

Stochastic Constraint Optimization using Propagation on Ordered Binary Decision Diagrams

Anna L.D. Latour

LIACS, Leiden University
Leiden, The Netherlands

a.l.d.latour@liacs.leidenuniv.nl

Behrouz Babaki

École Polytechnique de Montréal
Montreal, Canada

Siegfried Nijssen

ICTEAM, Université catholique de Louvain
Louvain-la-Neuve, Belgium

siegfried.nijssen@uclouvain.be

Abstract

A number of problems in relational Artificial Intelligence can be viewed as Stochastic Constraint Optimization Problems (SCOPs). These are constraint optimization problems that involve objectives or constraints with a stochastic component. Building on the recently proposed language SC-ProbLog for modeling such problems, we propose a new method for solving these problems. Earlier techniques used Probabilistic Logic Programming (PLP) techniques to create Ordered Binary Decision Diagrams (OBDDs), which were decomposed into smaller constraints in order to exploit existing constraint programming (CP) solvers. We argue that this approach has as drawback that a decomposed representation of an OBDD does not guarantee domain consistency during search, and hence limits the efficiency of the solver. For the specific case of *monotonic* distributions, we suggest an alternative method for using CP in SCOP, based on the development of a new propagator; we show that this propagator is linear in the size of the OBDD, and has the potential to be more efficient than the decomposition method because it maintains domain consistency.

1 Introduction

Making decisions under uncertainty is an important problem in business, governance, science and our daily lives in general. Examples are found in the fields of planning and scheduling, but also occur naturally in fields like data science and bioinformatics. Many of these problems are *relational* in nature.

Consider for example a *viral marketing problem* (?). We are given a social network of people (vertices) that have stochastic relationships (edges). We want to rely on word-of-mouth advertisement to turn acquaintances of people who buy our product into new product-buyers. How can we minimize the number of people we need to target directly in a marketing campaign, while a minimum number of people is expected to buy the product?

For another example we are given a network of stochastic protein-gene interactions, with a list of (protein, gene) pairs that are of interest to a biologist (?). We wish to reduce the network to the part that is relevant for modeling the interactions that are of interest to this biologist. This is known as a *theory compression problem* (?). How do we maximize the sum of interaction probabilities for the interesting pairs,

while restricting the number of edges we include in the extracted network?

These two problems have common features. First, both problems combine probabilistic networks and decision problems: we either decide who to target in our marketing campaign, or which interactions to select from a protein-gene interaction network. Second, they both involve an objective: minimizing the number of people targeted for marketing and maximizing a sum of probabilities, respectively. Third, they both have to respect a constraint: either reaching a target with respect to the expected number of product-buyers or limiting the number of edges we select for our biologist.

The motivation for our ongoing work is that there is a need for generic tools that can be used to model and solve such problems. In our vision, these tools should combine the state-of-the-art of *probabilistic programming* (PP) with *constraint programming* (CP). *Probabilistic programming* here provides mechanisms for calculating probabilities of paths in probabilistic networks. For making decisions, *constraint programming* provides well-established technology.

Note that the stochastic constraint in the viral marketing setting is a hard constraint on a sum of probabilities: we impose a bound on the expected number of people buying the product. This is a different setting than the soft constraints that can be expressed using *maximum a posteriori* (MAP) inference or *maximum probability estimation* (MPE).

Problems that involve these kinds of hard constraints on probabilities are the focus of the field of *stochastic constraint programming* (SCP) (?), which combines probabilistic inference and constraint programming to solve *Stochastic Constraint Optimization Problems* (SCOPs). SCP is closely related to *chance constraint programming* (?) and *probabilistic constraint programming* (?). However, these tools do not provide a modeling language suitable for solving relational problems in a generic manner, and do not link to the probabilistic programming literature.

Recently we proposed a new modeling language and a new tool chain that addresses the problem of modeling and solving relational SCOPs. This language, *Stochastic Constraint Probabilistic Prolog* (SC-ProbLog) (?), is based on (Decision Theoretic) ProbLog (?; ?), and is therefore particularly suited for modeling probabilistic paths. It extends ProbLog with syntax for specifying SCOPs that are formulated on probabilistic networks, and a tool chain for solv-

ing them. Building on ProbLog, SC-ProbLog has functionality for translating a Probabilistic Logic program in Boolean formulas, converting those formulas into Ordered Binary Decision Diagrams (OBDDs) for tractable *weighted model counting* (WMC), converting these OBDDs into Arithmetic Circuits (ACs) and decomposing these into Mixed Integer Programs (MIPs), which in turn serve as input for an off-the-shelf MIP solver or CP solver that solves the SCOP.

