, we effectively remove *valid paths* (recall the definition of a valid path from Section 5.3) from the OBDD. Valid paths from OBDD roots to internal OBDD nodes, or to or from active decision nodes can determine whether or not we consider OBDD nodes to still be *relevant*, given the current partial strategy and corresponding removed arcs.

There are two ways in which a node r can be relevant. In the first case, it is a free decision node *and* it is reachable through a valid path from an OBDD root. In the second case, it is itself not a free decision node, but there is at least one

valid path from a free decision node above r in the diagram down to r *and* there is a valid path from r itself down to a free decision node below it (see Algorithm B.9).

In order to determine if a node is relevant, and to keep track of the part of the OBDD that is active (see Section 5.4), we associate three counters with each node r:

`Reachable[r]` Indicates the number of valid paths from the *artificial root* (see below) of the OBDD down to r. The counter for this artificial root itself (and for the actual OBDD roots) is always 1.

`FreeIn[r]` Indicates the number of incoming arcs that are a part of a valid path from free decision nodes above r in the OBDD. This counter can take the values 0–$|parents(r)|$.

`FreeOut[r]` Indicates the number of arcs outgoing of r that are a part of a valid path from r down to free decision nodes below r. For each node of the OBDD, this counter can take the values 0–2. For the leaves, this counter is always equal to 0.

In the general case, an OBDD may have multiple roots, each one corresponding to a query in the original SCPMD. In order to define the `Reachable` counter in our implementation, we have added an artificial root to the OBDD, with one outgoing arc to each of the original roots.

The intuition behind the `Reachable` counter is the following: during search and propagation, assignments to decision variables may disconnect part of the OBDD from the root, because we remove arcs accordingly (observation (O6) in Section 5.4). This happens for example in Fig. 4b.

The `FreeIn` counter of a node r has a value in the same domain as the `Reachable` counter, but represents a different concept. As addressed in observation (O4) in Section 5.4, only score changes in nodes that are descendants of free decision nodes can influence the scores of those decision nodes. Therefore, during the bottom-up traversal of the OBDD to update the scores (Algorithm B.4), we do not always need to propagate all the way to the roots. Once we encounter a node r whose score has changed due to recent value assignment to decision variables, but from which there is no valid path back to the artificial OBDD root that passes through a free decision node, we do not need to enqueue the parents of r for score updates. We keep track of this by counting how many of the incoming arcs of node r are on such a path.

The logic behind the `FreeOut` counter is similar to that of the `FreeIn` counter. However: instead of stopping an upward sweep, it serves to stop the downward sweep for path weight computation, to address observation (O5) in Section 5.4. The value of the `FreeOut` counter for a node r is either 0, 1 or 2, as it represent the number of children of the `FreeOut` counter that are on valid paths down to free decision nodes. Observe that if a node r is a fixed decision node, the value of its `FreeOut` counter can never exceed 1, as one of the outgoing arcs of r is removed by fixing the corresponding decision variable.

B.2. An SCPMD solving algorithm

Algorithm B.1 shows the basic steps needed for solving an SCPMD in the *maximise expectation* setting (to which both problems described in Examples 2.1 and 2.2 belong).

Recall that these problems seek to maximise an expected score. We use the SCMD for solving these problems by solving the constraint

$$\sum_{r \in \text{roots}} \rho_r P(r \mid \sigma) > \theta, \tag{B.1}$$

and, as soon as we have found a solution with score s^*, we update θ to take that value, and continue the search until we find a new solution, with a larger score.

B.3. Initialisation

Before the search for a solution to the stochastic constraint of Equation (1) begins, we initialise the data structures needed for enforcing the SCMD with the function InitialiseSCMD(OBDD, X), as given in Algorithm B.2.

B.4. Partial-sweep propagation algorithm

During the search, as more and more decision variables become fixed, we repeatedly call the PropagateSCMD function in Algorithm B.3 to recompute scores, path weights, partial derivatives and the score of the partial strategy, but also to keep track of the relevant part of the OBDD.

We first update arrays that record the current scores and path weights of the nodes in the OBDD, using the functions in Algorithms B.4 and B.5. Then, we detect currently free decision variables that must be fixed to *true* in order to obtain a score larger than the current value of θ with the EnforceDomainConsistency function in Algorithm B.6. This function also fixes these variables accordingly. Finally, we maintain the relevant part of the OBDD by updating the counters presented in Appendix B.1, using the functions in Algorithms B.7 and B.8. To increase the readability of our pseudocode, we use the helper functions specified in Algorithm B.9.

Algorithm B.1 Solving an SCPMD, in the *maximise expectation* setting.

