Optimal decision-making under constraints and uncertainty

Latour, A.L.D.

Universiteit Leiden

Optimal decision-making under constraints and uncertainty

Proefschrift

ter verkrijging van

de graad van doctor aan de Universiteit Leiden

op gezag van rector magnificus prof.dr.ir. H. Bijl,

volgens besluit van het college voor promoties

te verdedigen op dinsdag 13 september 2022

klokke 11:15 uur

door

Anna Louise Dorothea Latour

Promotiecommissie

Promotores:
Prof. Dr. H.H. Hoos
Prof. Dr. J.N. Kok (Universiteit Twente, NL)

Co-promotor:
Dr. S. Nijssen (Université catholique de Louvain, BE)

Overige leden:
Prof. Dr. C.M. Jonker
Prof. Dr. K. Kersting (Technische Universität Darmstadt, DE)
Prof. Dr. H.C.M. Kleijn
Prof. Dr. P. Lucas (Universiteit Twente, NL)
Prof. Dr. A. Plaat
Prof. Dr. P. Stuckey (Monash University, Melbourne, AU)

This research was funded by the Netherlands Organisation for Scientific Research (NWO).

SIKS Dissertation Series No. 2022-25
The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

ISBN/EAN: 978-94-6419-574-3

Typeset using LaTeX
Printed by Gildeprint

Abstract

Scientists, policy makers and individuals must take decisions while dealing with uncertainty on a daily basis. Often-times, there also are limitations on the number or kind of decisions we can take. Especially when the stakes are high, we want to take an optimal decision: one that has, in expectation, the best possible outcome, according to some predefined metric of success.

Uncertainty can, for example, stem from randomisation in algorithms or human behaviour. Even if you follow someone on Twitter, does their post show up on your timeline? And if it does, do you read it? Alternatively, uncertainty may, for example, originate in the randomness of nature. When an earthquake strikes, which power lines in an electric grid survive this disaster? And how many households lose power as a result? In any application, uncertainty can stem from the practical problem that some things are hard to accurately measure or quantify.

We must also deal with constraints. In the realm of marketing, there is a limit on how many people we can reach with a single ad campaign or newspaper article. In the context of maintenance, there is a limit on how much money we can spend on making our electric grid strong and resilient.

The examples above all involve relationships between entities. People may have a (mutual) follower relationship with each other on social media. Power grids connect power stations to households. Because of the uncertainty in these examples, we can view these relationships as *probabilistic relational data*, which we

can model using *probabilistic networks*.

In this work, we develop new methods for exact optimal decision-making under constraints and uncertainty, specifically for problems that involve some form of probabilistic relational data. These *stochastic constraint (optimisation) problems (SCPs)* are hard to solve. Uncertainty about the future means that we must consider many different, and possibly overlapping, scenarios, to assess the effects of decisions. In addition, the number of potential strategies (sets of decisions) that we can adopt for solving the problem can grow exponentially with the number of individual decisions in the problem. In general, SCPs are $\mathcal{NP}$-hard problems.

We aim to design solving tools that are convenient to use, and thus accessible to people with limited programming skills, general enough to support a wide range of SCPs, and fast enough to be practical. In order to achieve these goals, we propose to build on technology from the probabilistic inference literature, and the *constraint programming (CP)* and *mixed integer programming (MIP)* literature.

In particular, we present a new probabilistic logic programming language, SC-ProbLog, especially designed for modelling SCPs formulated on probabilistic networks, and based on ProbLog. We show how to formulate constraints on the probability distributions induced by these networks, and how to model these constraints such that they can be solved by a CP or MIP solver. Here, we make use of existing tools to create *decision diagram (DD)* representations of probability distributions, which support tractable probabilistic inference. In doing so, we identify specific properties of these DDs that we can exploit to speed up the inference process, and thus the SCP solving process.

We take a modular approach to building our SCP solving pipelines. This ensures that, for each subtask, we can use any relevant state-of-the-art tools, which helps to keep the pipeline up-to-date with the latest developments, without having to (re)implement the latest techniques ourselves or having to integrate them into a monolithic design.

We do this following the paradigm of *programming by optimisation (PbO)*, implementing alternative design choices for different parts of the pipeline. The resulting pipelines are highly configurable, meaning that choosing the right components, and the right parameters for those components, for a specific type of problem may be difficult. We therefore use *automated algorithm configuration (AAC)*, to not only evaluate the performances of the SCP solving pipelines in a fair manner, but also find which parameter settings work well for problems from specific application domains.

The work presented in this dissertation advances the state of the art in SCP solving by proposing a new programming language to model SCPs, demonstrat-

ing how off-the-shelf CP and MIP technology can be leveraged for fast SCP solving, and by proposing an efficient propagation algorithm for stochastic constraints. We take a PbO-based approach, being, to the best of our knowledge, the first to do so in the field of exact probabilistic inference. We demonstrate the effectiveness of our methods on problems that are known in the data mining literature, including spread-of-influence problems and *frequent itemset mining (FIM)* problems.

Beknopte samenvatting

Wetenschappers, beleidsmakers en individuen maken dagelijkse beslissingen op basis van onzekerheden en kansen. Vaak zijn er ook beperkingen op het aantal en het type beslissingen dat we kunnen nemen. Zeker als de belangen groot zijn, willen we graag dat de uiteindelijke beslissing die we nemen, optimaal is. Een optimale beslissing is een beslissing die naar verwachting de beste uitkomst heeft, volgens een bepaalde maatstaf van succes.

Kansen in een beslissingsprobleem kunnen bijvoorbeeld komen door willekeur in algoritmes of onzekerheid over menselijk gedrag. Als je iemand volgt op Twitter, zie je hun tweets dan in je timeline? En als je een tweet ziet, stop je dan om te lezen, of scroll je verder? De probabilistische component van een beslissingsprobleem kan zijn oorsprong ook hebben in de chaos en willekeur van natuurlijke processen. Welke hoogspanningskabels blijven heel als ze getroffen worden door een aardbeving? Hoeveel huishoudens komen dan zonder elektriciteit te zitten? In alle toepassingen vinden we ook onzekerheden die veroorzaakt worden door het praktische probleem dat sommige dingen lastig exact te meten of the quantificeren zijn.

Daarnaast moeten we ons vaak houden aan bepaalde beperkingen. In een reclamecampagne is er een limiet aan het aantal mensen dat je kunt bereiken via een advertentie in de krant, omdat niet iedereen die krant leest. We hebben ook geen oneindig budget voor het versterken van onze energie-infrastructuur.

De bovengenoemde voorbeelden bevatten ook relaties tussen entiteiten. Mensen kunnen elkaar volgen op sociale media. Elektriciteitsnetwerken verbinden energiecentrales met huishoudens. De onzekerheid in de relaties in de bovengenoemde voorbeelden maakt dat we die relaties kunnen zien als *probabilistische data*, die we kunnen modelleren met *probabilistische netwerken*.

In dit proefschrift presenteren we nieuwe methodes voor het exact oplossen van optimalisatieproblemen waarin we beslissingen moeten nemen over onzekere, relationele data, en relevante beperkingen moeten respecteren. Deze *stochastic constraint (optimisation) problems (SCPs)* zijn moeilijk op te lossen. Om exact te kunnen te redeneren over het effect van onze beslissingen in een onzekere toekomst, moeten we alle mogelijke, en vaak deels overlappende, scenarios in acht nemen. Het feit dat het aantal combinaties van beslissingen dat we kunnen nemen exponentieel kan groeien met het totale aantal individuele beslissingen, maakt het nog lastiger. In het algemeen zijn SCPs $\mathcal{NP}$-volledig.

Ons doel is nieuwe algoritmes te ontwikkelen die gemakkelijk zijn toe te passen, en daardoor toegankelijk voor een grote groep potentiële gebruikers, inclusief gebruikers met beperkte programmeerervaring. Daarnaast moeten onze methodes algemeen genoeg zijn om een breed scala aan verschillende SCPs op te kunnen lossen, en dat snel genoeg doen om van praktisch nut te zijn. Om deze drie doelen te kunnen bereiken, bouwen wij verder op bestaande methoden voor het redeneren over kansen, en op technologie die helpt om te zoeken door verschillende combinaties van beslissingen: *constraint programming (CP)* en *mixed integer programming (MIP)*.

