



Universiteit  
Leiden  
The Netherlands

## Optimal decision-making under constraints and uncertainty

Latour, A.L.D.

### Citation

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

Version: Publisher's Version

License: [Licence agreement concerning inclusion of doctoral thesis in the Institutional Repository of the University of Leiden](#)

Downloaded from: <https://hdl.handle.net/1887/3455662>

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

# **Part II**

# **Contributions**



# 4


---

## Stochastic constraint (optimisation) problems

---

In this chapter we describe how to formally model *stochastic constraint (optimisation) problems (SCPs)* mathematically, and how to represent the probability distributions that they are formulated on in such a way that we can use *weighted model counting (WMC)* to perform probabilistic inference. We then introduce a new representation language, SC-ProbLog, as a convenient way to model not only the complex probability distributions that result from the probabilistic networks on which we formulate the SCPs in this work, but also their associated constraints and optimisation criterion. Later in the chapter, we put SCPs and the methods we propose for modelling and solving them in the context of the existing literature on *stochastic SAT (SSAT)*, probabilistic programming, stochastic constraint programming and knowledge compilation for solving stochastic optimisation problems. Finally, we describe a number of typical problem settings that we use in later chapters to evaluate the SCP solving methods proposed therein. Parts of

this chapter are based on the following publication:

-  A.L.D. Latour, B. Babaki, A. Dries, A. Kimmig, G. Van den Broeck, and S. Nijsen. ‘Combining Stochastic Constraint Optimization and Probabilistic Programming — From Knowledge Compilation to Constraint Solving’. In: *Principles and Practice of Constraint Programming — 23rd International Conference, CP 2017, Melbourne, VIC, Australia, August 28 – September 1, 2017, Proceedings*, Springer. pp. 495–511, 2017.

## 4.1 Introduction

The main focus of this work is on solving SCPs such as the ones described in Section 1.1. This work was partially motivated by the following observations on the limitations of the existing SCP solving literature:

- Most publications on SCPs focus on specific types of problems: scheduling and planning problems, typically.
- Existing languages for modelling stochastic constraint optimisation problems are less suitable for modelling SCPs formulated on probabilistic networks.

To address the first limitation, we note that there is a rich literature on solving SCPs in the domains of scheduling and planning [7, 57, 78, 82, 110, 113, 150, 171], with methods proposed for solving problems in those domains, specifically. Tools like MiniZinc [130] and the *Advanced Interactive Multidimensional Modeling System* (AIMMS)<sup>1</sup> are very well-equipped to model these problems. We discuss some of these problems and solving methods in Section 4.4.

However, to the best of our knowledge, no such methods exist for conveniently modelling and solving the examples of SCPs described in Section 1.1. They are specified over a very different type of distribution than common in existing SCP solving systems: *probabilistic networks*, i.e., networks in which edges exist with a certain probability.

Given the range of different application domains that these examples cover, from marketing to governance to bioinformatics, we conclude that SCPs outside the domains of planning and scheduling are plentiful. We thus aim to extend the focus of SCP solving research to also include these kinds of problems.

Therefore, in this chapter we introduce a programming language that can be used to model these SCPs, and potentially many other SCPs. In later chapters, we introduce tools for solving the resulting models.

---

<sup>1</sup>Available at [www.aimms.com](http://www.aimms.com).

Addressing the second limitation, we propose to exploit the fact that in recent years, significant progress has been made in the development of *probabilistic programming languages*, as discussed in Sections 2.6 and 3.2.

These languages allow users to model probability distributions on probabilistic networks very efficiently, as they are particularly well-suited for modelling relational data. Until now, however, they have rarely been linked to *constraint programming (CP)*.

In this work, we expand DT-ProbLog [178], a probabilistic programming language designed for modelling optimisation problems that involve uncertainty, and which we described in Section 3.2. It is particularly suited for modelling optimisation problems on probabilistic networks, so we adapt it such that it can be used to formalise SCPs as well, adding support for hard constraints. We call the resulting modelling language *stochastic-constraint probabilistic Prolog*, or SC-ProbLog.

The remainder of this chapter is organised as follows. In Section 4.2 we first describe how to model SCPs mathematically. We then describe SC-ProbLog and how to model SCPs such that they can be communicated to a computer, in Section 4.3. We close this chapter with a description of typical examples of SCP problem settings, and a number of specific problems, formulated on real-world data, in Section 4.5. Finally, we conclude this chapter in Section 4.6.

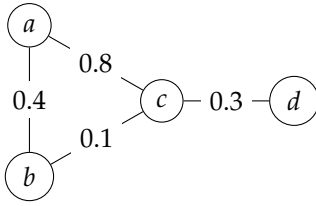
## 4.2 Modelling SCPs

Here we provide a concrete example of a problem instance for each of the two problem settings described in Section 1.1. These problem instances are formulated on the probabilistic networks shown in Figure 4.1. Then, we show how to model the associated probability distributions, such that we can use WMC to compute probabilities.

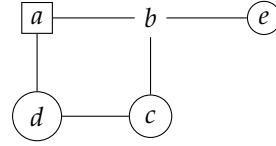
### 4.2.1 Modelling stochastic constraint (optimisation) problems

First, we define a problem instance for the spread of influence problem:

**Example 4.2.1** (Spread of influence: SCP). *Consider the network in Figure 4.1a. Nodes represent people. Edges represent probabilistic influence relationships, meaning that an individual  $u$  influences another individual  $v$  with probability  $p_{uv}$ , which labels edge  $(u, v)$ . We distribute free samples to a subset of the individuals, who can probabilistically influence other individuals to become customers as well. The objective is to maximise*



(a) A social network with four nodes representing Alexa, Behrouz, Claire and Daniël, and four undirected edges with mutually independent probabilities, representing their stochastic influence relationships.



(b) A power transmission grid with five nodes: one power producer (a), three power consumers (c, d and e) and one power transmitter (b). Edge probabilities depend on strategy.