Input: an OBDD, a set of decision variables X, a maximum cardinality k. These are all considered to be global variables.
Output: the optimal strategy σ^* and its corresponding score $s(\sigma^*)$.

procedure Branch(σ', d, a)
  $X_{\text{free}} \leftarrow X_{\text{free}} \setminus \{d\}$
  $F \leftarrow \{d\}$ ▷ The set of decision variables that are fixed in this call to the Branch function.
  $\sigma' \leftarrow \sigma' \cup \{d = a\}$ ▷ Update partial strategy.
  (conflict, σ', F) $\leftarrow$ PropagateSCMD(σ', F) ▷ See Algorithm B.3.
  if conflict **then return** and Backtrack **end if**
  (conflict, σ', F) $\leftarrow$ PropagateCardinalityConstraint(σ', F) ▷ Assumed given, outside the scope of this work.
  if conflict **then return** and Backtrack **end if**
  Solve(σ')

procedure Solve(σ')
  if $X_{\text{free}} = \varnothing$ **and** $s(\sigma') > s^*$ **then**
    $\sigma^* \leftarrow \sigma'$
    $s^* \leftarrow s(\sigma^*)$ ▷ Score is computed incrementally (see Algorithm B.3).
    UpdateSCMDthreshold(s^*)
    return and Backtrack
  for $d \in X_{\text{free}}$ **do** ▷ There are different selection strategies for determining which d to branch on next.
    $a \leftarrow$ SelectValue($dom(d)$) ▷ And different strategies for determining on which value to branch.
    Branch(σ', d, a)
    Branch(σ', d, $\overline{a}$)

InitialiseSCMD ▷ See Algorithm B.2.
InitialiseCardinalityConstraint(X, k) ▷ Assumed given, outside the scope of this work.
$X_{\text{free}} \leftarrow X$ ▷ Set of free decision variables, global variable.
$\sigma^* \leftarrow \{d = \bot \mid d \in X\}$, $s^* \leftarrow 0$ ▷ Optimal strategy and corresponding score, global variables.
$\sigma' \leftarrow$ EnforceDomainConsistency(X_{free}) ▷ Fix those variables that must be *true* to obtain partial strategy (Algorithm B.6).
Solve(σ')
return σ^*, $s(\sigma^*)$

Algorithm B.2 Initialisation of data structures. Note that OBDD and X are considered to be global variables.

procedure InitialiseFreeIn
  for $r \in$ OBDD **do** `FreeIn`$[r] \leftarrow 0$ **end for**
  for $r \in$ sorted(OBDD) **do** ▷ Downward sweep.
    if $var(r)$ is decision OR `FreeIn`$[r] > 0$ **then**
      `FreeIn`$[r^-] \leftarrow$ `FreeIn`$[r^-] + 1$
      `FreeIn`$[r^+] \leftarrow$ `FreeIn`$[r^+] + 1$

procedure InitialiseFreeOut
  for $r \in$ OBDD **do** `FreeOut`$[r] \leftarrow 0$ **end for**
  for $r \in$ reversed(sorted(OBDD)) **do** ▷ Upward sweep.
    if $var(r)$ is decision OR `FreeOut`$[r] > 0$ **then**
      for $p \in$ parents(r) **do** `FreeOut`$[p] \leftarrow$ `FreeOut`$[p] + 1$ **end for**

procedure InitialiseReachable
  for $r \in$ OBDD **do** `Reachable`$[r] \leftarrow 0$ **end for**
  `Reachable`$[root] \leftarrow 1$
  for $r \in$ sorted(OBDD) **do** ▷ Downward sweep.
    `Reachable`$[r^-] \leftarrow$ `FreeIn`$[r^-] + 1$
    `Reachable`$[r^+] \leftarrow$ `FreeIn`$[r^+] + 1$

procedure InitialiseScores
  for $r \in$ reversed(sorted(OBDD)) **do** ▷ Upward sweep.
    if $var(r)$ is decision **then**
      $s(r) \leftarrow s(r^+)$
    else
      $s(r) \leftarrow w(r) \cdot s(r^+) + (1 - w(r)) \cdot s(r^-)$

procedure InitialisePathWeights
  for $r \in$ OBDD **do** $\pi(r) \leftarrow 0$ **end for**
  for $r \in$ sorted(OBDD) **do** ▷ Downward sweep.
    if r is an original root of the OBDD **then**
      $\pi(r) \leftarrow \pi(r) + \rho_r$
    else
      for $p \in$ parents(r) **do**
        if $var(p)$ is decision **then**
          if r is hi child of p **then** $w \leftarrow 1$ **else** $w \leftarrow 0$
        else
          if r is hi child of p **then** $w \leftarrow w(p)$ **else** $w \leftarrow (1 - w(p))$
        $\pi(r) \leftarrow \pi(r) + \pi(p) \cdot w$

procedure InitialiseSCMD
  InitialiseFreeIn
  InitialiseFreeOut
  InitialiseReachable
  InitialiseScores
  InitialisePathWeights
  $\theta \leftarrow 0$ ▷ The current best score to beat.