The main contribution of this paper is a modification of the last step in this pipeline. While in earlier work, constraint optimization solvers were used as a black boxes on decomposed OBDDs, we propose to open the black box in this work. We will demonstrate that the *propagation* that is used in constraint satisfaction solvers, is not optimal for the constraints resulting from decomposition. Specifically, we will show that constraint propagation is not *domain consistent*: a search algorithm will branch over variables unnecessarily. To address this flaw, we first introduce a naïve propagation algorithm over OBDDs that is domain consistent, and for which the worst case complexity is $O(mn)$, where m is the size of the OBDD and n is the number of decision variables. Note that propagation is executed at every node of the search tree; any reduction of $O(mn)$ to a lower complexity would affect every node of the search tree. We will subsequently show how to calculate partial derivatives over the OBDDs (?), and use these derivatives to reduce the complexity of domain consistent propagation to $O(m + n)$. Here we build on earlier results for linear derivative computation on computational graphs (?; ?) and computation graphs for the deterministic Decomposable Negation Normal Form (d-DNNF) (?). This is a different and more efficient approach for the calculation of derivatives than the one proposed in (?). Furthermore, we will argue that our approach enables the creation of incremental constraint propagation algorithms; this allows for propagation that is more efficient than $O(m + n)$ in practice. Our method assumes the stochastic constraint to have a particular *monotonic* property, which we discuss in more detail in section 2.

In this paper, we give a description of how typical SCOPs can be modeled using SC-ProbLog in section 2, followed by a discussion on how they can be solved. In section 4 we provide a short introduction to some key concepts of CP, which we use in section 5 to introduce a proposal for OBDD-based stochastic constraint propagator for CP systems. We conclude this work with an outlook on future research.

2 Modeling SCOPs with SC-ProbLog

The goal of SC-ProbLog (?) is to provide a generic system for modeling and solving SCOPs. In this section we give an example SCOP and explain how it can be modeled using SC-ProbLog. Before we address that, let us first define the kinds of SCOP that we consider in this work.

Problem Definition

We consider problems that are defined on two types of variables: *decision variables* and mutually independent *stochastic variables* (denoted in this work as d_i and t_i , respectively).

The problems involve a (stochastic) objective function and a set of (stochastic) constraints, all of which can be expressed in terms of these variables. We consider an optimization criterion or constraint to be stochastic if its definition involves stochastic variables. The aim is to find an assignment to the decision variables (also referred to as a *strategy*) such that the constraints are respected and the objective satisfied.

In this work we restrict our focus to variables that can take Boolean values. We can assign a value of *true* or *false* to decision variables, while the value of stochastic variables is mutually independently determined by chance, characterized by an associated probability.

We consider a selection of constraints and objective functions. In particular, we consider constraints that represent a bound on expected utilities and objective functions that maximize or minimize an expected utility, e.g.:

$$\begin{aligned} \sum_i r_i v_i &\geq \theta && \text{stochastic constraint} \\ \max \sum_i r_i v_i &&& \text{stochastic optimization criterion} \end{aligned} \quad (1)$$

where v_i either represents the value of a decision variable d_i , or a conditional probability $P(\phi_i \mid \sigma)$. Here ϕ_i represents an event, and the conditional probability represents the probability of that event happening (i.e. ϕ_i evaluating to *true*), given a strategy σ . With v_i we associate a reward $r_i \in \mathbb{R}^+$, such that the expressions in equation (1) represent *expected utilities*. For simplicity we will assume $r_i = 1$ in this work, but note that generalizing our approach to $r_i \neq 1$ is trivial. Finally θ is a *threshold* for the constraint.

Intuitively, in the optimization criterion of the viral marketing problem, v_i represents the probability of the event ϕ_i that person i buys a product, given a marketing strategy. The marketing strategy is represented by decision variables d_i .

In this work we impose an additional *monotonicity* condition on each probability $P(\phi_i \mid \sigma)$: we require that for any strategy σ , switching the value of any decision variable from *false* to *true*, will yield a probability that is not smaller: $P(\phi_i \mid \sigma) \geq P(\phi_i \mid \sigma')$, if σ' differs from σ by one variable that is *true* in σ' but *false* in σ . This condition is met in all the example problems mentioned earlier.

In this work we will consider solving stochastic *constraints* rather than stochastic *optimization criteria*. However, it is easy to use our results in optimization as well: we can solve a problem involving the *optimization criterion* in equation (1) by repeatedly solving a constraint satisfaction problem involving the *constraint* in equation (1), increasing θ each time we have found a solution until we find a θ for which there exists no solution.