We presenteren een nieuwe probabilistisch-logische programmeertaal, SC-ProbLog, die speciaal ontwikkeld is om SCPs mee te modelleren. Deze taal is gebaseerd op ProbLog, een programmeertaal die bijzonder geschikt is om probabilistische relaties tussen entiteiten te modelleren. We laten zien hoe we beperkingen op de kansverdelingen die voortkomen uit probabilistische netwerken kunnen formuleren, en hoe we deze beperkingen kunnen modelleren zodat ze kunnen worden opgelost door CP of MIP software. We maken gebruik van bestaande algoritmes om *beslissingsdiagrammen* te genereren die een representatie vormen van deze kansverdelingen. Deze beslissingsdiagrammen helpen ons om efficiënt te redeneren over kansen en kansverdelingen, tijdens het oplossen van SCPs.

Onze oplosmethoden zijn allemaal modulair van aard. Hierdoor kunnen we voor elke deeltaak de best beschikbare relevante software gebruiken. Dit helpt om onze pijplijnen voor het oplossen van SCPs actueel te houden met de laatste ontwikkelingen, zonder dat we de laatste technieken steeds zelf moeten (her)implementeren, of moeten integreren in een monolitisch software-ontwerp.

We volgen hierbij het principe van *programmeren-door-optimalisatie*, en implementeren veel alternatieve oplossingen voor elke deeltaak in de pijplijnen, die hierdoor zeer configureerbaar zijn. Door deze flexibiliteit is het soms lastig om de juiste component voor elke deeltaak en de juiste instellingen voor elk component te vinden voor het oplossen van een specifiek type probleem. Daarom maken we gebruik van technieken die het configureren van onze algoritmes automatiseren. Dit doen we niet alleen om de kwaliteit van onze methodes op een eerlijke en consequente manier te toetsen, maar ook om te achterhalen welke instellingen goed werken voor welk soort problemen.

Ons werk verlegt de grenzen van de al beschikbare technologie voor het oplossen van SCPs door een nieuwe programmeertaal te introduceren voor het modelleren van SCPs, door te laten zien hoe de beschikbare CP en MIP software gebruikt kan worden om SCPs snel op te lossen, en door nieuwe algoritmes te introduceren voor het oplossen van problemen met beperkingen op kansverdelingen. Hierbij kiezen wij een programmeren-door-optimalisatie-aanpak, en zijn, voor zover wij weten, de eersten die dat doen in het vakgebied van probabilistisch redeneren. We demonstreren de effectiviteit van onze methodes op problemen uit de dataminingliteratuur, in het bijzonder op problemen rondom mond-op-mond reclame en *frequent itemset mining (FIM)*.

Acknowledgements

As a Master student, I once asked Siegfried if I should use “I” or “we” in my thesis. After all, the words were written by me, but we did the research together. He explained that, even if you are the sole author of a text, you should always write “we”, because no piece of research ever is the product of just one person. I believed him then, but the experience of being a PhD candidate drove the lesson home, never to be forgotten. In these few pages, however, I am truly “I”, and use them to thank those that make us “we”.

First, I thank my supervisor, Siegfried. His relentless hard work to help me finish our first paper (and the later ones) was motivating and made me feel like we were a team. His patient mentorship made me comfortable admitting my mistakes, so he could help me fix them. He taught me how to answer scientific questions, and helped me overcome adversity and failure.

I also thank my promotor, Holger, who adopted me into his research group when I was already many months into my PhD, and helped me arrange a research visit to the University of Toronto. Without him, my attitude towards algorithm development and my opinion on how it should be done, would not have been the same. In addition, he taught me the one-liner that informs how I look at any scientific research, be it my own or someone else’s: be a benevolent critic.

I thank my other promotor, Joost, with whom I started the PhD journey. He has been an optimistic and trusting mentor to me, which was very comforting.

When it mattered, he was always quick to respond and asked those questions that I had not yet thought of.

I also thank my advisor Fahiem for welcoming me at the University of Toronto and opening my eyes to new avenues of science and to new approaches to problem-solving.

Next, I thank the members of my doctorate committee for reading this dissertation, deliberating its scientific value, and giving me valuable feedback to improve its contents.

I am also grateful to the Netherlands Organisation for Scientific Research (NWO) for funding my research, to Google for awarding me a scholarship, and to the University of Toronto for funding my research visit.

I am grateful to my co-author Behrouz, who was the first person who made me feel comfortable talking about my own research, by being genuinely curious in a completely honest and non-intimidating manner. He is a good coder and a great friend.

I thank my other co-authors, Anton, Angelika, Guy, Daniël, Marie and Jeroen for all their hard work, their critical questions, their smart insights, our fruitful discussions, and all the jokes that made the pill of late-night writing easier to swallow.

This dissertation would have been of lower quality without the feedback from reviewers on earlier presentations of the work that we present here. They made our arguments sharper, our results more complete, and our science better. I am grateful to the anonymous reviewers who were our benevolent critics.

I am grateful to the system administrators who built and maintained the servers on which I ran my experiments, and to the administrative staff who helped me navigate bureaucracy. I thank the cleaners, concierges, receptionists, security personnel, mess hall employees and others who made my workspaces comfortable and safe.

My journey as a PhD candidate brought me to three different universities, each populated with an inspiring group of scientists. I thank my co-workers from Université catholique de Louvain for our discussions in the "cafet" and on Slack, and for the very best social activities that academia has to offer. I thank my co-workers from Leiden University for the games of Codenames that we played in the FooBar, for the afternoon walks, and for all the moral and practical support. I am grateful to my co-workers from the University of Toronto for our lunches, the games of Hanabi we played over Zoom, and the updates on the well-being of office plants.

I thank my former co-workers from KU Leuven, for keeping in touch with

me and inviting me to lectures, defences and drinks. I am also grateful to the mentors that I found inside and outside academia, who made me stronger, wiser and kinder. I thank my paranimphs Jessa and Behrouz for standing by my side, now and always.

Finally, I thank my friends and family. This dissertation would not have existed without their patience and support. They cheered me on and cheered me up. They knew me before I did. They were not scared away by my flaws, but reminded me of my talents. Not all of them are with us any more, but all of them are with me. I am grateful to have them in my life.

Contents

List of symbols

$\bot$	Truth value *false* or 0.
$\top$	Truth value *true* or 1.
c_i	Cost (or non-negative utility) associated with an event ϕ_i happening.
$\mathbf{D}$	Set of decision variables.
D	A decision variable with Boolean domain, such it can be assigned the value *true* or *false*.
Δ	A decision diagram such as an OBDD or SDD, or the root of such a diagram.
k	Threshold value for cardinality constraint, represents a bound on the number of decision variables that can be set to *true*.
κ	Minimum support threshold for CoverSize constraint in *frequent itemset mining (FIM)* problems.
Φ	Set of interest: a set of stochastic events that are of interest to a user.
ϕ	An event in a stochastic constraint optimisation problem. Abusing notation, we use this symbol both to indicate the event in an abstract sense, and to denote the propositional formula that models the event.

π A (partial) variable assignment $\pi : \mathbf{X} \mapsto \{\bot, \top\}$ that assigns Boolean truth values to the variables in $\mathbf{X}$.

$\langle \phi, W \rangle$ A literal-weighted formula or potential. The propositional formula ϕ describes the structure and relations in the model. The weight function W assigns appropriate weights to literals, such that we can use *weighted model counting (WMC)* to compute probabilities.

ρ_i Reward (or non-negative utility) associated with an event ϕ_i happening.

$sc(\phi)$ The scope of a formula ϕ is the set of variables that occur in ϕ.

σ A (partial) variable assignment $\sigma : \mathbf{D} \mapsto \{\bot, \top\}$ that assigns truth values to the Boolean decision variables $\mathbf{D}$. We call this a (partial) *strategy*.

$\mathbf{T}$ Set of stochastic variables.

T A stochastic variable with Boolean domain. Takes value *true* with probability p_T and value *false* with probability $1 - p_T$.

$\mathcal{T}$ A vtree or the root of that vtree.

θ Threshold value for stochastic constraint, represents a bound on an expectation.

v A (partial) variable assignment $v : \mathbf{T} \mapsto \{\bot, \top\}$ that assigns truth values to the Boolean stochastic variables $\mathbf{D}$. We call this a *possible world* or *scenario*.