Algorithm B.3 SCMD propagation algorithm for propagating the consequences of a given partial strategy σ'. Note that the set of currently free decision variables X_{free} is a global variable.

procedure PropagateSCMD(σ', s_{old}, F)
  $s \leftarrow s_{old}$ ▷ Score of previous partial strategy.
  $\delta \leftarrow$ UpdateScores(F) ▷ δ is sum of derivatives of decision variables that were recently fixed to *false*, see also Algorithm B.4.
  $s \leftarrow s - \delta$ ▷ score of current partial strategy σ'
  if $s \leq \theta$ **then return** $(true, \sigma', F)$ **end if** ▷ If we cannot satisfy the constraint, we must return and backtrack.
  UpdatePathWeights(F) ▷ See Algorithm B.5.
  $(\sigma', F) \leftarrow$ EnforceDomainConsistency(σ', F, s) ▷ See Algorithm B.6.
  UpdateReachableFreeIn(F) ▷ See Algorithm B.7.
  UpdateFreeOut(F) ▷ See Algorithm B.8.
  return $(false, \sigma', F)$

Algorithm B.4 Given a set F of decision variables that were recently fixed (either by branching or by propagation), update the node scores (Equation (5)) that may have changed due to these new truth assignments. See Algorithm B.9 for helper functions.

procedure UPDATESCORES(F) ▷ Upward sweep.
  $\mathcal{U} \leftarrow \{r \mid var(r) \in F \wedge var(r) = \bot \wedge \texttt{Reachable}[r] > 0\}$ ▷ Max heap (treat as set).
  $\delta \leftarrow 0$ ▷ The combined derivative for all variables that are fixed to *false* in this round.
  $s_{old} \leftarrow 0$ ▷ Old score of an OBDD node.
  while $\mathcal{U} \neq \varnothing$ **do**
    $r \leftarrow \mathcal{U}$.DEQUEUE
    $s_{old} \leftarrow s(r)$
    if $var(r) \in X$ **then** ▷ r is a decision node.
      $s(r) \leftarrow s(\text{ACTIVECHILD}(r))$
      if $var(r) \in F$ **and** $var(r)$ is *false* **then**
        $\delta \leftarrow \delta + \pi(r) \cdot \left(s(r^+) - s(r^-)\right)$
    else ▷ r is a stochastic node.
      $s_{new} \leftarrow w(r) \cdot s(r^+) + (1 - w(r)) \cdot s(r^-)$
    if $s_{new} \neq s_{old}$ **then** ▷ We do not need to continue the propagation if the score for r has not changed.
      $s(r) \leftarrow s_{new}$
      for $p \in$ PARENTS(r) **do**
        if not REMOVED(p, r) **then** ENQUEUERELEVANT($\mathcal{U}$, p) **end if**
  return δ

Algorithm B.5 Given a set F of decision variables that were fixed (either by branching or by propagation), update the path weights that may have changed due to this. See Algorithm B.9 for helper functions.

procedure UPDATEPATHWEIGHTS(F) ▷ Downward sweep.
  $\mathcal{Q} \leftarrow \varnothing$ ▷ Min heap (treat as set).
  for $r \in \{r \mid var(r) \in F \wedge \texttt{Reachable}[r] > 0\}$ **do**
    if $var(r)$ is *false* **then**
      $\mathcal{Q}$.ENQUEUE(r^-)
      $\mathcal{Q}$.ENQUEUE(r^+)
  while $\mathcal{Q} \neq \varnothing$ **do**
    $r \leftarrow \mathcal{Q}$.DEQUEUE
    $\pi_{old} \leftarrow \pi(r)$
    if r is an original root of the OBDD **then** $\pi_{new} \leftarrow \rho_r$ **else** $\pi_{new} \leftarrow 0$ ▷ Roots have a path weight of ρ_r, which is the utility of the corresponding query.
    for $p \in$ PARENTS(r) **do**
      if $var(p)$ is decision variable **then** ▷ r is a decision node
        if ACTIVECHILD(p) $= r$ **then** $w \leftarrow 1$ **else** $w \leftarrow 0$
      else ▷ r is a stochastic node
        if r is hi child of p **then** $w \leftarrow w(p)$ **else** $w \leftarrow (1 - w(p))$
      $\pi_{new} \leftarrow \pi_{new} + \pi(p) \cdot w$
    if $\pi_{new} \neq \pi_{old}$ **then** ▷ We do not need to continue the propagation if the path weight has not changed.
      $\pi(r) \leftarrow \pi_{new}$
      if $var(r)$ is stochastic variable **then** ▷ r is a stochastic node.
        ENQUEUERELEVANT($\mathcal{Q}$, r^-)
        ENQUEUERELEVANT($\mathcal{Q}$, r^+)
      else ▷ r is a decision node.
        ENQUEUERELEVANT($\mathcal{Q}$, ACTIVECHILD(r))