**Figure 4.1:** Two examples of probabilistic networks.

the expected number of people who become our customer, given a limited number  $k$  to distribute free product samples to people in the network.

We make the simplifying assumption that influence relationships are mutually independent, meaning that whether persons  $u$  and  $v$  influence each other is independent of whether persons  $w$  and  $x$  influence each other (where  $x \neq u \neq v$ ). We also assume that once a person becomes our customer, they will never stop being our customer, such that they can become our customer at most once. As the problem setting in Section 1.1 describes, we assume that we distribute the free product samples only at one moment in time.

Given these assumptions, we model this problem as follows:

- With each node  $i$  in the network we associate a Boolean decision variable  $D_i \in \{\top, \perp\}$ , representing whether person  $i$  receives a free sample.
- We are interested in the events  $\Phi = \{\phi_a, \phi_b, \phi_c, \phi_d\}$ , where  $\phi_i$  denotes the event of person  $i$  being our customer.
- Our objective is to find a strategy  $\sigma$  that maximises the expected utility  $\sum_{i \in \{a,b,c,d\}} \rho_i \cdot P(\phi_i | \sigma)$ , 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  $c_i := 1$ .

Similarly, we also define a problem instance for the power grid reliability problem:

**Example 4.2.2** (Power grid reliability: SCP). Consider the network in Figure 4.1b. Here, each edge represents a power line  $(u, v)$  that has a probability  $p_{uv}$  of remaining intact during a natural disaster. By using our maintenance budget to reinforce power lines, we can increase the survival probability of those power lines. Our goal is to use our

budget  $\beta$  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 the survival probabilities of power lines are mutually independent, meaning that the survival or breakage of power line  $(u, v)$  does not influence the survival or breakage of power line  $(w, x)$  (with  $x \neq u \neq v$ ).

We model this problem as follows:

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

Note that in both examples, we fix  $c_i = \rho_i = 1$  for reasons of simplicity, but it is straightforward to use alternative values, as long as  $c_i, \rho_i \in \mathbb{R}^+$ .

## 4.2.2 Stochastic optimisation criteria

Observe that the two examples above each involve a stochastic *objective function*, rather than a stochastic *constraint*. In those examples, the constraint is a linear constraint on the expenses of the company that wants to use spread of influence to market their product or the power company that wants to do a maintenance project on its power lines. Problems like these occur often in real-world situations, where there may be a cost associated with setting a decision variable to *true*, and the user has a limited budget.

Recall our discussion of how to turn a constraint *optimisation* problem into a constraint *satisfaction* in Section 3.3.4. We can straightforwardly apply that principle here, by starting with the following two constraints:

$$\sum_{\phi \in \Phi} \rho_\phi \cdot P(\phi | \sigma) > \theta \quad \text{and} \quad \sum_{0 \leq i < |\mathbf{D}|} c_i \cdot D_i \leq \beta,$$

where  $\rho_\phi$  is the reward associated with  $\phi$  evaluating to *true*,  $\theta$  is initialised to 0,  $c_i$  is the cost of setting decision variable  $D_i \in \mathbf{D}$  to *true*, and  $\beta$  is the budget.



Following the procedure as described in Section 3.3.4, we then iteratively solve this *constraint satisfaction problem (CSP)*, updating the value of  $\theta$  every time we find a new solution.

### 4.2.3 Modelling probability distributions

In order to complete our models for the problems described in Examples 4.2.1 and 4.2.2, we must define the probability of events  $\Phi_i$ , given a strategy. As argued in Chapter 2, in this work we take a propositional WMC approach to representing probability distributions, modelling them first using the decision-theoretic probabilistic logic programming language DT-ProbLog, which is based on probabilistic logic programming language ProbLog. Crucially, ProbLog provides functionality to ground probabilistic logic programs (see also Section 2.6). While in practice, these groundings are immediately compiled into *decision diagrams (DDs)*, DT-ProbLog has functionality for grounding probabilistic logic programs into literal-weighted propositional formulae on decision variables and stochastic variables. For the sake of discussion, and for the scope of this subsection, we assume that programs are ground into these formulae rather than DDs. Note that this represents simply a different way of representing the same information, since DDs can be seen as summaries of truth tables of (literal-weighted) propositional formulae (see Section 2.4).

Note that we make one crucial assumption in both examples above: the probabilities associated with the edges in the networks are mutually independent. This allows us to straightforwardly map every edge to a single stochastic variable, and then compute probabilities using WMC as described in Section 2.2.3.

Recall from Section 2.2 that, under the WMC approach, the following holds:

$$P(\phi|_{\sigma}) = \sum_{\mu \in \mathcal{M}} \prod_{T \in \mu} W(T), \quad (4.1)$$

where  $\mu$  is a set of truth assignments to all stochastic variables in  $\mathbf{T}$ , such that  $\mu$  is a model of  $\phi|_{\sigma}$ ,  $\mathcal{M}$  is the set of all models of  $\phi|_{\sigma}$ ,  $T \in \mathbf{T}$  is a stochastic variable,  $W(T) := w_T$  if  $T = \top$  in  $\mu$ , and  $W(T) := w_{\bar{T}}$  if  $T = \perp$  in  $\mu$ .

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

**Example 4.2.3** (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  $\phi_d$  takes place in Figure 4.1a can then be modelled by a literal-weighted propositional formula that we already encountered in Section 2.5, which we repeat here, for convenience:

$$\begin{aligned} \phi_d(\mathbf{D}, \mathbf{T}) := & D_d \vee (D_c \wedge T_{cd}) \vee (D_b \wedge T_{bc} \wedge T_{cd}) \vee (D_a \wedge T_{ac} \wedge T_{cd}) \vee \\ & (D_b \wedge T_{ab} \wedge T_{ac} \wedge T_{cd}) \vee (D_a \wedge T_{ab} \wedge T_{bc} \wedge T_{cd}). \end{aligned}$$

This formula represents all the different situations in which Daniël 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  $\phi_d$  to happen is when Claire gets a free sample and has enough influence over Daniël to convince him to buy the product.