An Example SCOP

Consider the network in figure 1, and suppose that information can flow through each edge with a certain probability. We can formulate a theory compression problem as described in section 1 on this network. Suppose we want to maximize the sum of probabilities that information can flow from a to c and from a to d , but we want to limit the number of edges in the network, such that there are no more than 2 (cardinality constraint). We can model this as follows:

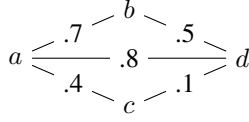


Figure 1: A small network of four nodes (a , b , c and d) and five undirected edges with associated probabilities.

- with each edge (i, j) in the network we associate a stochastic variable t_{ij} and a decision variable d_{ij} ;
- with each variable t_{ij} we associate a probability $p(t_{ij})$;
- the events considered are $\phi_{a \rightarrow c}$ and $\phi_{a \rightarrow d}$, which represent flow of information from a to c and from a to d ;
- our objective is to find a σ that maximizes $P(\phi_{a \rightarrow c} \mid \sigma) + P(\phi_{a \rightarrow d} \mid \sigma)$;
- our constraint is $\sum_k d_k \leq 2$.

Subsequently, we need to define the probability of events $\phi_{a \rightarrow c}$ and $\phi_{a \rightarrow d}$, given a strategy. Here, we use a WMC approach. We use a logical formula to represent when an event is *true*, given an assignment to the decision variables and a sample for the stochastic variables:

$$\phi_{a \rightarrow c} = (d_{ac} \wedge t_{ac}) \vee (d_{ad} \wedge t_{ad} \wedge d_{cd} \wedge t_{cd}) \vee (d_{ab} \wedge t_{ab} \wedge d_{bd} \wedge t_{bd} \wedge d_{cd} \wedge t_{cd}). \quad (2)$$

Here, if t_{ij} and d_{ij} are *true*, then information can travel through edge (i, j) . The logical formula represents all the ways in which information can travel from a to c .

The probability $P(\phi_{a \rightarrow c} \mid \sigma)$ is then defined as the sum of the probabilities of all the (logical) models of this formula. Given strategy $\sigma = (d_{ac} = d_{ad} = d_{cd} = d_{ab} = d_{bd} = \top)$, one model is for instance $t_{ac} = \top, t_{ab} = t_{ad} = t_{cd} = t_{bd} = \perp$, of which the probability is $.4 \cdot (1 - .8) \cdot (1 - .1) \cdot (1 - .5) \cdot (1 - .7)$; in principle, we sum the probabilities of all such models to obtain $P(\phi_{a \rightarrow c} \mid \sigma)$. Note that equation (2) has indeed a monotonic property: the more decision variables are *true*, the higher the probability of the event is.

To program such a formulas in a generic manner, as well as to define constraints and optimization criteria, we proposed SC-ProbLog (?), which is also based on weighted model counting. The following program in SC-ProbLog would model the problem described above:

```
% Deterministic facts
1. node(a).      node(b).      node(c).      node(d).

% Probabilistic facts
2. 0.7::t(a,b).   0.8::t(a,d).   0.5::t(b,d).
3. 0.4::t(a,c).   0.1::t(c,d).

% Decision variables
4. ?::d(a,b).     ?::d(a,d).     ?::d(b,d).
5. ?::d(a,c).     ?::d(c,d).

% Relations
6. e(X,Y) :- t(X,Y), d(X,Y).  e(Y,X) :- t(X,Y), d(X,Y).
7. path(X,Y) :- e(X,Y).
8. path(X,Y) :- X \= Y, e(X,Z), path(Z,Y).
```

```
% Constraints and optimization criteria
9. { d(X,Y) => 1 :- node(X), node(Y). } 2.
10. #maximize { path(a,c) => 1. path(a,d) => 1. }.
```

Here, we define the nodes in the network on line 1. Lines 2 and 3 associate the correct probability with each edge; these are the stochastic variables. We define the decision variables in lines 4 and 5. Edges are made undirected in line 6 and we give the definition of a path in lines 7 and 8. In line 9 we define the constraint: we assign a utility of 1 to each decision variable that is *true*. We also specify that we only allow decision variables that reflect the edges between nodes that are actually present in the network. Finally, line 10 represents the optimization criterion: we assign a utility of 1 to there being a path from a to c and to there being a path from a to d . The utilities are summed and weighted by the actual probability of there being such paths. The logical formulas $\phi_{a \rightarrow c}$ and $\phi_{a \rightarrow d}$ are constructed from the program by ProbLog.

An interesting feature of SC-ProbLog is that any problem that does not contain negation or negative weights, represents a monotonic utility function. We restrict our attention in this work to such functions.