Algorithm B.6 Enforce domain consistency by fixing free variables to *true* if we find that fixing them to *false* cannot lead to a solution to the stochastic constraint.

procedure ENFORCEDOMAINCONSISTENCY(σ', F, s)
  for $d \in X_{free}$ **do**
    $\Delta \leftarrow 0$ ▷ Partial derivative for free decision variable d.
    for $r \in \{r \in \text{OBDD}_d \mid \texttt{Reachable}[r]\}$ **do**
      $\Delta \leftarrow \Delta + \pi(r) \cdot \left(s(r^-) - s(r^+)\right)$ ▷ Update the partial derivative for d.
    if $s - \Delta \leq \theta$ **then** ▷ The current partial strategy cannot be extended to a valid solution if we fix d to *false*.
      $\sigma' \leftarrow \sigma' \cup \{d = true\}$ ▷ Infer that d must be *true*.
      $X_{free} \leftarrow X_{free} \setminus \{d\}$
      $F \leftarrow F \cup \{d\}$
  return (σ', F)

Algorithm B.7 Update the `Reachable` and `FreeIn` counters after fixing decision variables F. See Algorithm B.9 for helper functions.

```
1: procedure UpdateReachableFreeIn(F)                          ▷ Downward sweep.
2:     Q ← ∅                                                   ▷ Min heap (treat as set).
3:     procedure EnqueueIfNeedToPropagate(r)
4:         if FreeOut[r] > 0 and (FreeIn[r] = 0 OR Reachable = 0) then
5:             Q.enqueue(r)

6:     S ← {r | var(r) ∈ F and Reachable[r] > 0 and FreeOut[r] > 0}
7:     for r ∈ S do
8:         a ← activeChild(r)
9:         i ← inactiveChild(r)
10:        if a is not a leaf and FreeIn[r] = 0 then
11:            FreeIn[a] ← FreeIn[a] − 1
12:            if a ∉ S then EnqueueIfNeedToPropagate(a) end if
13:        if i is not a leaf then
14:            FreeIn[i] ← FreeIn[i] − 1
15:            Reachable[i] ← Reachable[i] − 1
16:            if i ∉ S then EnqueueIfNeedToPropagate(i) end if
17:    while Q ≠ ∅ do
18:        r ← Q.dequeue
19:        if Reachable[r] = 0 then
20:            for c ∈ children(r) do
21:                if c is not a leaf and removed(r, c) then
22:                    FreeIn[c] ← FreeIn[c] − 1
23:                    Reachable[c] ← Reachable[c] − 1
24:                    EnqueueIfNeedToPropagate(c)
25:        else
26:            if FreeIn[r] = 0 and var(r) is decision and var(r) is bound then
27:                for c ∈ children(r) do
28:                    if c is not a leaf and not removed(r, c) then
29:                        FreeIn[c] ← FreeIn[c] − 1
30:                        EnqueueIfNeedToPropagate(c)
```

Algorithm B.8 Update the `FreeOut` counter after fixing decision variables V. See Algorithm B.9 for helper functions.

```
1: procedure UpdateFreeOut(V)                                  ▷ Upward sweep.
2:     U ← {r | var(r) ∈ V ∧ Reachable[r] > 0 ∧ FreeIn[r] > 0}  ▷ Max heap (treat as set).
3:     while U ≠ ∅ do
4:         r ← U.dequeue
5:         if var(r) ∈ V then
6:             if FreeOut[activeChild(r)] > 0 then FreeOut[r] ← 1 else FreeOut[r] ← 0
7:         if FreeOut[r] = 0 and (var(r) is stochastic variable OR var(r) is bound) then
8:             for p ∈ parents(r) do
9:                 if not removed(p, r) and relevant(p) then
10:                    FreeOut[p] ← FreeOut[p] − 1
11:                    U.enqueue(p)
```

Appendix C. Expansion of the parameter space

In this appendix, we provide some details on the design space of the compilation phase and global SCMD propagator solving phase, in addition to our brief summary of Section 8.4.

C.1. Compilation phase parameter space

We summarise the parameters for the compilation phase of our pipelines in Table C.8. Note that the first three parameters are to choose between OBDD or SDD compilation, and to choose if and how to minimise the DD. Because of our focus on OBDDs in this work, we have explicitly listed the OBDD compilation parameters in Table C.8. As the table shows, we tune three categorical OBDD-specific parameters, two parameters with integer domains and one with a real domain.