To define a distribution over the network, we associate a probability  $p(T_{ij})$  with each Boolean variable  $T_{ij}$  that this variable is true. We call  $T_{ij}$  a stochastic variable. The probability  $P(\phi_d|\sigma)$  is then defined as the sum of the probabilities of all the (logical) models of this formula, given the strategy. Given strategy  $\sigma := \{D_a := \top, D_b := \perp, D_c := \perp, D_d := \perp\}$  (where we give a free product sample to Alexa, but to nobody else), an example of a scenario that is a model for  $\phi_d|\sigma$  is  $\{T_{ac} = T_{cd} = \top, T_{ab} = T_{bc} = \perp\}$  (where Alexa convinces Claire, who convinces Daniël to buy our product), which has a probability of  $0.8 \cdot 0.3 \cdot (1 - 0.4) \cdot (1 - 0.1) = 0.1296$ .

**Example 4.2.4** (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  $p'_{uv} > p_{uv}$  if reinforced.
- With each power line  $(u, v) \in L$  we associate a survival probability  $\pi_{uv}$  that takes the following values:

$$\pi_{uv} := \begin{cases} p_{uv} & \text{if } D_{uv} = \perp; \\ p'_{uv} > p_{uv} & \text{if } D_{uv} = \top. \end{cases} \quad (4.2)$$

The possible worlds in which event  $\phi_d$  takes place in Figure 4.1b are defined by the propositional formula

$$\begin{aligned} \phi_d(\mathbf{D}, \mathbf{T}) := & (T_{ad} \vee (S_{ad} \wedge D_{ad})) \vee \\ & ((T_{ab} \vee (S_{ab} \wedge D_{ab})) \wedge (T_{bc} \vee (S_{bc} \wedge D_{bc})) \wedge (T_{cd} \vee (S_{cd} \wedge D_{cd}))). \end{aligned}$$

This formula represents all the different situations in which power consumer  $d$  will still be connected to a power 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. We need two stochastic variables to model the survival probability of each power line in the network: one to model the survival probability if it is reinforced ( $S_{uv}$ ), and one to model the survival probability if it is not ( $T_{uv}$ ).

**Table 4.1:** The weighted model count for  $\phi_{uv}|_{D_{uv}=\top} = T_{uv} \vee S_{uv}$ .

model	weight
$\{T_{uv} := \top, S_{uv} := \top\}$	$P(T_{uv} = \top) \cdot P(S_{uv} = \top)$
$\{T_{uv} := \top, S_{uv} := \perp\}$	$P(T_{uv} = \top) \cdot (1 - P(S_{uv} = \top))$
$\{T_{uv} := \perp, S_{uv} := \top\}$	$(1 - P(T_{uv} = \top)) \cdot P(S_{uv} = \top)$
$P(T_{uv} = \top) + (1 - P(T_{uv} = \top)) \cdot P(S_{uv} = \top)$	

In this model, we associate the following probabilities with variables  $T_{uv}$  and  $S_{uv}$ :  $P(T_{uv} = \top) = p_{uv}$ , and  $P(S_{uv} = \top) = (p'_{uv} - p_{uv}) / (1 - p_{uv})$ . Here, probability  $P(T_{uv} = \top) = p_{uv}$  is taken directly from the definition given in Equation 4.2. To see why we do not set  $P(S_{uv} = \top) = p'_{uv}$ , consider the following propositional formula that models the survival probability of a line  $(u, v)$ :  $\phi_{uv} = T_{uv} \vee (S_{uv} \wedge D_{uv})$ . Here, we associate  $T_{uv}$  with the stochastic survival of line  $(u, v)$  if that line is not reinforced, and  $S_{uv}$  with the stochastic survival of line  $(u, v)$  if it is. If we decide to not reinforce this line ( $D_{uv} := \perp$ ), the probability that  $\phi_{uv}$  evaluates to true (and thus that line  $(u, v)$  survives) is equal to  $P(T_{uv} = \top)$ . Now suppose that we do reinforce line  $(u, v)$ , by setting  $D_{uv} := \top$ . In this case, the probability that  $\phi_{uv}$  is true becomes  $P(T_{uv} = \top) + (1 - P(T_{uv} = \top)) \cdot P(S_{uv} = \top)$ , as demonstrated in Table 4.1. Therefore, if we want to model the probabilities as they are in Equation 4.2, we cannot set  $P(S_{uv} = \top) = p'_{uv}$ , but must instead set this probability to  $P(S_{uv} = \top) = (p'_{uv} - p_{uv}) / (1 - p_{uv})$ . This ensures that  $P(\phi_{uv} = \top \mid D_{uv} = \top) = P(T_{uv} = \top) + (1 - P(T_{uv} = \top)) \cdot P(S_{uv} = \top) = p_{uv} + (1 - p_{uv}) \cdot (p'_{uv} - p_{uv}) / (1 - p_{uv}) = p'_{uv}$ , which is exactly the probability as specified in Equation 4.2. Note that the need to perform this trick stems from the fact that we use a disjunction ( $\vee$ ) to model the possible survival of line  $(u, v)$  and not an exclusive-OR.

Consequently, for  $p_{uv} := 0.4$  and  $p'_{uv} := 0.875$  (values from the literature [61]), we get  $P(S_{uv} = \top) \approx 0.79167$ . For strategy  $\sigma = \{D_{ad} = D_{bc} = \top, D_{ab} = D_{bf} = D_{cd} = \perp\}$ , one example of a model for  $\phi_d|_\sigma$  is:  $S_{ad} = T_{ab} = S_{ab} = T_{cd} = S_{be} = \top, T_{ad} = T_{bc} = S_{bc} = S_{cd} = T_{be} = \perp$ , of which the probability is  $0.79167^3 \cdot 0.20833^2 \cdot 0.4^2 \cdot 0.6^3 = 7.44235 \cdot 10^{-4}$ .