List of acronyms

#SAT problem of counting the number of models of a propositional formula

AAC automated algorithm configuration
AC arithmetic circuit
ADT affine decision tree
ASP answer set programming

BDD binary decision diagram

CDCL conflict-driven clause learning
CNF conjunctive normal form
CP constraint programming
CSP constraint satisfaction problem

DAG directed acyclic graph
DD decision diagram
d-DNNF deterministic decomposable negation normal form
DNF disjunctive normal form

E-MAJSAT exists-majority SAT

FIM frequent itemset mining

GAC generalised arc consistency

MAP maximum a posteriori
MARG marginal distribution
MEU maximum expected utility
MILP mixed integer-linear programming
MIP mixed integer programming
MMAP marginal MAP
MPA maximum probability assignment
MPE most probable explanation

NNF negation normal form

OBDD ordered binary decision diagram

PAR10 penalised average runtime with penalty factor 10
PbO programming by optimisation
PLP probabilistic logic programming
PMC projected model counting

SAT Boolean satisfiability problem
SCMD stochastic constraint on monotonic distributions
SCP stochastic constraint (optimisation) problem
SCPD stochastic constraint on probability distributions
SCPMD stochastic constraint (optimisation) problem on monotonic distributions
SDD sentential decision diagram
sd-DNNF smooth deterministic decomposable negation normal form
SMP single mixed path
SRL statistical relational learning
SSAT stochastic SAT
StarAI statistical relational artificial intelligence

WMC weighted model count

XSSAT extended SSAT

List of Figures

List of Tables

1

Introduction

> This is where we've filled ourselves up with so many questions that they're starting to overflow and become answers.
>
> Sir Terry Pratchett

1.1 Motivation

In business, governance, science as well as in our daily lives, we often have to solve problems that involve decision making under constraints and uncertainty. Examples of these problems arise in a diversity of domains, such as:

- planning (*e.g.*, decide when to invest in which company or product, to maximise your return on investment) [7],

- scheduling (*e.g.*, finding rosters for nurses that honour their preferences as much as possible and can deal with a stochastic number of daily patients) [181],
- production planning (*e.g.*, deciding how many books to print in each quarter of the year, to minimise storage costs but have enough to satisfy customers) [181],
- vehicle routing (*e.g.*, deliver all the packages ordered by people who are under sheltering-in-place during a pandemic as efficiently as possible, while dealing with stochastic demand) [150], and even
- bioinformatics (*e.g.*, model stochastic protein-protein and protein-gene interaction in a network that is as small as possible, but still explains the interactions as true to nature as possible) [50, 133].

Note that all the above examples do not just require us to make a decision under constraints and uncertainty, but also require some kind of *optimality* of that decision. We want to maximise profit or minimise cost. We want to maximise efficiency or minimise size. This is a common property of decision making problems in the world around us. In real-world problems, we often face multiple, possibly conflicting, objectives, such that solutions to a problem contain a certain trade-off with respect to these objectives. These trade-offs can often be captured in a single cost or utility value. We therefore only consider single-objective optimisation problems in this work.

Given the abundance of relational data in the areas mentioned above, many problems also involve probabilistic network data [50, 56, 61, 92]. Consider, for example, the following two problems.

Spread of influence This is a problem setting that is well-known from the data mining literature [56, 92]. We are given a social network in which the nodes represent people and the edges represent probabilistic mutual influence relationships, such as people following each other on social media. We are also given a marketing budget, which we can spend on providing free samples of our product to selected individuals in the social network. We then rely on a word-of-mouth marketing strategy, in which people who have tried our product may become our customers and try to convince their friends, family and acquaintances to also buy the product and become a customer. Depending on how much influence they hold over those relations, they are either successful in convincing a relation, or not. Our objective is to maximise the number of people in the social network who are convinced that they should buy our product, and are thus converted to customers. We can start this process by using our budget to provide free samples to a subset of the people in the network, which we distribute all at the same time. We

must therefore identify the subset of people that is small enough such that we do not exceed our budget (constraint), but maximises the expected number of people who will eventually be convinced to buy our product (optimisation criterion).

Power grid reliability This is an optimisation version of a problem known from the literature [61]. We are given a high-voltage power grid in which the nodes may either represent power producers (like nuclear power plants or solar farms), power consumers (power stations that transform the high-voltage power into lower-voltage power to distribute among buildings), and power transmitters (power stations that simply pass on the power, possibly splitting or merging lines). Power lines connect nodes to each other. With each power line we associate a probability that it will remain intact during a natural disaster like an earthquake, hurricane or storm surge. This probability may depend on its length or the terrain in which it exists. If too many lines are damaged, consumers may lose power. We are given a power line maintenance budget (constraint) and must decide which power lines we spend it on, such that we maximise the expected number of power consumers that are still connected to at least one power producer after a natural disaster (optimisation criterion). The budget must be assigned at a single moment in time.

Note that, because for each problem setting, we have to decide how to spend the budget in one moment in time, both these problem settings are considered *single-stage* constraint optimisation problems. These problems are instances of a general class of problems, known as *stochastic constraint (optimisation) problems (SCPs)*. SCPs have the following characteristics:

- They involve (Boolean) *decision* variables and (Boolean) *stochastic* (or *random*) variables.
- They involve reasoning over probability distributions.
- They involve constraints that limit the decisions we can make.
- They involve an optimisation criterion.

We provide a formal description of SCPs in Section 1.2.

For this work, we have chosen to limit ourselves to studying single-stage problems. As there are many real-world single-state problems, some of which were mentioned above, we believe that this choice to limit the scope is justified. We study how to solve SCPs efficiently, and use optimisations that are only possible in the single-stage setting.

The main goal of this work is to develop methods that find exact solutions to single-stage SCPs that are formulated on probabilistic networks, making sure that these methods strike a reasonable balance between:

Convenience, such that our methods and tools are accessible and easy to use, even for people with little or no background in programming or computer science.

Generality, such that our methods and tools can be used for solving a diversity of problem and problem types, from different application domains, and

Speed, such that our methods and tools solve SCPs fast enough to be practical.

In meeting the last requirement, we attempt to find algorithms with a low theoretical bound on the running time, as well as algorithms that have the potential to be faster in practice than others, even if their theoretical complexity is not lower. In order to find these exact solutions, we need to take three distinct steps:

Model the problem mathematically: Define the real-world problem and what constitutes a solution to that problem.

Specify the model in a computer-friendly manner: Use a suitable programming language to communicate the problem to a computer.

Solve the problem: Let an algorithm find the exact optimal solution.

In the literature, this second step is also referred to as 'modelling', so we will use that term in the remainder of this dissertation for different tasks, trusting that the context is enough to disambiguate, and clarifying wherever necessary.

The remainder of this chapter is organised as follows. In Section 1.2, we describe the stochastic constraint that is central to this work. Then, in Section 1.3 we briefly reflect on the hardness of SCPs and how we address the computational complexity of SCPs. We list and motivate our main research questions in Section 1.4, and specify which contributions we present in this dissertation in relation to these questions. This last section also serves as an outline to the remainder of this dissertation.

1.2 Stochastic constraints on probability distributions

The SCPs that we aim to solve in this work are all characterised by the presence of a stochastic constraint, similar to the ones studied by Papadimitriou [137] and

Littman *et al.* [111], or a stochastic optimisation criterion, similar to the ones studied by Walsh [181] and Van den Broeck *et al.* [178]. Specifically, in this work we study stochastic constraints of the following form:

$$\sum_{\phi \in \Phi} \rho_\phi \cdot P\left(\phi \mid \sigma\right) > \theta. \tag{1.1}$$

The sum represents an *expected utility*, in which Φ is a set of stochastic events that are of interest to us, $P\left(\phi \mid \sigma\right)$ represents the probability of an event ϕ happening, given a *strategy* σ over Boolean variables; and $\rho_\phi \in \mathbb{R}^+$ is a reward for this event. This constraint specifies a *lower bound* $\theta \in \mathbb{R}^+$ for an expected utility. Our methods can also be applied to stochastic constraints that impose an *upper bound* on an expected utility, instead.

Recall that SCPs are problems that may involve a stochastic *objective function*, rather than a stochastic *constraint*. We can straightforwardly employ constraints on probability distributions to solve minimisation or maximisation problems over expected utilities (under other constraints). We describe how this can be done in Chapter 3.

1.3 Computational complexity of SCPs

SCPs are difficult to solve exactly (*i.e.*, in a way that produces provably optimal solutions). Indeed, well-known instances of SCPs are shown to be $\mathcal{NP}$-hard [92], and solving SCPs exactly is $\mathcal{NP}$-hard in the general case. Specifically, exact SCP solving involves two components:

1. To evaluate the quality of a strategy σ, we have to compute $P(\phi \mid \sigma)$, which involves a counting task that is $\#\mathcal{P}$-complete in general [155, 176, 177].
2. We have to perform this evaluation a potentially exponential number of times, since the number of possible strategies for $|\mathbf{D}|$ Boolean decision variables is $2^{|\mathbf{D}|}$.