Since SDDs can be more succinct than OBDDs, and may therefore yield smaller and potentially easier-to-solve LPs, we also consider SDD encodings in our configuration experiments. For the SDD compilation we tune a convergence threshold parameter, four parameters related to limiting the size of the Cartesian product created during SDD minimisation operations, six parameters related to limiting the relative increase of the size of the SDD during minimisation operations and six parameters related to limiting the time taken by minimisation operations. The SDD-specific parameters that we tune include eight categorical parameters, five parameters with integer domains and four parameters with real domains.

Algorithm B.9 Helper functions for the update algorithms.

Upon finding a solution with score s, update the threshold θ, which is the next score to beat.

```
1: procedure UpdateSCMDthreshold(s)
2:     θ ← s
```

For free and *true* variables, the hi child is active. For *false* variables the lo child is active. This function returns the active child of node r.

```
3: procedure activeChild(r)
4:     switch var(r) do
5:         case var(r) is free
6:             return r+
7:         case var(r) is true
8:             return r+
9:         case var(r) is false
10:            return r−
```

Fixing variables to values corresponds to removing their other outgoing arc (corresponding to the opposite value) from the diagram. This function checks if an arc is removed.

```
11: procedure removed(p, r)
12:     if var(p) is not free and activeChild(p) ≠ r then
13:         return true
14:     return false
```

A node is relevant if it corresponds to a free decision variable that has a connection to the root, or if the node corresponds to a stochastic variable and is on a path from one free decision node to another.

```
15: procedure relevant(r)
16:     if var(r) is free and Reachable[r] > 0 then
17:         return true
18:     else if FreeIn[r] > 0 and FreeOut[r] > 0 then
19:         return true
20:     else
21:         return false

22: procedure enqueueRelevant(Q, r)
23:     if relevant(r) then Q.enqueue(r) end if
```

Table C.8
The configuration space of the compilation phase of our pipelines. Note that some of the parameters are conditional on the value of other parameters.

parameter	domain	description
`Diagram`	{*OBDD*, *SDD*}	Compile $\phi(X, T)$ into an OBDD or SDD.
`Minimise`	{*true*, *false*}	Minimise DD after compilation or not.
`DynMinimise`	{*true*, *false*}	Use dynamic minimisation during compilation.
Parameters conditional on `Diagram` = *OBDD*:		
`VarOrder`	{*Sif*, *SymSif*, *GSif*, *WP*, *SA*, *GA*, *Rand*}	Variable reordering algorithm used for OBDD minimisation (if `Minimise` = *true*).
`Converging`	{*true*, *false*}	Repeat variable reordering algorithm until no improvement on OBDD size is found (if `VarOrder` ∈ {*Sif*, *SymSif*, *GSif*, *WP*}).
`MaxSwap`	$\mathbb{N}^+$	Upper bound on number of times two variables can be swapped in the variable order (if `VarOrder` ∈ {*Sif*, *SymSif*, *GSif*}).
`MaxSift`	$\mathbb{N}^+$	Upper bound on number of variables that are sifted, *i.e.*, moved up or down in the variable order (`VarOrder` ∈ {*Sif*, *SymSif*, *GSif*}).
`MaxGrowth`	$\mathbb{R}^+$	Upper bound on relative OBDD size increase during minimisation (if `VarOrder` ∈ {*Sif*, *SymSif*, *GSif*}).
`WSizes`	{2, 3, 4}	Evaluate permutations of `WSizes` consecutive variables in the variable order at a time (if `VarOrder` = *WP*).

C.2. SCMD propagator parameter space

Since branching heuristics can have a big influence on the size of the search tree, we also implemented a number of alternative branching heuristics for the global SCMD propagation method. We briefly discussed these in Section 8.4, and summarise them in Table C.9.

In problems where decision variables are associated with nodes in a network (*e.g.*, Example 2.1), the *Degree* heuristic branches based on the unweighted, undirected degree of the nodes. Similarly, *Influence* estimates the influence of nodes in order to quickly find a high-quality solution, inspired by work on social influence [11]. We translate the influence heuristic to problems with decision variables associated with the edges in the underlying network (*e.g.*, Example 2.2), by taking for each edge the sum of the influence scores of its endpoints. We compute a degree-based score for edges using the *local-degree* measure from [45].