## 4.3 SC-ProbLog

After formalising the problem in a mathematical model, the next steps are to represent this model in a way that is usable for a computer, and compile the relevant probability distributions into DDs. In particular, we want to use a computer to get from the mathematical models described in Examples 4.2.1 and 4.2.2 to *ordered binary decision diagrams* (OBDDs) or *sentential decision diagrams* (SDDs) that we can use to compute *weighted model counts* (WMCs).

In this section, we kill two birds with one stone by building on existing tools from the probabilistic logic programming literature. In Section 3.2 we described a probabilistic programming language that is particularly suited for modelling optimisation problems defined on probabilistic networks: DT-ProbLog [178]. Additionally, this language, because it is based on ProbLog [52], offers functionality to ground probabilistic logic programs into DDs that can be used for tractable weighted model counting.

DT-ProbLog, however, does not offer support for constraints. It would be very convenient if we could model the entire SCP using just one language. We therefore expanded DT-ProbLog into a new language: *stochastic constraint probabilistic Prolog*, or SC-ProbLog. Compared to DT-ProbLog, SC-ProbLog adds support for hard constraints on probability distributions. Additionally, where DT-ProbLog only supports maximisation problems, the syntax and semantics we added allows the user to specify whether they want the objective function to be minimised or maximised.

We illustrate how to use the SC-ProbLog language by showing how we can to model SCPs described in Examples 4.2.1 and 4.2.2 with SC-ProbLog programs.

**Example 4.3.1** (Spread of influence: SC-ProbLog). *Recall the DT-ProbLog program in Program 3.1, and notice how it matches the problem described in Example 4.2.1. We only have to adapt it slightly to turn it into the SC-ProbLog program shown in Program 4.1.*

**Program 4.1:** An SC-ProbLog program for the spread of influence problem.

---

```
% Background knowledge
1. person(alex).           person(claire).
2. person(behrouz).        person(daniel).

% Probabilistic relation facts
3. 0.4::dir(alex,behrouz).  0.8::dir(alex,claire).
4. 0.1::dir(behrouz,claire). 0.3::dir(claire,daniel).

% Relation rules
5. influences(X,Y) :- dir(X,Y).
6. influences(X,Y) :- dir(Y,X).
```

---

```

    % Decisions
7. ?::gets_free_sample(P) :- person(P).

    % Customer conversion rules
8. buys(X) :- gets_free_sample(X).
9. buys(X) :- influences(Y,X), buys(Y).

    % Constraint and optimisation criterion
10. { gets_free_sample(P) => 1 :- person(P). } k.
11. #maximise { buys(P) => 1 :- person(P). }.

```

---

Lines 1–9 are taken directly from Program 3.1. Note that we have not copied the lines that indicate the utility of the different events. Instead, they are incorporated in the constraint in line 10 and the objective function in line 11.

In the example above, lines 10 and 11 represent the syntax and semantics that we added to obtain SC-ProbLog. Here, we borrow the syntax from the answer set programming literature (see, *e.g.*, *Answer Set Programming* by Lifschitz [108]).

In particular, line 10 represents the constraint. It assigns a cost of 1 to the decision to give a person a free sample, indicated by the  $\Rightarrow$  in the head of the rule in the braces, where we assume that *not* giving a person a free sample has a cost of 0. The braces indicate that the costs (or utilities) within them must be summed. The  $k$  corresponds to the  $k$  in Example 4.2.1, and represents the upper bound (or threshold) on the sum of the utilities.

Line 11 represents the optimisation criterion. The syntax in the braces is the same as in line 10. However, we now indicate that the sum of the expected utilities associated with a person buying the product is to be maximised.

Note that, in the example above, the constraint is formulated over decision variables, while the optimisation criterion is formulated over predicates whose truth values depend on the values of stochastic variables, but this need not be the case. We could also add multiple constraints, or omit the optimisation criterion, but we do not support multiple optimisation criteria.

A key property of ProbLog (and therefore also of SC-ProbLog) is that the rules that are stated are not assumed to be mutually exclusive. For example, the rules in lines 8 and 9 could be probabilistic, meaning that there is a chance  $p_{fs}$  of turning someone into a customer by giving them a free sample, and a chance  $p_{infl}$  of turning them into a customer if they are influenced by someone who is already a customer. These two customer conversion processes are not mutually exclusive.

However, in the power grid reliability problem, we are dealing with probabilistic facts that are mutually exclusive: a power line is either reinforced, or it is not. Recall the probabilities that we associated with the stochastic variables

in Example 4.2.4, and that they stemmed from the choice to model the possible survival of a power line as a (non-mutually exclusive) disjunction instead of as an exclusive-OR. We can therefore use those probabilities directly in a ProbLog program, as we demonstrate in the next example.

**Example 4.3.2** (Power grid reliability: SC-ProbLog). *We model the power grid reliability problem described in Example 4.2.2 as follows:*

**Program 4.2:** *An SC-ProbLog program for the power grid reliability problem.*

---

```

% Background knowledge
1. power_line(a,b).      power_line(a,d).
2. power_line(b,c).      power_line(b,e).      power_line(c,d).
3. producer(a).          consumer(c).
4. consumer(d).          consumer(e).

% Decisions
5. ?::reinforce(X,Y) :- power_line(X,Y).

% Probabilistic facts
6. 0.79167::survives(X,Y) :- power_line(X,Y), reinforce(X,Y).
7. 0.79167::survives(X,Y) :- power_line(Y,X), reinforce(Y,X).
8. 0.4::survives(X,Y) :- power_line(X,Y).
9. 0.4::survives(X,Y) :- power_line(Y,X).

% Relations
10. connection(X,Y) :- survives(X,Y).
11. connection(X,Y) :- connection(X,Z), survives(Z,Y).
12. connected_to_producer(X) :- producer(Y), connection(X,Y).

% Constraint and optimisation criterion
13. { reinforce(X,Y) => 1 :- power_line(X,Y). } b.
14. #maximise { connected_to_producer(X) => 1 :- consumer(X). }.