In the next section we briefly discuss how to compute the probabilities of such formulas efficiently and how to solve the SCOP of which they are a part.

3 Solving SCOPs using CP

We assume that the reader is familiar with ProbLog¹. In case of absence of that familiarity, we refer the reader to the literature, e.g. (?; ?). We start this section with a short recap of why ProbLog uses *knowledge compilation* to obtain OBDDs; subsequently, we discuss how OBDDs can be used to solve naively the associated SCOP. Then we discuss the earlier proposed tool chain for solving SCOPs (?) and reflect on it.

From ProbLog to OBDD

Consider equation (2), and observe that computing $P(\phi_{a \rightarrow c} \mid \sigma)$ is complicated, as the different paths need to be enumerated, but may also overlap. Therefore, computing this probability involves a disjoint sum problem; in the general case WMC is #P-complete (?).

In ProbLog the tractability of this task is addressed by compiling the formulas during a preprocessing phase into a Sentential Decision Diagram (SDD) (?) or OBDD that allows for tractable WMC. The advantage of this method is that, once this diagram is compiled, computing $P(\phi \mid \sigma)$ has a complexity that is linear in the size of the diagram, thus reducing the complexity of the WMC task (at a cost of having to preprocess the formula). This work focuses on stochastic constraints that can be expressed by OBDDs. We assume familiarity with OBDDs, for we will only discuss a few of their characteristics here. For a more extensive overview, see for example (?).

To see how we can compute $P(\phi \mid \sigma)$ using an OBDD, consider figure 2. It shows an OBDD that represents the probability of equation (2) evaluating to *true*. Observe that

¹<https://dtai.cs.kuleuven.be/problog>

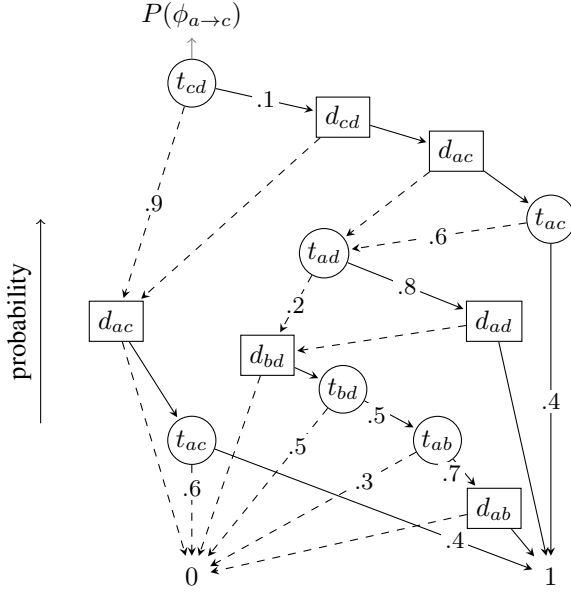


Figure 2: An OBDD representing the probability that information can travel from a to c in figure 1, i.e. the probability that $\phi_{a \rightarrow c}$ evaluates to true given any strategy σ . The variable order corresponding to this OBDD is $t_{cd} < d_{cd} < d_{ac} < t_{ac} < t_{ad} < d_{ad} < d_{bd} < t_{bd} < t_{ab} < d_{ab}$. Circular nodes represent stochastic variables, squares represent decision variables. No specific strategy is reflected here.

the weights on the outgoing arcs of nodes that represent stochastic variables (those labeled with t_{ij}) correspond to the probability that that variable is *true* (for the solid, or *hi*, arcs) or *false* (dashed, or *lo*, arcs). A strategy σ is represented in the OBDD by adding weights of 0 and 1 to the outgoing arcs of the nodes corresponding to decision variables (those labeled with d_{ij}). For example: if we choose $d_{ac} = \perp$, we put a weight of 0 on the outgoing *hi* arc of nodes labeled with d_{ac} and weight 1 on their outgoing *lo* arcs.

Given a strategy σ and arcs labeled accordingly, the OBDD can straightforwardly be mapped to an Arithmetic Circuit (AC). We can compute $P(\phi_{a \rightarrow c} \mid \sigma)$ as follows. In a bottom-up traversal, each OBDD node r takes the value

$$v(r) = w(r) \cdot v(r^+) + (1 - w(r)) \cdot v(r^-), \quad (3)$$

where r^+ (r^-) is the *hi* (*lo*) child of r , i.e. the child connected through the solid (dashed) outgoing arc of r ; $v(r) = 0$ for the negative leaf and $v(r) = 1$ for the positive leaf. Observe that $v(\text{root}) = P(\phi \mid \sigma)$.