Table C.9
The branching heuristics when using the global SCMD propagator. Some of the parameters are conditional on the value of other parameters.

parameter	domain	description
`VarSelHeur`	{*Top*, *Bottom*, *Derivative*, *Degree*, *Influence*, *Triangle*, *Similarity*, *Simmelian*, *ForestFire*, *Betweenness*, *Random*}	Heuristics to select which variable to branch on next.
`ValSelHeur`	{0, 1}	Heuristics to select which value to branch on first.
`TimeSteps`	$\mathbb{N}^+$	If `VarSelHeur` = *Influence*.
`NumSamples`	$\mathbb{N}^+$	If `VarSelHeur` = *Betweenness*.
`FireProb`	[0, 1]	If `VarSelHeur` = *ForestFire*.
`EdgesBurnt`	[0, 1]	If `VarSelHeur` = *ForestFire*.

We observe that problems such as the theory compression problem of the *spine* problem instances or the power grid reliability problem of Example 2.2 are very similar to graph sparsification problems. We therefore derived the *Triangle*, *Similarity*, *Simmelian* and *ForestFire* heuristics from recent work on this problem [45,66]. For the *Triangle* heuristic, we simply take the number of triangles that a node or edge is part of (not taking into account weights or directionality), to create versions of this heuristic that are suitable for problems with decision variables on nodes or edges, respectively. We translated the *Similarity*, *Simmelian* and *ForestFire* heuristics to problems with decision variables associated with the nodes in the underlying network (*e.g.*, Example 2.1), by summing the scores of all incident edges on a node (not taking into account their weights or directionality). Finally, we use an estimate of either node or edge betweenness centrality as a proxy for the importance of a decision variable in the *Betweenness* heuristic.

References

[1] C.C. Aggarwal, J. Han (Eds.), Frequent Pattern Mining, Springer, 2014.
[2] C. Ansótegui, Y. Malitsky, H. Samulowitz, M. Sellmann, K. Tierney, Model-based genetic algorithms for algorithm configuration, in: IJCAI, AAAI Press, 2015, pp. 733–739.
[3] B. Babaki, T. Guns, L. De Raedt, Stochastic constraint programming with AND-OR branch-and-bound, in: IJCAI, ijcai.org, 2017, pp. 539–545.
[4] P. Balaprakash, M. Birattari, T. Stützle, Improvement strategies for the F-Race algorithm: sampling design and iterative refinement, in: Proceedings of HM, Springer, 2007, pp. 108–122.
[5] A. Bart, F. Koriche, J. Lagniez, P. Marquis, An improved CNF encoding scheme for probabilistic inference, in: ECAI, IOS Press, 2016, pp. 613–621.
[6] A. Biere, M. Heule, H. van Maaren, T. Walsh (Eds.), Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 185, IOS Press, 2009.
[7] S. Bistarelli, F. Rossi, Semiring-based soft constraints, in: Concurrency, Graphs and Models, Springer, 2008, pp. 155–173.
[8] V.D. Blondel, J. Guillaume, R. Lambiotte, E. Lefebvre, Fast unfolding of community hierarchies in large networks, arXiv:0803.0476 [abs], 2008.
[9] H.L. Bodlaender, F. van den Eijkhof, L.C. van der Gaag, On the complexity of the MPA problem in probabilistic networks, in: ECAI, IOS Press, 2002, pp. 675–679.
[10] B. Bollig, I. Wegener, Improving the variable ordering of OBDDs is NP-complete, IEEE Trans. Comput. 45 (1996) 993–1002.
[11] C. Borgs, M. Brautbar, J.T. Chayes, B. Lucier, Maximizing social influence in nearly optimal time, in: SODA, SIAM, 2014, pp. 946–957.
[12] S. Bova, SDDs are exponentially more succinct than OBDDs, in: AAAI, AAAI Press, 2016, pp. 929–935.