Informally, a $\#\mathcal{P}$-complete problem requires the counting of all solutions to a Boolean formula, of which it may have exponentially many, and can be harder than determining if a propositional formula has a solution, which is $\mathcal{NP}$-complete [38]. Thus, naïvely solving an SCP by enumerating all possible strategies and evaluating their score, which requires counting all the possible consequences of a strategy, is typically computationally impractical. We discuss $\mathcal{NP}$, $\#\mathcal{P}$ and other relevant complexity classes, as well as propositional formulae and the counting task referred to above, in Section 2.2.

There is earlier work on which we can build to address the two challenges listed above. In particular, *knowledge compilation* [48, 123, 165] techniques have been used to make probabilistic inference tractable. Similarly, *constraint programming (CP)* [154] or *mixed integer programming (MIP)* [25] have been used to model and solve constraint optimisation problems. In this work, we investigate whether and how these approaches can be combined to solve SCPs quickly, in theory or in practice.

The answer to that question is not immediately obvious. In order to reap the benefits of knowledge compilation, we have to encode the resulting representations of probability distributions in such a way that they can be communicated to a CP or MIP solver, otherwise the associated stochastic constraints cannot be solved by these solvers. We then have to choose how to model these constraints such that they can not only be solved quickly, but are also convenient and easy for the user to specify, and ideally generic enough to be implemented in a range of different solvers. The focus of this dissertation is on finding SCP solving methods with reasonable trade-offs between convenience, generality and speed.

1.4 Contributions

In this work we aim to answer four main research questions. We start by acknowledging that a new technology's success stands or falls on accessibility and ease-of-use. We therefore ask:

MRQ1 How can we conveniently model SCPs and specify them to a computer?

The contributions of this thesis with respect to **MRQ1** are as follows:

C1 We formulate a *stochastic constraint on probability distributions (SCPD)*, which allows us to model SCPs. As we will show in Section 4.2, this constraint can also be used to formulate stochastic optimisation problems.

C2 Next, we develop a new declarative programming language, *stochastic constraint probabilistic Prolog*, or SC-ProbLog, for programming SCPs. This language builds on earlier logic programming languages that allow for convenient modelling of probability distributions, and extends these with syntax and semantics for modelling constraints and optimisation criteria.

As we described in Section 1.3, it is not immediately obvious how we can use existing CP, MIP and knowledge compilation techniques to solve SCPs. We therefore ask:

MRQ2 How can we leverage CP, MIP and knowledge compilation technology to solve SCPs?

The contributions of this thesis with respect to **MRQ2** are as follows:

C3 We develop an SCPs solving pipeline, which takes as input an SCPs programmed in SC-ProbLog. It grounds the program and converts the resulting logic formulae into either *ordered binary decision diagrams (OBDDs)* or *sentential decision diagrams (SDDs)*. We either impose a stochastic constraint on the *decision diagram (DD)* representations of these probability distributions, or formulate an optimisation criterion that aims to maximise or minimise an expected utility that is computed from these probability distributions. We then *decompose* the OBDD or SDD into a CP or MIP model, which we solve using off-the-shelf solvers.

An important part of this work focuses on how encodings of probability distributions that are obtained through knowledge compilation interact with the CP and MIP solvers that we use to solve the SCPs that are formulated on those probability distributions. That part of this work is done to answer the following research question:

MRQ3 How can we leverage the properties of SDDs and OBDDs for faster SCP solving?

The contributions of this dissertation with respect to **MRQ3** are as follows:

C4 We observe that MIPs are much easier to solve if they are linear, rather than quadratic, and decomposed SDDs typically do not yield linear MIPs. Additionally, we observe that smaller SDDs yield smaller MIPs and show that smaller MIPs tend to take less long to solve than larger MIPs. To address these observations, we identify a class of SDDs that yield linear MIP decompositions and develop a minimisation algorithm for finding minimised SDDs that belong to this subset of SDDs.

C5 We show that CPs solvers cannot guarantee *generalised arc consistency (GAC)* in a naïve decomposition of stochastic constraints on OBDD representations of probability distributions. This results in the CP solver potentially searching a part of the search space that does not contain any feasible solutions, and thus wasting computation time. We also show that a GAC-guaranteeing decomposition of such a constraint comes at the cost of extra memory use and does not improve solving times significantly.

C6 To remedy these shortcomings, we introduce a novel, *global* constraint on probability distributions that are represented by OBDDs and have a certain *monotonic* property. We also present and implement a propagation algorithm for this *stochastic constraint on monotonic distributions (SCMD)*. This propagator leverages the structure of the OBDDs to incrementally compute the solution to the constraint, and the fact that the underlying probability distribution is monotonic to guarantee GAC.

In addressing **MRQ1** to **MRQ3**, we develop a number of different SCP solving methods, each with a number of different components, that can each be implemented in different ways. Since SCPs are hard to solve in general, and since each application domain may yield SCPs with different properties, it is not immediately obvious which solving method with which exact implementation choices for its different components works well to solve SCPs from a specific application domain. Additionally, since we use different existing tools (such as CP solvers Gecode[1] and OscaR [132], MIP solver Gurobi[2], and knowledge compilers CUDD [168] and sdd [36]), whose default parameter settings may be tuned on use cases that are rather different from the one studied in this work, it is unclear what a good parameter setting for these components might be. These uncertainties form a challenge for any scientist (or other user) who not only wants to evaluate the performance of these methods in a fair and informative manner, but also wants to use them as effectively as possible. This observation naturally begs an additional research question for this dissertation:

MRQ4 How can we fairly and informatively evaluate the running time performance of complex solving pipelines on problems from different application domains, and ultimately best employ these pipelines for solving real-world SCPs?

In addressing this question, we make the following additional contribution:

C7 We apply the paradigm of *programming by optimisation (PbO)* [80] to all solving pipelines described in this paper. For most of our design choices we implement alternatives and/or expose parameters to make the pipelines maximally configurable. We then use *automated algorithm configuration (AAC)* [79] to find optimised configurations of these solving pipelines. To the best of our knowledge, this work represents the first instance of using first use of AAC in exact probabilistic inference.

[1] Available at `www.gecode.org`.
[2] Available at `www.gurobi.com`.

The remainder of this work is organised as follows. Part I serves to provide some background for the reader. Specifically, we provide some background on how chance, logic and reasoning are related to each other in Chapter 2. We then describe several programming paradigms for optimisation in Chapter 3.

Part II is dedicated to the contributions of this work. Specifically, we start Part II with a detailed description of the SCPs that we study in this work, and briefly discuss related problems in Chapter 4. In that chapter, we also describe the problem settings used in our experiments, and the data on which those problems are formulated.

In Chapter 4, we also show how we use the *stochastic constraint on probability distributions (SCPD)* formulated in Section 1.2 to model these problems (**C1**) and present the new programming language that we propose specifically for modelling SCPs (**C2**). Then, in Chapter 5 we propose a specific kind of exact SCP solving method that takes a stochastic constraint on an SDD or OBDD encoding and *decomposes* it into a multitude of smaller constraints, resulting in a CP or MIP model that is then solved with an off-the-shelf CP or MIP solver (**C3** and **C4**). These parts of Chapter 4 and all of Chapter 5 are based on research previously published in:

Anna L.D. Latour, Behrouz Babaki, Anton Dries, Angelika Kimmig, Guy Van den Broeck, and Siegfried Nijssen. '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)*. 2017, pp. 495–511.

In the next chapter, Chapter 6, we note that the decomposition approach does not guarantee GAC and that a trivial modification of this approach does not significantly improve performance (**C5**). We therefore propose a new, global SCMD, which operates on OBDD encodings of probability distributions with a specific *monotonic* property, and demonstrate its superior performance (**C6**). Chapter 6 is based on research previously published in:

Anna Louise D. Latour, Behrouz Babaki, Siegfried Nijssen. 'Stochastic Constraint Propagation for Mining Probabilistic Networks'. In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI 2019)*. 2019, pp. 1137–1145.

We then take the decomposition method described in Chapter 5 and the global constraint propagation method described in Chapter 6, and apply the paradigm of PbO to these methods in Chapter 7. We implement alternative design choices

for the different elements of the SCP solving pipelines and use AAC to automatically configure them on sets of problems instances from several applications domains, demonstrating that the global SCMD propagation algorithm from Chapter 6 tends to outperform the other methods (**C7**). This chapter is based on research previously presented in:

- Daniël Fokkinga, Anna Louise D. Latour, Marie Anastacio, Siegfried Nijssen, and Holger Hoos. 'Programming a Stochastic Constraint Optimisation Algorithm, by Optimisation'. In: *Data Science meets Optimization workshop 2019 (DSO 2019), co-located with IJCAI 2019, Macao, 2019*.