The complexity of evaluating $P(\phi \mid \sigma)$ is thus linear in the size of the OBDD, but the number of strategies is 2^n , with n the number of decision variables. The naïve way of solving a SCOP is to enumerate all possible strategies, use the OBDD to evaluate the objective function and/or constraints for each strategy, evaluate possible other constraints, and store the best feasible strategy found so far. Since the number of strategies is exponential in the number of decision variables, this naïve method does not scale well.

Solving SCOPs with the SC-ProbLog tool chain

Since SCOPs are constraint optimization problems, one obvious approach to improving on the naïve method is to leverage the state-of-the-art CP solvers that are available. The tool chain described in (?) takes the OBDD generated by ProbLog and instead of assigning weights to the outgoing arcs of the nodes in the OBDD that represent decision variables, converts the OBDD into an AC in which those weights are present as boolean decision variables.

A constraint is imposed on the value of the AC, and the then decomposed into a Mixed Integer Program (MIP); a set of smaller constraints is constructed that represent the value at each node of the OBDD according to equation (3). See figure 3 for an example of what such a MIP may look like.

As mentioned in section 1, this method has a disadvantage: during the search process, the solver cannot guarantee domain consistency on the MIP representing the constraint. We propose an alternative to this decomposition method in section 5, but will first make the notion of some basic CP concepts, including domain consistency, more concrete.

4 Introduction to Constraint Programming

Constraint programming is an area that studies the development of modeling languages and solvers for constraint satisfaction and optimization problems. Two processes form the basis of Constraint Programming solvers: *search* and *propagation*. We briefly discuss these concepts, for they are critical to understanding our contributions in this work. For a more comprehensive overview of CP, we refer the reader to the literature, e.g. *Principles of Constraint Programming* (?). Then we continue with a discussion of the relation between these principles and the circuit decomposition method (?).

Search and Propagation

The search process is some structured method for exploring the search space of the problem. In our SCOP setting, the search space consists of all possible assignments to the (binary) decision variables, from which we need to find one that satisfies the constraints and optimizes the objective function.

The details of the search process are outside the scope of this work, but for search over binary variables the process is roughly as follows. Initially, all variables are considered to be free or unassigned; they have a *domain* of $\{0, 1\}$. Then repeatedly a free decision variable d is selected, and fixed to a value (either *true* or *false*). After each such assignment, *propagation* is used to determine whether other variables can be fixed. Propagation is the process of updating the domains of other *free variables*, making them reflect the consequences of the assignments made to decision variables (the *fixed variables*) so far. If propagation yields a contradiction, the search backtracks over the last variable assignment; otherwise, if a free variable remains, its value is fixed and the search process continues.

The constraints of the problem guide the propagation. For example: the problem may contain a *cardinality constraint* that puts an upper bound of N on the number of variables that can be set to *true*. Suppose that during search, variable

d is selected and fixed to *true*, becoming the N th decision variable to be *true*. Now we know that 1 should be removed from the domain of each remaining free variable. This reduces the search space by making domains smaller.

During propagation two things can happen (possibly simultaneously). One possibility is that the domain of a free variable becomes empty. In this case there is no solution given the current partial assignment to decision variables and we must backtrack to explore a different part of the search space. Alternatively the size of the domain of a free variable is reduced to 1, leaving only one possible value for that variable (given the current partial assignment). Such a variable can then be fixed and removed from the set of free variables, reducing the search space by reducing the number of free variables.

There are myriad optimizations for both search and propagation, but these are outside the scope of this work. Observe that both the nature of the search and of the propagation depend on the type of variables and on the nature of the constraint. In this work we focus on developing a propagator that enforces *domain consistency* on OBDDs.

Domain Consistency

An important notion in propagation is that of *domain consistency*. We define it as follows:

Definition 1. Let $\varphi(x_1, \dots, x_n)$ be a constraint over boolean variables x_1, \dots, x_n . Furthermore, let σ' be partial assignment to the variables x_1, \dots, x_n . Then a propagator for constraint φ is domain consistent if for any σ' this propagator calculates a new partial assignment σ satisfying these conditions: (1) σ extends σ' , (2) for all variables v not assigned by σ , both the partial assignments $\sigma \cup \{v \mapsto \perp\}$ and $\sigma \cup \{v \mapsto \top\}$ can be extended to a complete assignment that satisfies the constraint φ .

In other words, after domain consistent propagation for a constraint, all values have been removed from all variable domains that cannot be part of a solution for that constraint.

We illustrate this notion with an example. A standard practice in CP is to call the propagator *before* the search starts, in order to make the initial domains consistent with the constraint, and, ideally, detect the variables that are forced to a specific value by the constraint.