[13] S.P. Bradley, A.C. Hax, T.L. Magnanti, Applied Mathematical Programming, Addison-Wesley, 1977.
[14] R.E. Bryant, Graph-based algorithms for Boolean function manipulation, IEEE Trans. Comput. 35 (1986) 677–691.
[15] Y. Caseau, F. Laburthe, Solving various weighted matching problems with constraints, Constraints 5 (2000) 141–160.
[16] A. Charnes, W.W. Cooper, Chance-constrained programming, Manag. Sci. 6 (1959) 73–79.
[17] M. Chavira, A. Darwiche, On probabilistic inference by weighted model counting, Artif. Intell. 172 (2008) 772–799.
[18] A. Choi, A. Darwiche, Dynamic minimization of sentential decision diagrams, in: AAAI, AAAI Press, 2013, pp. 187–194.
[19] G.H. Dal, P.J.F. Lucas, Weighted positive binary decision diagrams for exact probabilistic inference, Int. J. Approx. Reason. 90 (2017) 411–432.
[20] A. Darwiche, On the tractable counting of theory models and its application to truth maintenance and belief revision, J. Appl. Non-Class. Log. 11 (2001) 11–34.
[21] A. Darwiche, A differential approach to inference in Bayesian networks, J. ACM 50 (2003) 280–305.
[22] A. Darwiche, Modeling and Reasoning with Bayesian Networks, Cambridge University Press, 2009.
[23] A. Darwiche, SDD: a new canonical representation of propositional knowledge bases, in: IJCAI, IJCAI/AAAI, 2011, pp. 819–826.
[24] L. De Raedt, K. Kersting, A. Kimmig, K. Revoredo, H. Toivonen, Compressing probabilistic Prolog programs, Mach. Learn. 70 (2008) 151–168.
[25] L. De Raedt, A. Kimmig, H. Toivonen, ProbLog: a probabilistic Prolog and its application in link discovery, in: IJCAI, 2007, pp. 2462–2467.
[26] P.M. Domingos, M. Richardson, Mining the network value of customers, in: KDD, ACM, 2001, pp. 57–66.
[27] C. Domshlak, J. Hoffmann, Probabilistic planning via heuristic forward search and weighted model counting, J. Artif. Intell. Res. 30 (2007) 565–620.
[28] DTAI Research Group, K.U. Leuven, ProbLog Python library, https://github.com/ML-KULeuven/problog, 2015–2019.
[29] L. Dueñas-Osorio, K.S. Meel, R. Paredes, M.Y. Vardi, Counting-based reliability estimation for power-transmission grids, in: AAAI, 2017, pp. 4488–4494.
[30] D. Fierens, G. Van den Broeck, J. Renkens, D. Shterionov, B. Gutmann, I. Thon, G. Janssens, L. De Raedt, Inference and learning in probabilistic logic programs using weighted Boolean formulas, Theory Pract. Log. Program. 15 (2015) 358–401.
[31] D. Fokkinga, A.L.D. Latour, M. Anastacio, S. Nijssen, H. Hoos, Programming a stochastic constraint optimisation algorithm, by optimisation, in: Data Science Meets Optimisation Workshop in Conjunction with IJCAI, 2019.
[32] G. Gange, P.J. Stuckey, V. Lagoon, Fast set bounds propagation using a BDD-SAT hybrid, J. Artif. Intell. Res. 38 (2010) 307–338.
[33] P. Hawkins, P.J. Stuckey, A hybrid BDD and SAT finite domain constraint solver, in: PADL, Springer, 2006, pp. 103–117.
[34] D. Hemmi, G. Tack, M. Wallace, A recursive scenario decomposition algorithm for combinatorial multistage stochastic optimisation problems, in: AAAI, 2018, pp. 1322–1329.
[35] H.H. Hoos, Automated algorithm configuration and parameter tuning, in: Autonomous Search, Springer, 2012, pp. 37–71.
[36] H.H. Hoos, Programming by optimization, Commun. ACM 55 (2012) 70–80.
[37] J. Huang, Combining knowledge compilation and search for conformant probabilistic planning, in: ICAPS, AAAI, 2006, pp. 253–262.
[38] J. Huang, M. Chavira, A. Darwiche, Solving MAP exactly by searching on compiled arithmetic circuits, in: AAAI, AAAI Press, 2006, pp. 1143–1148.