```

---

We define directed power lines in lines 1 and 2, and then define the power producers and the power consumers in lines 3 and 4. Line 5 represents the decision variables in this problem: one for each power line.

Lines 6–9 have two purposes: they make the directed power lines from lines 1 and 2 undirected, and they model the different survival probabilities of those power lines, depending on whether they have been reinforced to make them strongly and less likely to break. Note that lines 6 and 7 correspond to the  $S_{uv}$  variables as described in Example 4.2.4, while lines 8 and 9 correspond to the  $T_{uv}$  variables.

Lines 10–12 in Program 4.2 serve to define what it means to be connected to a power producer. Line 13 associates a cost of 1 with reinforcing a power line and defines an upper bound of  $b$  on the sum of these costs. Finally, line 14 associates a reward of 1 with each

power consumer that is connected to at least one power producer, which is a probabilistic fact, and specifies that the sum of the resulting expectations is the value that we want to maximise.

In the example above, the first 12 lines use the DT-ProbLog functionality. Lines 13 and 14 represent the new functionality introduced in SC-ProbLog: a constraint and an optimisation criterion, much like lines 10 and 11 in Program 4.1.

Now that we have modelled the problems described in this chapter using SC-ProbLog, we can rely on ProbLog’s technology to *ground* these programs to obtain the OBDDs or SDDs that represent the (stochastic) events of interest. In the next chapter, we describe how we can use these DDs representations of probability distributions to build a fast pipeline for solving SCPs. Before that, however, we first provide an overview of existing work that relates to SCP solving or the methods we propose to use in our SCP solving pipelines.

## 4.4 Related work

We now give a brief overview of work that is related to SCPs and their solving methods. Specifically, we first highlight a number of problems known from the literature that are very closely related to either checking if a stochastic constraint is satisfied, or to maximising a stochastic optimisation criterion. Since the approach to *modelling* SCPs in this work is rooted in probabilistic logic programming, we then provide a brief overview of probabilistic programming paradigms. Then, because our methods *solving* SCPs is mostly based on CP techniques, we discuss existing literature on stochastic constraint programming. We end this section with a brief discussion of other work that exploits knowledge compilation techniques for solving SCP-like problems.

### 4.4.1 Stochastic satisfiability

Solving a stochastic constraint like the one in Equation 1.1, which we repeat here for convenience:

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

and maximising a stochastic optimisation criterion as described in Section 4.1 can each be seen as instances of SSAT, as defined in Chapter 27 of the Handbook of Satisfiability [16].<sup>2</sup>

---

<sup>2</sup>An earlier version of this problem was proposed by Papadimitriou, who called it a ‘game against nature’ [137].

SSAT in its most general form is defined over formulae of the following kind:

$$\psi(\mathbf{X}) := Q_1 X_1 \cdots Q_{|\mathbf{X}|} X_{|\mathbf{X}|} \phi(\mathbf{X}), \quad (4.3)$$

where  $Q_i \in \{\exists, \forall\}$  represent the quantifiers and  $X_i \in \mathbf{X}$  the corresponding variables. The quantifier  $\forall$  indicates that the corresponding variable is ‘randomly’ quantified, meaning that this variable takes the values  $\top$  or  $\perp$  with a certain probability, independently of the other variables. While in the general case, the order of the quantifiers in the prefix of  $\psi$  is arbitrary, for this work only orders in which all the existentially quantified variables come first in the prefix, are relevant. Hence, below we only discuss settings of this kind, and we omit all discussion of settings in which the order of quantifiers is different.

To see how the SSAT problem connects to solving stochastic constraints like the one in Equation 1.1, we also distinguish a specific decision version of SSAT.

**Definition 4.4.1 (E-MAJSAT).** *Given a formula  $\psi(\mathbf{D}, \mathbf{T})$  as defined in Equation 4.3, where all the existentially quantified ( $\mathbf{D}$ ) variables come first in the prefix and the randomly quantified variables ( $\mathbf{T}$ ) take the value  $\top$  with a given rational probability  $0 \leq p_X \leq 1$ , and given a rational threshold value  $0 \leq \theta \leq 1$ , is there an assignment  $\sigma$  to the existentially quantified variables  $\mathbf{D}$ , such that*

$$P(\psi|_\sigma = \top) > \theta? \quad (4.4)$$

Here,  $P(\cdot)$  indicates a probability and  $\psi|_\sigma$  is the residual formula obtained by removing all existentially quantified variables from the prefix of  $\psi$  and substituting the existential variables in  $\phi(\mathbf{D}, \mathbf{T})$  by their truth values as specified by  $\sigma$ .

Thus, the *exists-majority SAT (E-MAJSAT)* problem asks if there *exists* an assignment  $\pi$  such that the probability that  $\psi$  is satisfied exceeds a certain threshold value  $\theta$ .

A well-known special case of E-MAJSAT in which  $\mathbf{D} = \emptyset$  (and thus all variables are randomly quantified), for each  $T \in \mathbf{T}$  we have  $p_T = 1/2$  and  $\theta = 1/2$ , is known in the literature as the *majority SAT (MAJSAT)* problem [16, 137]. This MAJSAT problem is known to be complete for the *probabilistic polynomial time (PP)* complexity class [16]. Recall the definition of the class  $\mathcal{PP}$  in Definition 2.2.8, and note how it indeed loudly echoes the task of the MAJSAT problem.

Note that computing the exact success probability of  $\psi$ , and thus *counting* the number of solutions, is a WMC task (as described in Section 2.2.3), and is  $\#\mathcal{P}$ -complete [155]. It is not hard to see that evaluating if Equation 1.1 is satisfied for a given strategy  $\sigma$  can be seen as a generalisation of the MAJSAT problem, in which  $0 \leq p_X \leq 1$  and  $0 \leq \theta \leq 1$  can take arbitrary, rational values.