Consider the OBDD in figure 3 and the associated constraint $P(\phi \mid \sigma) \geq .4$. Observe that the four possible strategies yield the following conditional probabilities, which are monotonic in the decision variables:

$$\begin{aligned} P(\phi \mid x = y = 0) &= 0 & P(\phi \mid x = 1, y = 0) &= .3 \\ P(\phi \mid x = y = 1) &= .6 & P(\phi \mid x = 0, y = 1) &= .6 \end{aligned}$$

From this we conclude that only those strategies in which $y = 1$ can possibly satisfy the constraint. A propagator that ensures domain consistency will detect this before the start of the search and fix y to 1.

The circuit decomposition method translates this constraint on the OBDD in a CP model that is also given in figure 3. Suppose a propagator is called on this decomposed model, before the search starts. This propagator may start by trying to infer the minimum value $v(y_1)$ needs to take if

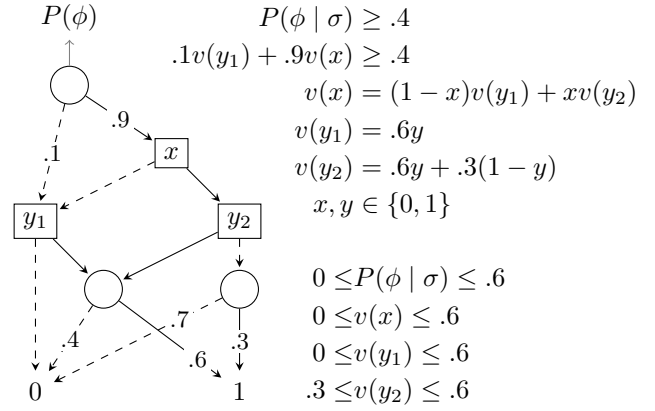


Figure 3: A small OBDD (left) with three stochastic variables and two decision variables. The two nodes corresponding to decision variable y are indexed for clarity. The MIP on the right is constructed using equation (3).

$v(x)$ takes its maximum possible value. To do this, the propagator assumes for a moment that $v(x) = .6$ holds. Now it can infer that, in order for the constraint to be satisfied, $v(y_1) \geq (.4 - .9 \cdot .6) / .1 = -1.4$ should hold. Unfortunately, this does not tell us anything, for we already knew that the domain of y is $\{0, 1\}$ and thus does not include -1.4 . Based on this, we cannot remove 0 from the domain of y . Repeating this procedure to determine a bound for $v(x)$ by assuming $v(y_1)$ takes its maximum value, and from there continuing to determine bounds for $v(y_1)$ and $v(y_2)$ does not yield conclusive evidence to deduct that y must be fixed to 1, either.

This shortcoming of the circuit decomposition method causes a lack of efficiency, since the search space is not reduced as much as possible. In the next section we introduce a propagator for OBDDs that does ensure domain consistency.

5 Approach

We intend to improve upon the existing circuit decomposition approach for solving SCOPs, by allowing an OBDD-based constraint to be added directly to a CP solver, rather than decomposed into a multitude of (linear) constraints. In order to achieve this, we need to introduce a propagator for OBDDs. As discussed in section 4, this propagator should guarantee domain consistency in the OBDD.

In this section we will first introduce a naïve approach for such a domain consistent propagator. Subsequently, we will show how to obtain a better worst-case complexity by using the idea of *derivatives*.

Naïve Propagator

As discussed earlier, we can calculate the quality of any strategy with an algorithm that traverses the OBDD bottom-up, using equation (3).

For the creation of a domain consistent propagator, our first important observation is that our scoring function is monotonic; hence, the largest possible score is obtained by assigning the value *true* to all free decision variables.

The idea behind domain consistent propagation is to repeat the following process for each free decision variable d :

1. fix variable d to the value *false*;
2. fix all other free variables to the value *true*;
3. calculate the score for the resulting assignment;
4. if the score is lower than the desired threshold, remove the value *false* from the domain of variable d .

By construction, this process is domain consistent.

Let n be the number of free decision variables, and let m be the size of the OBDD. Then the complexity of the algorithm above is $O(mn)$: for every free variable we perform a bottom-up traversal of the OBDD. Given that propagation is the most computationally intensive part of search algorithms under our constraint, it is important to obtain a better performance. We will improve this complexity to $O(n+m)$, using an approach similar to that for d-DNNFs (?).