- Anna L.D. Latour, Behrouz Babaki, Daniël Fokkinga, Marie Anastacio, Holger H. Hoos, and Siegfried Nijssen. 'Exact Stochastic Constraint Optimisation with Applications in Network Analysis'. In: *Artificial Intelligence, vol 304, 2022.*

Part I

Background

2

Logic, probability and inference

2.1 Introduction

As described in Chapter 1, the focus of this work is on developing exact solving methods for *stochastic constraint (optimisation) problems (SCPs)*, that strike a reasonable balance between convenience, generality and speed. Because probability is such an important part of SCPs, we devote this chapter to background on modelling probability distributions and on executing well-known inference tasks. In this work we take a propositional logic-based approach to modelling probability distributions and reasoning about uncertainty. This decision is motivated by the fact that logic-based models of probability distributions generalise many others. Additionally, propositional logic connects very naturally to the *constraint* solving component of the subject matter of this dissertation. Consequently, our focus in this chapter is on probability distribution representations that are based in propositional logic.

We therefore open this chapter in Section 2.2 with some background on propo-

sitional logic, typical problems that can be formulated on propositional formulae, and their associated complexity classes. Since this work aims to solve constraint (optimisation) problems that involve a *stochastic* component in particular, and since the field of probabilistic inference has produced a rich literature on how to solve such problems, in Section 2.3 we give a brief overview of the main probabilistic inference tasks known from this literature.

Moving on to concrete methods for solving probabilistic inference problems, we then discuss the technique of *knowledge compilation* in Section 2.4, and how to use the compilation of propositional formulae to *decision diagrams (DDs)* for probabilistic inference in Section 2.5.

While propositional formulae provide a general way of representing probability distributions, *constructing* these formulae is not always straightforward. We therefore also describe another way of representing probability distributions: the probabilistic logic program, in Section 2.6. This provides a convenient and human-friendly language for programming probability distributions, which can then be converted to DDs or propositional formulae. We conclude this chapter in Section 2.7.

2.2 Propositional logic

In this work, we take a probabilistic logic-based approach to reasoning about uncertainty. In particular, we focus on (literal-weighted) propositional formulae to represent probability distributions. In this section we first give a brief recap of propositional formulae. We then discuss how they can represent probability distributions and how we can compute probabilities from propositional formulae. Finally, we reflect on some relevant complexity classes, using problems formulated on propositional formulae as representative members of those classes. This discussion serves to put the next section (about different kinds of probabilistic queries) into a computational complexity context.

2.2.1 Propositional formulae

Recall the stochastic constraint Equation 1.1 that we defined in Section 1.2. In Section 2.2.3, we show how we can use the formalism of *weighted model counting (WMC)* to compute the probability that an event ϕ occurs. In order to do so, we model ϕ as a *propositional formula*. We now give a brief overview of the notation and terminology that we use in this work. For a detailed description of propositional logic, we refer the reader to the literature, *e.g.*, *Mathematical Logic*

for Computer Science [14].

A propositional formula $\phi(\mathbf{X})$ is defined on Boolean variables $X \in \mathbf{X}$ that are connected in the formula through binary logical connectives $\vee$ ("or") and $\wedge$ ("and"). Variables in the formula may be negated ($\neg$). We define a *literal* as either a variable X or its negation, $\neg X$. In this work, we use *true*, $\top$ and 1 to indicate 'true', and *false*, $\bot$ and 0 to indicate 'false'.

We call $sc(\phi) = \mathbf{X}$ the *scope* of ϕ. In this work, we assume the variables in $sc(\phi)$ to be partitioned into a set of Boolean decision variables $\mathbf{D}$ and a set of Boolean stochastic variables $\mathbf{T}$. We use $D \in \mathbf{D}$ to indicate a Boolean decision variable, and $T \in \mathbf{T}$ to indicate a Boolean stochastic variable. When the type of variable is irrelevant, we use $X \in \mathbf{X}$ to indicate a generic Boolean variable and a generic set of Boolean variables.

When literals are connected through only $\vee$s, we call this a *disjunction* or a (disjunctive) *clause*. When they are connected through only $\wedge$s, we call this a *conjunction*. If a propositional formula consists of a conjunction of clauses, this formula is in *conjunctive normal form (CNF)*. If it consists of a disjunction of conjunctions of literals, the formula is in *disjunctive normal form (DNF)*. For the scope of this thesis we do not assume the input propositional formulae to be in any particular normal form, unless otherwise specified.

We sometimes assign *truth values* to (some of) the variables in a propositional formula ϕ and then consider the *residual formula* that we obtain by replacing those variables by their truth values in ϕ and simplifying the result. We denote such a *partial assignment* by $\pi : \mathbf{X} \mapsto \{\top, \bot\}$ and denote the residual formula as $\phi|_\pi$. In this work in particular, we often want to evaluate the residual propositional formula after we have assigned truth values to the *decision variables* in particular. Since such a partial assignment corresponds to us making a set of decisions, we refer to a partial assignment that only assigns truth values to the decision variables, $\sigma : \mathbf{D} \mapsto \{\top, \bot\}$, as a *strategy*. If σ assigns truth values to only a subset of the decision variables in $sc(\phi)$, we call it a *partial strategy*. When a partial assignment only assigns truth values to stochastic variables, we call $v : \mathbf{T} \mapsto \{\top, \bot\}$ a *scenario* or *possible world*. Any π that assigns a truth value to all variables in $sc(\phi)$ is called an *interpretation* of ϕ. We will sometimes abuse notation and treat a partial assignment or interpretation simply as the set of literals that it sets to $\top$.

We say that a formula evaluates to *true* under σ, in which case we call ϕ *satisfiable*, if $\phi|_\sigma = \top$. We call any interpretation of ϕ that makes ϕ evaluate to *true* a *model*, *witness* or *solution* of ϕ.

The task of deciding whether a propositional formula in CNF has at least one model, is known as the *Boolean satisfiability problem (SAT)* and is known to be

$\mathcal{NP}$-complete [38, 106] (see Section 2.2.4). The task of *counting* the total number of models of a propositional formula, is known as the *propositional model counting problem (#SAT)*, and is the canonical #$\mathcal{P}$-complete problem (see Section 2.2.4). In Section 2.2.3, we discuss how a weighted version of this counting problem relates to computing probabilities, but we first briefly recall some basic concepts of *first-order logic*.

2.2.2 First-order logic

While in this work we limit ourselves to propositional formulae as a basis for reasoning about uncertainty, for the ease of discussion in later sections, we now briefly recall basic concepts and notation of *first-order* formulae. For a formal and more detailed description, we refer the reader to the literature, *e.g.*, *Mathematical Logic for Computer Science* [14].

Like propositional formulae, complex first-order formulae are constructed from simpler formulae using logical connectives such as $\vee$ and $\wedge$, a negation operator $\neg$ and the constants $\top$ and $\bot$. The most simple form of logical formula is the literal, which in the context of first-order logic is more generally defined than in propositional logic. In first-order logic, a literal is an *atomic formula* (*atom*) or its negation. An atom is a *predicate* that operates on *terms*, of which there are different types.

Informally, we can think of terms as objects or entities. The simplest ones are constants, *e.g.*, a and b. They can represent, for example, people. Terms can also be variables. We can interpret variables and constants by giving them values from a domain. The third type of terms are functions. Function symbols are applied on terms to define new terms.

As stated above, the literals of first-order logic are predicates and their negations. These predicates express properties of objects (terms) or relations between objects. A predicate can take any number of arguments, including zero, which is called a nullary predicate. Regardless of the input, a predicate always returns a truth value ($\top$ or $\bot$). Consider for example the domain of friends $\{a, b\}$ and the binary predicate $isFriendOf(X, Y)$. If we interpret X and Y by assigning them the values a and b, respectively, the predicate $isFriendOf(a, b)$ evaluating to *true* may mean that a is a friend of b. Predicates are the reason that first-order logic is sometimes referred to as *predicate logic* [84].

Note that, while expressions involving only constants, variables, and functions are terms, no expression involving a predicate is a term. Rather, an expression involving a predicate is a formula.