Since in this work, we do not only solve stochastic constraints, but also deal with stochastic optimisation criteria, we also define the following variant of SSAT:

**Definition 4.4.2** (Functional E-MAJSAT). *Given a formula  $\psi(\mathbf{D}, \mathbf{T})$  as defined in Definition 4.4.1, which assignment  $\sigma$  of truth values to the existentially quantified (decision) variables in  $\mathbf{D}$  maximises  $P(\psi|_{\sigma} = \top)$ ?*

The solution to a functional E-MAJSAT problem is the optimal assignment  $\sigma^*$ .

We finally point to Littman *et al.*'s *extended* version of SSAT: XSSAT [111]. This problem generalises the SSAT problem by, aside from existentially quantified variables and randomly quantified variables, also allowing universally quantified variables, in arbitrary orders. A formal definition of this problem is outside the scope of this work.

#### 4.4.2 Probabilistic programming

In Section 2.6 we briefly described ProbLog [52] and motivated why we use its decision-theoretic version DT-ProbLog [178] as a basis to build SC-ProbLog on. However, ProbLog is not the only probabilistic programming languages that we could have chosen. We now briefly discuss alternative languages and their uses. For an extensive overview of the probabilistic logic programming literature, we refer the reader to De Raedt & Kimmig's recent survey [51].

As we mentioned in Section 2.3, a popular method for representing probability distributions, is Bayesian networks (BNs) [140]. One of the first languages that extended Prolog to include probabilities, was Poole's *probabilistic Horn abduction (PHA)* language, designed as a representation language for Bayesian networks [146]. PHA assigns probabilities to facts, and computes these probabilities by generating mutually exclusive *explanations* for these facts. Because these explanations are disjoint, their individual probabilities can simply be summed to obtain the probability of the fact that they explain. Dependencies are modelled by inventing new hypotheses. Sato's symbolic-statistical modelling language PRISM [161, 162], Muggleton's stochastic logic programs [129] and Poole's independent choice logic [147, 148] impose similar constraints on which facts can be true at the same time.

Recall from Section 4.3 that ProbLog, and languages derived from it, do not impose such constraints, as is evidenced by the fact that in the spread of influence example of Example 4.3.1 there are multiple ways of converting somebody into a customer, that do not need to exclude each other.

An interesting proposal for unifying the representations languages of BNs and propositional logic are Kersting & De Raedt's *Bayesian logic programs* [93, 94].

BNs can be seen as an extension of propositional logic, adding quantitative information (probabilities) to the qualitative information (local influences between random variables). As such, they inherit the limitations of propositional logic, particularly its rigidity and inability to represent a variable number of objects in the problem encoding, or a variable number of relations between those objects. This is something that probabilistic logic programming is much more suited for. *Bayesian logic programs* is a representation language that generalises both BNs and probabilistic logic programs, separating the qualitative information from the quantitative information.

Finally, we mention another interesting paradigm related to probabilistic logic programming, is that of *probabilistic databases* [170], with applications in data retrieval and reasoning over the web. Specifically, Fuhr’s *probabilistic Datalog* [68], designed specifically as a language for such information retrieval, is very similar to ProbLog in how it attaches probabilities to facts and rules. Its reasoning powers, however, are limited compared to ProbLog’s [52].

### 4.4.3 Stochastic constraint programming

This work is also closely related to *chance constraint programming* [31] and *probabilistic constraint programming* [172]. In particular, the problem we consider can be framed as a *single-stage stochastic constraint satisfaction problem (SCSP)* [181].

As briefly discussed in Section 1.1, we limit ourselves in this work to single-stage optimisation problems, because of restrictions on the probability distributions required by the SCP solving pipeline that we present in Chapter 6. We briefly mention existing work on multi-stage stochastic optimisation problems, for the interested reader.

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 the 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 [7, 113], and can be modelled as special cases of the SSAT problem [16], where blocks of existentially quantified and randomly quantified variables alternated in the prefix of the propositional formula (see also Section 4.4.1). The authors of stochastic MiniZinc [150] implemented a generic framework to encode multi-stage SCPs in a solver-agnostic manner.

Note that, while SCPs are certainly related to constraint optimisation under *soft constraints* [18], we impose *hard constraints* on probability distributions, and thus refrain from a further discussion of soft constraints. We also stress that in this work, we focus on finding exact solutions. There is an extensive literature on approximation methods, see, *e.g.*, [23, 35, 121, 143, 185], the discussion of which is outside the scope of this work.

Mixed networks [125] essentially combine *probabilistic graphical models*, which are used to model probability distributions, and *constraint networks*, which are used to express constraints. The authors define the *constraint (or conjunctive normal form (CNF)) probability evaluation (CPE)* task for a problem that can be specified on a *belief network* (a type of probabilistic graphical model) and a set of constraints, which are expressed in a CNF. Their goal is to find the probability distribution of the belief network, for all models of the CNF. As such, it corresponds to computing Equation 2.7 for all possible queries, and is thus closely related to SCP solving. Since we use a probabilistic propositional framework to represent our models, we consider a detailed description of probabilistic graphical models, although they are conceptually somewhat related to our framework, to be outside the scope of this work.

As briefly mentioned in Section 1.1, our work distinguishes itself from earlier, more generic, stochastic constraint programming approaches because we explicitly use the structure of the encoding of the underlying probability distributions to speed up the solving process.

#### 4.4.4 Knowledge compilation for SCP solving

Pipatsrisawat and Darwiche use knowledge compilation to solve E-MAJSAT problems [145]. In their approach, all constraints are encoded together into one diagram, which can cause it to blow up, depending on the number and type of constraints that must be encoded. Additionally, by integrating all constraints into one representation, they lose information about the structure of those constraints. Constraint solvers typically exploit this information in dedicated constraint propagators, an option that is no longer available once all constraints are encoded into one diagram. Moreover, not all constraints can be (trivially) encoded into CNF, which limits the expressiveness of the approach. In this work, we study if another approach is possible.