Overview of our Propagator

The key idea behind our improved propagator is that we calculate a derivative

$$\frac{\partial f(d, \sigma' \setminus d)}{\partial d} = f(\sigma') - f(d = \perp, \sigma' \setminus d) \quad (4)$$

for every free decision variable d . Here, σ' represents a full assignment to all decision variables. In this assignment, every free variable is assumed to have the value *true*. Function f represents the function defined by equation (3) on the root of the OBDD. Hence, $f(\sigma')$ represents the best score currently possible, in which all free variables have been given the value \top ; $f(d = \perp, \sigma' \setminus d)$ represents the assignment in which the value for variable d has been switched to \perp .

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

$$f(\sigma') - \frac{\partial f(d, \sigma' \setminus d)}{\partial d} \geq \theta. \quad (5)$$

Clearly, the main question becomes how to calculate $\partial f(d, \sigma' \setminus d) / \partial d$ for all free variables efficiently. Here, we will build on ideas introduced by Darwiche in 2003 (?) to build an $O(m)$ algorithm. This algorithm adapts the ideas of Darwiche to our specific context; we will argue that this enables us to perform propagation for monotonic constraints in an incremental manner, effectively making the complexity lower than $O(m)$.

Calculating the Derivative

We first need to define the concept of *path weight*:

Definition 2. Let r_m be a node labeled with variable x_m in an OBDD with variable order $x_1 < \dots < x_n$. We define the path weight of r_m :

$$\pi(r_m) = \sum_{\ell \in L_{r_m}} \prod_{r_i \in \ell} u_i, \quad (6)$$

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* arc from a node labeled with a decision variable that is false and
- the *lo* arc from a node labeled with a decision variable that is true or free.

In other words: we take paths that reflect the current partial assignment, and take the *hi* arc from free decision nodes.

In our definition of u_i , we use $u_i = 1$ for the outgoing arcs of decision nodes that can be part of a valid path.

For the outgoing arcs of stochastic nodes labeled with a stochastic variable x_i that has weight w_i as defined in the ProbLog program, we use:

$$u_i = \begin{cases} w_i & \text{if we take the hi arc of } r_i; \\ 1 - w_i & \text{if we take the lo arc of } r_i. \end{cases} \quad (7)$$

Note that the path weight $\pi(r_m)$ is expressed in terms of variables $x_i < x_m$ only.

An example: if we were to fix d_{cd} and d_{ac} to *true* in figure 2, then the path weight of the node labeled with t_{ad} would be $\pi(r_{t_{ad}}) = .1 \cdot 1 \cdot (1 \cdot .6 + 0) = .06$.

Our algorithm is based on the observation that derivatives can be calculated using the following equation:

Theorem 1. The derivative of the OBDD with respect to a decision variable can be calculated as follows:

$$\frac{\partial f(d, \sigma' \setminus d)}{\partial d} = \sum_{r_d \in OBDD_d} \pi(r_d) (v(r_d^+) - v(r_d^-)), \quad (8)$$

where $OBDD_d$ represents all nodes in the OBDD labeled with variable d .

Proof. Let $f(x_1, \dots, x_n)$ be the polynomial associated with an OBDD with variable order $x_1 < \dots < x_n$. Let r_i be a node labeled with x_i and let w_i be the positive weight of that variable. Observe that for any variable x_m (with $x_1 \leq x_m \leq x_n$) we can write f as

$$f(x_1, \dots, x_m, \dots, x_n) = \sum_{r_m \in OBDD_m} \pi(r_m) (w_m v(r_m^+) + (1 - w_m) v(r_m^-)), \quad (9)$$

where the values of the the *hi* and *lo* child of r_m are $v(x_m^+)$ and $v(x_m^-)$, respectively, following equation (3). Recall that in the expression of the path weight of r_m , w_m does not occur. Note also that $v(r_m^+)$ and $v(r_m^-)$ are expressed in variables $x_i > x_m$ only. The derivative of this formula (with respect to w_m) corresponds to the claim in the theorem. \square

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

- a top-down pass over the complete OBDD for calculating all path weights;
- 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 pseudo codes for these passes are given in algorithms 1 and 2, respectively.

Once these two passes are completed, we can recompute the derivatives for all decision variables that are still free,

Algorithm 1 Compute path weights.

```

1:  $\pi(\text{root}) \leftarrow 1$ 
2: for  $r_i \in \text{OBDD}$  and  $r_i \neq \text{root}$  do  $\pi(r_i) \leftarrow 0$  end for
3: for each internal node  $r_i \in \text{OBDD}$  (in topological order) do
4:   if  $r_i$  corresponds to decision variable  $d_i$  then
5:     if  $d_i$  is true or free then
6:        $\pi(r_i^+) \leftarrow \pi(r_i^+) + \pi(r_i)$ 
7:     else
8:        $\pi(r_i^-) \leftarrow \pi(r_i^-) + \pi(r_i)$ 
9:   else
10:     $\pi(r_i^+) \leftarrow \pi(r_i^+) + w_i \pi(r_i)$ 
11:     $\pi(r_i^-) \leftarrow \pi(r_i^-) + (1 - w_i) \pi(r_i)$ 