In addition, first-order logic uses *quantifiers* to indicate for how many values

the relationship expressed by predicates must hold. The *existential quantifier* $\exists$ ("exists") and the *universal quantifier* $\forall$ ("for all") can be used to express that a certain relationship holds for at least one arbitrary value, or for all values, respectively. For example: $\psi = \forall x \exists y.pred(x, y)$ means that ψ evaluates to *true* iff, for any value x, we can find at least one value y such that the predicate $pred(x, y)$ evaluates to *true*.

Note that first-order logic can express subtle details about the relationships between objects and can be a more compact way to encode information than propositional logic, especially if there is a lot of structure or repetition in the problem or information that we want to model [84]. If the same predicate holds for many (combinations of) values, this can often be more compactly expressed in first-order logic than in propositional logic.

2.2.3 Propositional weighted model counting

In this work, we will use propositional formulae to represent probabilistic models. We then use the formalism of *weighted model counting (WMC)* to compute probabilities from these formulae.

There are many alternative formalisms for representing probability distributions, which we will reflect on in the next chapter. In this work, however, we use a propositional WMC approach, since it generalises many other well-known approaches and is common practice in the domains of probabilistic reasoning, planning and learning [13, 32, 33, 45, 52, 57, 61, 64, 158].

Before we give a formal definition of WMC, we first need to define the notion of *literal-weighted propositional formulae*:

Definition 2.2.1 (Literal-weighted propositional formula)**.** *A* literal-weighted propositional formula *is a tuple* $\langle \phi(\mathbf{X}), W(\mathbf{L}) \rangle$*, of a propositional formula* ϕ *and a* weight function $W : \mathbf{L} \mapsto \mathbb{Q}_0^+$ *that maps each literal in* $\mathbf{L}$ *to a non-negative rational number. Here,* $\mathbf{X}$ *the set of variables of* ϕ*, i.e.,* ϕ*'s scope, and* $\mathbf{L}$ *the corresponding set of literals.*

Specifically, we must define a particular type of literal-weighted formula: one in which all weights can be interpreted as probabilities:

Definition 2.2.2 (Probability-weighted propositional formula)**.** *A* probability-weighted propositional formula *is a tuple* $\langle \phi(\mathbf{T}), W(\mathbf{L}) \rangle$*, with* $\mathbf{T}$ *the set of variables of* ϕ*, i.e.,* ϕ*'s scope, and* $\mathbf{L}$ *the corresponding set of literals. The weight function* $W : \mathbf{L} \mapsto \{w \in \mathbb{Q} \mid (0 \leq w \leq 1) \wedge (\forall L \in \mathbf{L} : W(\neg L) = 1 - W(L))\}$ *assigns a rational weight between* 0 *and* 1 *to each literal* $L \in \mathbf{L}$*, such that the weight of* L *(denoted by* $W(L)$*) and the weight of its negation sum up to* 1*.*

Here, we follow Sato's semantics for probability distributions [160]. In practice, this means that we interpret the probability $W(L)$ to be the probability that L has the value $\top$. We also assume that the probabilities associated with variables are mutually independent. We will use literal-weighted propositional formulae as a way to represent probabilistic models (see Section 2.3.1) and for probabilistic reasoning about those models (see Section 2.2.3). Note that, because of the assumption that $W(\neg L) = 1 - W(L)$, probability-weighted propositional formulae describe Bernoulli distributions.

Specifically, we use WMC to compute the probability $P(\phi|_\sigma)$ of an event $\phi(\mathbf{D}, \mathbf{T})$, defined on decision variables $\mathbf{D}$ and stochastic variables $\mathbf{T}$, occurring for a given strategy σ that maps each decision variable in $\mathbf{D}$ to a truth value. Using the above definition of a probability-weighted propositional formula, we define the WMC as follows:

Definition 2.2.3 (Weighted model count)**.** *The weighted model count of a probability-weighted propositional formula $\langle \phi(\mathbf{T}), W(\mathbf{L}) \rangle$ is given by:*

$$wmc(\phi) = \sum_{v \in \mathcal{M}} \prod_{L \in v} W(L), \tag{2.1}$$

where v denotes a model of $\phi(\mathbf{T})$, represented as the set of literals that are set to $\top$ in that model and $\mathcal{M}$ denotes the set of models of $\phi(\mathbf{T})$, and $\mathbf{T}$ and $\mathbf{L}$ as defined in Definition 2.2.2.

We interpret $wmc(\phi)$ as the probability that $\phi(\mathbf{T})$ evaluates to $\top$, given the probabilities as defined by $W(\mathbf{L})$. We therefore also write $P(\phi)$ to denote ϕ's weighted model count. Note that Equation 2.1 shows that we assume that the probabilities associated with positive literals by W are mutually independent. If we had not made this assumption, we would not be able to simply multiply literal weights in Equation 2.1.

The WMC formalism can be used for counting the number of solutions to a propositional formula, and is known to be $\#\mathcal{P}$-complete [155].

We can now use the notion of the weighted model count of a probability-weighted propositional formula to compute the *success probability* $P(\phi|_\sigma)$ of the residual formula that is obtained when conditioning a propositional formula $\phi(\mathbf{D}, \mathbf{T})$ on a full strategy σ, which assigns truth values to all variables in $\mathbf{D}$. We illustrate this with the following example:

Example 2.2.1 (The weighted model count of a propositional formula)**.** *Consider*

the following propositional formula, strategy and weight function:

$$\begin{aligned}\phi_d(\mathbf{D},\mathbf{T}) &:= D_d \vee (D_c \wedge T_{cd}) \vee (D_b \wedge T_{bc} \wedge T_{cd}) \vee (D_a \wedge T_{ac} \wedge T_{cd}) \vee \\ &\quad (D_b \wedge T_{ab} \wedge T_{ac} \wedge T_{cd}) \vee (D_a \wedge T_{ab} \wedge T_{bc} \wedge T_{cd}), \\ \sigma_d &:= \{D_a := \top, D_b := \top, D_c := \bot, D_d := \bot\}, \\ W(\mathbf{L}) &:= \{p_{T_{ab}} := 0.4, p_{T_{ac}} := 0.8, p_{T_{bc}} := 0.1, p_{T_{cd}} := 0.3\},\end{aligned} \tag{2.2}$$

with $\mathbf{D} = \{D_a, D_b, D_c, D_d\}$ *and* $\mathbf{T} = \{T_{ab}, T_{ac}, T_{bc}, T_{cd}\}$*. Here, we have only listed the weights of the positive literals for brevity, and denote the weight of a literal* L *with* p_L*, since it reflects a probability. Conditioning* ϕ_d *on* σ_d *yields the following residual formula:*

$$\phi_d|_{\sigma_d} := (T_{bc} \wedge T_{cd}) \vee (T_{ac} \wedge T_{cd}) \vee (T_{ab} \wedge T_{ac} \wedge T_{cd}) \vee (T_{ab} \wedge T_{bc} \wedge T_{cd}). \tag{2.3}$$

We can now 'roll the dice' for each stochastic variable to obtain a truth value for it. The resulting possible world $v : \mathbf{T} \mapsto \{\bot, \top\}$ *represents just one scenario of what could happen once the probabilistic truth values of the stochastic variables are revealed. Substituting the truth values specified by* v *into the residual formula* $\phi_d|_{\sigma_d}$ *either satisfies that formula, in which case* $v \in \mathcal{M}$ *is a model of* $\phi_d|_{\sigma_d}$*, or falsifies it.*

We can now use Equation 2.1 *to compute the* success probability *of* $\phi_d|_{\sigma_d}$*, which we illustrate in Table 2.1, finding a success probability of* $P\left(\phi_d|_{\sigma_d}\right)$ 0.2460 *for this formula.*

Table 2.1: *An example of how to compute the weighted model count of Equation* 2.3*. Note that the table only lists interpretations that are models of* $\phi_d|_{\sigma_d}$*.*

model	**weight**
$\{T_{ab} := \top, T_{ac} := \top, T_{bc} := \top, T_{cd} := \top\}$	$0.4 \cdot 0.8 \cdot 0.1 \cdot 0.3 = 0.0096$
$\{T_{ab} := \top, T_{ac} := \top, T_{bc} := \bot, T_{cd} := \top\}$	$0.4 \cdot 0.8 \cdot 0.9 \cdot 0.3 = 0.0864$
$\vdots$	$\vdots$
$\{T_{ab} := \bot, T_{ac} := \top, T_{bc} := \top, T_{cd} := \top\}$	$0.6 \cdot 0.8 \cdot 0.1 \cdot 0.7 = 0.0336$
	$P(\phi_d\|_{\sigma_d}) = 0.2460$

2.2.4 Relevant complexity classes

The problems studied in this work are $\mathcal{NP}$-hard in the general case. They involve solving sub-problems (probabilistic inference problems, specifically) that are shown to be $\#\mathcal{P}$-complete. Later in this chapter, we will describe different types of probabilistic inference and how they relate to the SCPs studied in this

work. In order to put those into (their computational complexity) context, we first provide some definitions.