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 [70, 77, 179]. By associating MDD arcs in such encodings with probabilities, one can sample

solutions to a constraint [141]. 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.

## 4.5 Problem settings

So far, we have been using the spread of influence problem and power grid reliability problem as running examples to illustrate our methods. In the subsequent chapters, we will use concrete instances of these problems to evaluate our methods. In addition, we will use problem instance from other domains, so we can evaluate our proposed pipelines on a variety of problem types. Therefore, in this section, we describe the problem settings that we consider in the experiments in Sections 5.3, 6.5 and 7.3. Where relevant, we also describe how we processed the input data sets to obtain individual problem instances for our experiments.

### 4.5.1 Theory compression or graph sparsification

The first problem setting that we consider in this work is one from the data mining literature [50]. We are given a network of genes, proteins (both represented by vertices) and their interactions (edges), where these interactions are probabilistic. Furthermore, we are given *knock-out pairs*: pairs of vertices for which knocking out one vertex leads to a positive or negative change in the expression level of the other vertex. Paths of interaction can explain the positive or negative effect of one vertex on another. Our goal is to obtain a sparser network that preserves the pairwise interactions we are most interested in, in order to better understand these interactions. This problem is known from the literature as a *theory compression problem* [50].

Let  $\Phi^+$  and  $\Phi^-$  be our sets of interest, where events  $\phi_{u \rightarrow v} \in \Phi^+$  represents a vertex pair  $(u, v)$  for which a knock-out of protein  $u$  leads to an observed positive change in the expression level of gene  $v$ , and similar for events in  $\Phi^-$ . We associate a decision variable  $d$  and a stochastic variable  $t$  with each edge in the network. Here, the decision variable represents whether or not we select the corresponding edge to be part of the network that we extract, while the stochastic variable represents the strength of the interaction of the vertices on which the edge is incident.

We use a gene-protein and protein-protein interaction network called the *Signalling-regulatory Pathway Inference (SPINE)* [133] network. The full network has 4 696 vertices that represent genes and proteins. It has a total of 5 568 directed

protein-gene edges, and 15 147 undirected protein-protein edges. The SPINE network provides probabilities for both the directed and undirected edges. We used Gephi’s implementation of the Louvain community detection algorithm [19] to extract communities of different sizes, containing different sets of positive and negative vertex pairs, because the full network is too large to handle by our methods. In the rest of this work, we refer to the problem instances from the SPINE network as **spine** instances.

In our experiments, we consider some variants of this problem, which are each combinations of an optimisation criterion and a constraint, where one of these elements involves an expectation and the other the cardinality of the solution:

**Variante 1:** Maximise expectation, upper bound on solution cardinality. Given a set of vertex pairs  $\Phi \in \{\Phi^+, \Phi^-\}$ , our aim is to maximise the expected number of pairs in this set in which there is interaction between the two vertices in the pair, while placing an upper bound on the number of edges we can pick for the extracted network:

$$\text{maximise } \sum_{\phi \in \Phi} P(\phi \mid \sigma), \text{ subject to } \sum_{D \in \mathbf{D}} D \leq k, \quad (4.5)$$

**Variante 2:** Minimise solution cardinality, lower bound on expectation. Here, the goal is to minimise the size of the network induced on the extracted edges, but to guarantee that the summed expected strength of interactions between the vertex pairs meets a certain lower bound:

$$\text{minimise } \sum_{D \in \mathbf{D}} D, \text{ subject to } \sum_{\phi \in \Phi} P(\phi \mid \sigma) \geq \theta, \quad (4.6)$$

where again  $\Phi \in \{\Phi^+, \Phi^-\}$  and  $\theta \in \mathbb{R}^+$  represents the lower bound on the expectation.

**Variante 3:** Maximise expectation, upper bound on another expectation. This is a setting in which we are less concerned with network size, but more with how ‘pure’ the extracted network is in its ability to explain the interaction between vertices from *one set of interest* only, and not the other:

$$\text{maximise } \sum_{\phi^+ \in \Phi^+} P(\phi^+ \mid \sigma), \text{ subject to } \sum_{\phi^- \in \Phi^-} P(\phi^- \mid \sigma) \leq \theta, \quad (4.7)$$

or with the roles of  $\Phi^+$  and  $\Phi^-$  reversed.

**Variante 4:** Maximise solution cardinality, upper bound on expectation. A slightly less intuitive setting, where we aim to filter out a specific proportion of the

interaction between the vertices in one set of interest:

$$\text{maximise } \sum_{D \in \mathbf{D}} D, \text{ subject to } \sum_{\phi \in \Phi} P(\phi \mid \sigma) \leq \theta, \quad (4.8)$$

where again  $\Phi \in \{\Phi^+, \Phi^-\}$  and  $\theta \in \mathbb{R}^+$  a lower bound on the expectation.

Finally, for each (community, variant) pair we determined a threshold  $k$  or  $\theta$  that yields a hard problem to solve. We provide more details in Section 5.3.

### 4.5.2 Spread of influence

This is the problem setting described in Example 4.2.1, and is known from the data mining literature [56, 92]. For our experiments, we relax some of the simplifying assumptions made in Example 4.2.1. 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. In this problem setting, we associate decision variables with the vertices of the network, and stochastic variables with both vertices and edges.

To generate problem instances, we took a directed multigraph that represents user interactions on Facebook [180]. The full network comprises 46 952 users (vertices) and 876 993 unweighted edges (wall posts). We then used Kempe et al.’s approach [92] to create weighted edges between users, by assigning a weight of  $1 - (1 - p)^n$  to an edge  $(u, v)$  if  $u$  posted  $n$  times on  $v$ ’s wall, with  $p = 0.1$ .