```

Algorithm 2 Compute values.

```

1:  $v(0) \leftarrow 0, v(1) \leftarrow 1$  ▷ Leaf nodes
2: for each internal node  $r_i \in \text{OBDD}$  (in reversed topological order) do
3:   if  $r_i$  corresponds to decision variable  $d_i$  then
4:     if  $d_i$  is true or free then
5:        $v(r_i) \leftarrow v(r_i^+)$ 
6:     else
7:        $v(r_i) \leftarrow v(r_i^-)$ 
8:   else
9:      $v(r_i) \leftarrow w_i v(r_i^+) + (1 - w_i) v(r_i^-)$ 

```

and evaluate equation (5) for each of those to see if we can remove *false* from their domain, such that we can enforce domain consistency. The pseudo code for this is provided in algorithm 3 for clarity, but can be integrating with algorithm 2. It is easy to see that the overall calculation finishes in $O(n + m)$ time.

Traversing Part of the OBDD

For efficient propagation, it is desirable that the complexity of the algorithm above can be brought down further; we should avoid traversing unnecessary parts of the OBDD as much as possible. Building on the ideas presented earlier, a number of different observations allow for more efficient propagation in practice.

As we observed before, the expression for the path weight of an OBDD node labeled with variable x_m (equation (6)) only contains variables $x_i < x_m$. We thus conclude that fixing a decision variable d can only affect the path weights of nodes *below* the nodes labeled with that variable d .

Moreover: because we take the hi arc both from decision nodes that are *free* and from those that are *true*, path weights below free decision nodes are not changed at all when we fix a decision node to *true*.

Therefore: whenever we fix a decision variable, our propagator only needs to call algorithm 1 if we fix it to *false*, and even then it only has to traverse the part of the diagram that is below the nodes labeled with that decision variable.

A similar argument holds for the values of the OBDD nodes. Because they are computed in a bottom-up traversal

Algorithm 3 Enforce domain consistency.

```

1: for each free variable  $d$  do
2:    $\Delta_d \leftarrow 0$ 
3:   for each node  $r \in \text{OBDD}_d$  do
4:      $\Delta_d \leftarrow \Delta_d + \pi(r) \cdot (v(r^+) - v(r^-))$ 
5:   if  $v(\text{root}) - \Delta_d < \theta$  then
6:     remove false from domain of  $d$ 

```

of the OBDD, fixing a variable can only affect the values of the nodes labeled with that variable themselves, and those *above* them in the diagram. Again: only fixing a variable to *false* actually requires the propagator to update values at all.

We can narrow down the parts of the diagram that need to be considered further. Consider the decision variable that occurs closest to the root of the OBDD. We do not need to maintain the values for any of the nodes in the OBDD above it, as we will never need to calculate the derivative for any variable in this part of the diagram. Similarly, consider the variable closest to the leaves; we do not need to maintain pathweights for its children either. It can be shown that by only maintaining the part of the OBDD between two borders, one can calculate the derivatives exactly, as well as calculate the true value of the optimization criterion without propagating towards the root of the diagram.

6 Conclusion and Outlook

Many problems in AI can be seen as SCOPs. In this work we proposed a new method for solving SCOPs that are modeled using PLP techniques, specifically: SC-ProbLog (?; ?). In SC-ProbLog, we can convert the SCOP's stochastic constraints into constraints on OBDDs. This work was motivated by the observation that an earlier approach was not build on domain consistent propagation. We sketched a propagator for such OBDD constraints that does enforce domain consistency in linear time.

We limited our attention to a representation of distributions in OBDDs. The advantage of this representation is that we can clearly identify parts of the diagram *above* and *below* a decision variable; we argued that this can be used to limit the *active* part of the diagram and to limit which type of calculation is performed on which part of the diagram.

Several details were omitted from this paper. We did not include extensive details regarding the maintenance of active parts of OBDDs or the incremental calculation of optimization criteria. Furthermore, we did not include an extension to constraints on sums of probabilities, or to constraints of the form $P(\phi \mid \sigma) \leq \theta$.

Concrete next steps are the implementation of our approach, its evaluation on data, a comparison of different approaches for maintaining active parts of OBDDs, and its possible extension to other types of diagrams.

Acknowledgements. This research was supported by the Netherlands Organisation for Scientific Research (NWO). Behrouz Babaki is supported by a postdoctoral scholarship from IVADO.