Recall from Section 2.2.1 that the SAT problem asks, for a given propositional input formula ϕ in CNF, if there is a model of that formula. If there is, we call such a formula *satisfiable*. Informally, problems that are members of the $\mathcal{NP}$ complexity class are decision problems for which we can efficiently verify if a candidate solution to an instance of this decision problem is indeed a solution to this problem instance. Here, the word "efficiently" means "in time polynomial in the input size on a deterministic Turing machine or equivalent model of computation".

In order to define the class $\mathcal{NP}$ more formally, we remark that we can encode a problem instance (a propositional formula, in case of SAT) as a string. We can define the problem size as the length of that string. Using these strings, we can define *languages* that describe a problem class. In the SAT example, the SAT language would consist of all strings that describe propositional formulae that are satisfiable.

We can now imagine the existence of an algorithm that checks (verifies) if an interpretation π of ϕ is indeed a model of ϕ. It thus takes two inputs: a string encoding the propositional formula and a string encoding the interpretation. In the computational complexity literature, this algorithm is typically modelled by a Turing Machine (TM). Using these notions, we can more formally define the class $\mathcal{NP}$ as follows (inspired by the definitions given by Arora & Barak [6] and Martin [124]):

Definition 2.2.4 (The class $\mathcal{NP}$)**.** *Given a finite set of characters (an* alphabet*) A and a language $L \subseteq A^*$, which is a set of strings of arbitrary length, comprising characters from A. This language L is in $\mathcal{NP}$ iff there exists a polynomial $p : \mathbb{N} \mapsto \mathbb{N}$ and a polynomial-time TM M, which we call the* verifier *for L, such that for every string $x \in A^*$,*

$$x \in L \Leftrightarrow \exists u \in A^{p(|x|)} \text{ s.t. } M(x, u) = \top$$

Here, we call u a certificate *for input string x (with respect to L and M) if $x \in L$ and $u \in A^{p(|x|)}$, with $|x|$ the length of the input string x.*

Without loss of generality, we can choose this alphabet to simply be $\{\bot, \top\}$, which we use to both encode the input string x describing the problem instance, and the certificate u. Note that encoding x using an alphabet of $A = \{\bot, \top\}$ is note quite as trivial as encoding u. In the computational complexity literature, it is often assumed that the alphabet also has a *blank* symbol $\square$ and a *start* symbol $\triangleright$ for easier encoding in a Turing Machine. Because a discussion of how to encode x and a discussion of the exact workings of the Turing Machine model of computation are outside the scope of this work, we will ignore these special symbols.

As long as the alphabet is finite, the exact choice of the alphabet is unimportant, since we can always come up with a transformation of one alphabet to another. To get an intuition for this, consider two finite alphabets, $A = \{\bot, \top\}$ and B, with $|A| < |B|$. Now, B can easily simulate A by simply mapping $\top$ to a symbol $b \in B$ and $\bot$ to a symbol $b' \in B$ and not using all other symbols in B. Conversely, A can encode any symbol in B using $\log |B|$ bits.

For the sake of completeness, we also recap the concepts of *hardness* and *completeness*. Informally, a problem P' is called *hard* for a certain time complexity class if any problem P in that complexity class can be *reduced* to P' in polynomial time. Intuitively, this polynomial reduction means that we can transform input string $x \in L$ (the encoding of problem P) into a an input string $x' \in L'$ in time that is polynomial in $|x|$, such that a TM that verifies L' can be used to verify L.

More formally, we repeat the following definitions from Arora & Barak [6]:

Definition 2.2.5 (Reduction)**.** *A language $L \in A^*$ is* polynomial-time Karp reducible to a language L'*, denoted by $L \leq_p L'$, if there is a polynomial-time computable function $f : A^* \mapsto A^*$ such that for every $x \in A^*$, $x \in L$ iff $f(x) \in L'$.*

Note that L and L' need not be languages on the same alphabet A. Rather, f may also be a function from strings on finite alphabet A to strings on a different finite alphabet B, because of the aforementioned possibility of using one alphabet to simulate another.

Using the definition above, we can give definitions for $\mathcal{NP}$-hardness and $\mathcal{NP}$-completeness:

Definition 2.2.6 ($\mathcal{NP}$-hardness and $\mathcal{NP}$-completeness)**.** *We say that L' is* $\mathcal{NP}$-hard *if $L \leq_p L'$ for every $L \in \mathcal{NP}$. We say that L' is* $\mathcal{NP}$-complete *if L' is $\mathcal{NP}$-hard and $L' \in \mathcal{NP}$.*

Here the "NP" in $\mathcal{NP}$ stand for "non-deterministic polynomial". Intuitively, $\mathcal{NP}$-complete problems are those problems for which there cannot be an algorithm that can efficiently (*i.e.*, in polynomial time) and deterministically find a solution to an arbitrary instance of that problem that it is presented with, but for which it is possible to construct an algorithm that efficiently *checks* if a given solution (*e.g.*, a non-deterministic *guess*) to the problem instance is indeed a solution.

At least, that is the hunch that the overwhelming majority of computer scientists share on this topic. Unless we find that $\mathcal{P}$=$\mathcal{NP}$, in which case there would be a way to construct polynomial-time ("efficient") algorithms for solving $\mathcal{NP}$-complete problems, or definitively prove that $\mathcal{P}\neq\mathcal{NP}$, we operate under the *assumption* that no theoretically efficient (polynomial) algorithms exist to find a solution to an arbitrary instance of any $\mathcal{NP}$-complete problem.

Recall that we study stochastic constraint (optimisation) problems in this work, and aim to solve them *exactly*. In the case of the stochastic constraint *satisfaction* problems, it is clear that they are *decision problems*: we ask if the constraint is satisfied. We can turn stochastic constraint *optimisation* problems into stochastic constraint *satisfaction* problems by simply asking if the value that we aim to maximise exceeds a certain threshold (and analogously for minimisation problems). We discuss the relationship between stochastic constraint satisfaction and stochastic constraint optimisation in more detail in Section 4.1.

While the SCPs that we study in this work are shown to be $\mathcal{NP}$-hard in the general case, they need not be in $\mathcal{NP}$, and thus need not be $\mathcal{NP}$-complete. This is because real-world examples of SCPs (such as the ones studied in this work) often-times contain a lot of structures and symmetries that can be analysed to define *special cases* of SCPs for which we *can* design theoretically efficient solving algorithms.

As briefly touched upon in Section 1.3, in order to exactly solver SCPs, we need to evaluate the quality of different strategies, and evaluating the quality of a strategy is $\#\mathcal{P}$-complete. The complexity class $\#\mathcal{P}$ (pronounced "sharp p") captures the class of problems in which we are not just interested in *finding* a solution, but rather in *counting* all solutions to a problem instance. Perhaps unsurprisingly, the canonical problem of this complexity class is the *problem of counting the number of models of a propositional formula (#SAT)*, the *model counting problem* for short, in which we count all models of a given propositional formula, as discussed in Section 2.2.1.

More formally, we again give a definition analogous to the one given by Arora & Barak [6]:

Definition 2.2.7 (The class $\#\mathcal{P}$)**.** *A function $f : A^* \mapsto \mathbb{N}$ is in $\#\mathcal{P}$ if there exists a polynomial $p : \mathbb{N} \mapsto \mathbb{N}$ and a polynomial-time TM M such that for every $x \in A^*$:*

$$f(x) = \left| \left\{ u \in A^{p(|x|)} : M(x, u) = \top \right\} \right|.$$

Note that this definition implies that the count itself can be encoded in a string whose length is polynomial in the length of the input string x that encodes the problem instance. The weighted version of this problem, as described in Section 2.2.3, is also shown to be $\#\mathcal{P}$-complete in the general case [155].

Perhaps unsurprisingly, counting can be harder in practice than deciding. In particular, some problems that are easy to decide have an associated counting version that is hard. A well-known example are propositional formulae in DNF. Their satisfiability can be decided easily (while deciding their falsifiability

is hard), but counting the number of models of a DNF formula is not easier than counting the number of models of a CNF formula.