[39] F. Hutter, H.H. Hoos, K. Leyton-Brown, Automated configuration of mixed integer programming solvers, in: CPAIOR, Springer, 2010, pp. 186–202.
[40] F. Hutter, H.H. Hoos, K. Leyton-Brown, Sequential model-based optimization for general algorithm configuration, in: Proceedings of LION, Springer, 2011, pp. 507–523.
[41] D. Kempe, J.M. Kleinberg, Éva Tardos, Maximizing the spread of influence through a social network, in: KDD, ACM, 2003, pp. 137–146.
[42] L. Kotthoff, Constraint solvers: an empirical evaluation of design decisions, CoRR, arXiv:1002.0134 [abs], 2010.
[43] A.L.D. Latour, B. Babaki, A. Dries, A. Kimmig, G. Van den Broeck, S. Nijssen, Combining stochastic constraint optimization and probabilistic programming — from knowledge compilation to constraint solving, in: CP, Springer, 2017, pp. 495–511.
[44] A.L.D. Latour, B. Babaki, S. Nijssen, Stochastic constraint propagation for mining probabilistic networks, in: IJCAI, ijcai.org, 2019, pp. 1137–1145.
[45] G. Lindner, C.L. Staudt, M. Hamann, H. Meyerhenke, D. Wagner, Structure-preserving sparsification of social networks, in: ASONAM, ACM, 2015, pp. 448–454.
[46] M.L. Littman, J. Goldsmith, M. Mundhenk, The computational complexity of probabilistic planning, J. Artif. Intell. Res. 9 (1998) 1–36.
[47] M. Lombardi, M. Milano, Allocation and scheduling of conditional task graphs, Artif. Intell. 174 (2010) 500–529.
[48] R. Mateescu, R. Dechter, Mixed deterministic and probabilistic networks, Ann. Math. Artif. Intell. 54 (2008) 3–51.
[49] K.I.M. McKinnon, H.P. Williams, Constructing integer programming models by the predicate calculus, Ann. Oper. Res. 21 (1989) 227–245.
[50] M.E. Newman, The structure of scientific collaboration networks, Proc. Natl. Acad. Sci. 98 (2001) 404–409.
[51] OscaR Team, OscaR: scala in OR, Available from https://bitbucket.org/oscarlib/oscar, 2012.
[52] O. Ourfali, T. Shlomi, T. Ideker, E. Ruppin, R. Sharan, SPINE: a framework for signaling-regulatory pathway inference from cause-effect experiments, in: ISMB/ECCB (Supplement of Bioinformatics), 2007, pp. 359–366.
[53] C.H. Papadimitriou, Games against nature, J. Comput. Syst. Sci. 31 (1985) 288–301.
[54] J.D. Park, A. Darwiche, Solving MAP exactly using systematic search, in: UAI, Morgan Kaufmann, 2003, pp. 459–468.
[55] G. Perez, J. Régin, MDDs: sampling and probability constraints, in: CP, Springer, 2017, pp. 226–242.
[56] K. Pipatsrisawat, A. Darwiche, New compilation languages based on structured decomposability, in: AAAI, AAAI Press, 2008, pp. 517–522.
[57] K. Pipatsrisawat, A. Darwiche, A new d-DNNF-based bound computation algorithm for functional E-MAJSAT, in: IJCAI, 2009, pp. 590–595.
[58] T. Rainforth, T.A. Le, J. van de Meent, M.A. Osborne, F.D. Wood, Bayesian optimization for probabilistic programs, in: NIPS, 2016, pp. 280–288.
[59] A. Rendl, G. Tack, P.J. Stuckey, Stochastic MiniZinc, in: CP, Springer, 2014, pp. 636–645.
[60] S. Riedel, Improving the accuracy and efficiency of map inference for markov logic, in: Proceedings of the Twenty-Fourth Conference on Uncertainty in Artificial Intelligence, 2008, pp. 468–475.
[61] F. Rossi, P. van Beek, T. Walsh (Eds.), Handbook of Constraint Programming, Foundations of Artificial Intelligence, vol. 2, Elsevier, 2006.
[62] D. Roth, On the hardness of approximate reasoning, Artif. Intell. 82 (1996) 273–302.
[63] R. Rudell, Dynamic variable ordering for ordered binary decision diagrams, in: Proceedings of IEEE/ACM ICCAD, IEEE, 1993, pp. 42–47.
[64] T. Sang, P. Beame, H.A. Kautz, Performing Bayesian inference by weighted model counting, in: AAAI, AAAI Press / The MIT Press, 2005, pp. 475–482.
[65] T. Sato, A statistical learning method for logic programs with distribution semantics, in: ICLP, MIT Press, 1995, pp. 715–729.
[66] V. Satuluri, S. Parthasarathy, Y. Ruan, Local graph sparsification for scalable clustering, in: SIGMOD Conference, ACM, 2011, pp. 721–732.
[67] P. Schaus, J.O.R. Aoga, T. Guns, Coversize: a global constraint for frequency-based itemset mining, in: CP, Springer, 2017, pp. 529–546.
[68] F. Somenzi, CUDD: CU Decision Diagram package-release 2.4.0, University of Colorado at Boulder, 2004.
[69] S.A. Tarim, B. Hnich, S.D. Prestwich, R. Rossi, Finding reliable solutions: event-driven probabilistic constraint programming, Ann. Oper. Res. 171 (2009) 77–99.
[70] H. Verhaeghe, C. Lecoutre, P. Schaus, Compact-MDD: efficiently filtering (s)MDD constraints with reversible sparse bit-sets, in: IJCAI, 2018, pp. 1383–1389.
[71] B. Viswanath, A. Mislove, M. Cha, K. Gummadi, On the evolution of user interaction in Facebook, in: Proceedings of ACM SIGCOMM WOSN, 2009, pp. 37–42.
[72] T. Walsh, Stochastic constraint programming, in: ECAI, IOS Press, 2002, pp. 111–115.
[73] B. Wiegmans, Gridkit: European and North-American extracts, https://doi.org/10.5281/zenodo.47317, 2016.
[74] Y. Xue, X. Wu, D. Morin, B. Dilkina, A. Fuller, J.A. Royle, C.P. Gomes, Dynamic optimization of landscape connectivity embedding spatial-capture-recapture information, in: AAAI, AAAI Press, 2017, pp. 4552–4558.