Additionally, we took the *high-energy physics collaboration* undirected network [131], which was used in earlier publications on viral marketing [92]. The full network has 7 610 authors (vertices) and 15 751 directed unweighted edges, which we turn into probabilities, again following Kempe et al.’s approach. If an edge  $(u, v)$  has weight  $n$ , where  $n$  is the number of times that author  $v$  cites author  $u$ , the edge gets a weight of  $1 - (1 - p)^n$ , where we choose  $p = 0.1$ . Note that for this specific network, we may not be interested in spread of influence for the purposes of word-of-mouth marketing. Rather, the spread of influence may refer to the spread of ideas, research interest or even research styles within a scientific community, by means of citation.

We used the Louvain community detection algorithm [19] to extract communities of suitable sizes. In this work, we refer to instances from these datasets as **facebook** and **hepth** instances, respectively.

Again, we distinguish several variants, similar to the ones described above:

**Variante 1:** Maximise expectation, upper bound on solution cardinality. This setting is the one described in Section 1.1 and Example 4.2.1, where we aim to maximise the expected number of eventual customers of our product, given a fixed budget with which we send a free product sample to  $k$  people in the given social network.

**Variante 2:** Minimise solution cardinality, lower bound on expectation. Here we have a requirement that the expected number of people who will eventually buy our product is at least  $\theta$ , while we minimise the number of free samples that we have to hand out to achieve this goal.

These are the only variants we consider, as the other ones do not make much sense in this problem setting. Again, for each (community, variant) pair we determined a threshold  $k$  or  $\theta$  that yields a hard problem to solve. We provide more details in Sections 5.3, 6.5 and 7.3.

### 4.5.3 Power grid reliability

This is the problem described in Example 4.2.2; 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 source vertices and the target vertices; these vertices are not paired. We wish to maximise the expected number of target vertices that can be reached from at least one of the source vertices. Moreover, where in the sparsification problem described above, setting a variable that represents an edge to *false* is interpreted as removing that edge from the graph, in the power grid reliability problem, its connection probability becomes lower, but not zero. Finally, the graphs in the power grid reliability problem are undirected rather than directed.

We take network models of European and North-American high-voltage power grids [183], extracted by GridKit<sup>3</sup>. We extract connected components from geographic regions (countries for the European network and states for the North-American network), making sure that they contain both source vertices (power producers) and target vertices (power consumers).

For the survival probabilities of the power lines that are or are not reinforced, we turn to the literature [61]. We associate a uniform survival probability with each reinforced power line of 0.875, which drops to 0.4 when it is not reinforced.

---

<sup>3</sup>Available at [github.com/bdw/GridKit](https://github.com/bdw/GridKit)

In this work we refer to instances from this problem set as **powergrid** instances. We only consider **Variante 1**-type problems in this work, where we assume that a country or state has a fixed budget for power line maintenance and aims to maximise the expected number of households that still have power after a natural disaster. We provide specifics about these instances in Sections 6.5 and 7.3.

#### 4.5.4 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. Note that FIM problems, like the problem setting described below, cannot currently fully be modelled using SC-ProbLog, and thus require us to combine different representation languages to model them.

A challenging problem of our times is the spread of fake news. Often-times, 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:  $\sum_{v \in V} P(\phi_v \mid \sigma_U) \geq \theta$ , where  $\phi_v$  represents the event that a user  $v$  believes or adopts a piece of fake news, and  $\sigma_U$  represents the ‘strategy’ in which all users in  $U$  are considered to be the initial distributors of that news. In words: the *collective* influence of the users in  $U$  is at least  $\theta$ . 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  $\kappa$ , 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  $w$ . Intuitively, we can think of the users in set  $W$  as social media followers of a fake news distributor  $u \in U$ . The cardinality constraint of  $|W| \geq \kappa$  then expresses the minimum following of  $u$ . This second constraint corresponds to a minimum support constraint over a transaction database in FIM (see, e.g., [2]).

We create this *transaction database*  $\mathcal{D}$  by including in it one transaction  $\tau$  per user  $v \in V$ . Here,  $\tau$  represents the set of other users who influence  $v$  directly. Thus, if fake news spreading user  $u \in U$  has a following of  $|W|$ , it means that  $u$  is present in  $|W|$  transactions in  $\mathcal{D}$ . We used the **facebook** [180] dataset to generate communities as described above, and formulated a fifth problem variant:



**Variante 5:** Lower bound on expectation, lower bound on support. We aim to identify the sets of users  $U$  (itemsets) such that  $U \subseteq \tau$  for at least  $\kappa$  individual transactions  $\tau \in \mathcal{D}$  (making them frequent), where each itemset  $U$  has a collective expected influence of at least  $\theta$ .

Hence, we combine the stochastic constraint from Equation 1.1 with a minimum support constraint, known from the FIM literature [164].

Note that, in all of the above example problems and problem settings, we are summing over probabilities  $P(\phi \mid \sigma)$  for different  $\phi \in \Phi$ . It is in this context that multiple-rooted DDs (as mentioned in Section 2.4), and thus multiple-rooted *arithmetic circuits* (ACs) (as mentioned in Section 2.5) are relevant.

We believe that the problem settings described above present a varied and relevant set of problems for us to test our methods on. Specifically, we believe the variety between the problem settings to be sufficient enough for us to evaluate if there are approaches that seem to be universally suited for solving SCPs, or if our methods may have more complementary properties when it comes to solving problems from these different domains.

## 4.6 Conclusion

In this chapter, we provided a basic introduction to SCPs and how to model them using our newly proposed language *stochastic constraint probabilistic Prolog*: SC-ProbLog. Additionally, we provided an overview of work that is related to SCPs, and to methods we use in this work to solve them, placing our modelling and solving methods in the context of existing work on stochastic satisfiability problems, probabilistic logic programming, stochastic constraint programming and the use of knowledge compilation for solving stochastic optimisation problems. Finally, we described a variety of problems and problem settings that we will use to evaluate our SCP solving methods in the next three chapters of this dissertation. As such, this chapter can be seen as a thorough and extensive introduction to the background required for remainder of this dissertation.