In order to define completeness for $\#\mathcal{P}$, we must first define $\mathcal{FP}$, the class of functions $f : A^* \mapsto \mathbb{N}$ that are computable by a deterministic polynomial-time Turing Machine [6]. Intuitively, computable functions are those that can be computed by an algorithm or computer without going into infinite loops. A formal definition of $\#\mathcal{P}$-completeness is outside the scope of this work, but informally, a function f is $\#\mathcal{P}$-complete if it is in $\#\mathcal{P}$ and the existence of a polynomial-time algorithm for f implies that $\#\mathcal{P} = \mathcal{FP}$. We refer the reader to the literature for more details, *e.g.*, *Computational Complexity*, by Arora & Barak [6].

Finally, we point to the class of problems that represents *decision versions* of problems in $\#\mathcal{P}$: $\mathcal{PP}$. Intuitively, and taking the #SAT problem as an example, the question asked in these problems is whether the model count of a propositional formula meets a certain threshold. More formally [6]:

Definition 2.2.8 (The class $\mathcal{PP}$)**.** *A language L is in $\mathcal{PP}$ if there exists a polynomial-time TM M and a polynomial $p : \mathbb{N} \mapsto \mathbb{N}$ such that for every $x \in A^*$,*

$$x \in L \Leftrightarrow \left|\left\{u \in A^{p(|x|)} : M(x,u) = \top\right\}\right| \geq \frac{1}{2} \cdot 2^{p(|x|)},$$

where we assume that $A = \{\bot, \top\}$.

Here, $2^{p(|x|)}$ represents the total number of possible certificates for the input instance x, and it is assumed that the length of a certificate u is polynomial in the length of x.

In the language of Turing Machines, we can think of L as a language whose strings are accepted by the *majority* of paths in non-deterministic TM M. Unlike problems that are in $\mathcal{NP}$, and taking the SAT problem as an example, we do not need to find just one model, but we must check that the majority of interpretations are models of the input formula.

We finally point to the concept of *oracles*, which are used in computational complexity theory to reason about different complexity classes. Informally, an oracle is a black box that answers queries. Even though those queries represent problems in a specific complexity class, it costs a Turing Machine only one time step to query the black box. For example, $\mathcal{NP}^{PP}$ is the class of problems that can be decided by a non-deterministic, polynomial-time Turing Machine, provided that it has access to an oracle that decides problems that are in $\mathcal{PP}$.

In the next section, we describe a number of probabilistic inference tasks that are members of the complexity classes described above.

2.3 Probabilistic inference

As we described in Chapter 1, we study problems that involve some sort of stochastic component, and thus require us to perform some kind of probabilistic reasoning. Given a probabilistic model, we use the term *probabilistic inference* to refer to answering *probabilistic queries* about the model, such as "what is the probability that it will be windy and rainy when I go outside?" In this section we describe several probabilistic queries known from the probabilistic inference literature.

Specifically, we identify three main types of inference tasks: *max-inference* tasks, *sum-inference* tasks and *mixed-inference* tasks [143]. They are listed here in increasing order of difficulty, their complexity classes ranging from $\mathcal{NP}$-complete, to $\#\mathcal{P}$-complete to $\mathcal{NP}^{PP}$-complete, respectively [47, 139], with $\mathcal{NP} \subseteq \#\mathcal{P} \subseteq \mathcal{NP}^{PP}$.

Before we describe these tasks, their complexities, and techniques that have been developed to solve them, we first introduce some notation and terminology related to the probabilistic models on which these tasks are formulated.

2.3.1 Probabilistic models

Probabilistic models can be represented in different ways. In the previous section, we discussed a specific way of modelling probability distributions: literal-weighted propositional formulae. A popular alternative approach to representing probabilistic models are graphical models [97, 140].

A well-known example of a graphical model is the *Bayesian network (BN)* [140], whose name was coined by Pearl in the 1980s. BNs are directed-acyclic graphs in which each node represents a variable and the directed edges indicate dependence relationships. With each node they also associate a *conditional probability table (CPT)* that describes that relationship. Another well-known example is the *Markov Network (MN)*, or *Markov Random Field (MRF)* [140], which is an undirected version of the Bayesian network.

However, in this work we choose *literal-weighted propositional formulae* to model probability distributions (as described in Section 2.2), since they generalise other approaches [32, 158]. Because of that choice, in this work, we consider problems formulated on Boolean variables. For ease of discussion, here we therefore assume that all variables have Boolean domains, but this need not be the case in general (in fact, they need not even be discrete).

The contents of this section are agnostic of any specific probabilistic model representation, unless indicated otherwise.

In our discussion of probabilistic inference tasks below, we consider a probabilistic model $\mathcal{P} = \langle \mathbf{E}, \mathbf{S}, \mathbf{Q}, \Phi \rangle$ on variables that are partitioned into three disjoint sets: $\mathbf{E}$, $\mathbf{S}$ and $\mathbf{Q}$. Intuitively, we can think of these sets as variables whose values represent *evidence* ($\mathbf{E}$), variables that must be marginalised out ($\mathbf{S}$), which is done by *summation*, and variables whose values we want to *query* ($\mathbf{Q}$). Taking the spread of influence problem as described in Section 1.1 as an example, we can think of the *evidence* as a decision on which people get a free sample. Maybe we want to target a specific group (*e.g.*, women in tech, people who like running, reading enthusiasts, ...). We then *query* members of that group to predict the expected reach of our marketing campaign to that specific target audience, leaving members outside that target audience (who might still participate in spreading the word about our product) to be *marginalised out*.

Joint probability distributions on these variables are, in the graphical model literature, typically defined using a set $\mathbf{F}$ of *potentials* f. These potentials map sets of (truth value) assignments to variables in $\mathbf{X}$ to real numbers: $f : \{\top, \bot\}^{|\mathbf{X}|} \mapsto \mathbb{R}$. In general, each potential must take a nonnegative value for at least one set of truth value assignments [187]. In the context of graphical models, we can typically interpret these potentials as *conditional probability tables (CPTs)* that associate a probability with an assignment of truth values to the variables in the scope of the potential ($\mathbf{X}$). We thus consider all the potential's values to be nonnegative in this chapter.

Recall the discussion of literal-weighted propositional formulae and their weighted model counts from Section 2.2.3. Similar to the CPTs in the context of graphical models, we can see Table 2.1 as a potential. Instead of having multiple potentials that define a probability distribution over subsets of all variables involved, now we have just one potential. This potential maps models of the literal-weighted propositional formula to their weights, and any interpretation of the formula that is not a model to 0.

We now continue with a description of the three main probabilistic tasks that can be formulated on probabilistic models.

2.3.2 Max-inference tasks

Max-inference tasks typically aim to find the most probable configuration of a joint probability distribution. A typical max-inference task is the *most probable explanation (MPE)* task. In some of the literature, this task is also known as the *maximum probability assignment (MPA)* task [20], or the *maximum a posteriori (MAP)* task, *e.g.*, in [1, 35, 91, 104, 112, 126, 153, 167, 184]. Given a probabilistic model $\mathcal{P}$ as described above, with $\mathbf{S} = \varnothing$, and some *evidence* $\mathbf{e}$ in the form of truth

assignments to the variables in $\mathbf{E}$, $\mathbf{e} : \mathbf{E} \mapsto \{\bot, \top\}$, the MAP task aims to find an assignment of truth values $\mathbf{q}$ to the variables in $\mathbf{Q}$, $\mathbf{q} : \mathbf{Q} \mapsto \{\bot, \top\}$, that maximises $P(\mathbf{Q} = \mathbf{q} \mid \mathbf{E} = \mathbf{e})$.

Then, in the context of graphical models, for each possible $\mathbf{q}$ we can write:

$$P(\mathbf{Q} = \mathbf{q} \mid \mathbf{E} = \mathbf{e}) := \frac{\prod_{f \in \mathbf{F}} f(\mathbf{e}, \mathbf{q})}{P(\mathbf{E} = \mathbf{e})} \tag{2.4}$$

Note that, in the context of literal-weighted propositional formula representations of probabilistic models, instead of Equation 2.4, for each possible $\mathbf{q}$ we would write:

$$P(\mathbf{Q} = \mathbf{q} \mid \mathbf{E} = \mathbf{e}) := \begin{cases} \prod_{L \in \mathbf{q}} W(L) & \text{if } \mathbf{e} \cup \mathbf{q} \in \mathcal{M}(\phi) \\ 0 & \text{otherwise,} \end{cases} \tag{2.5}$$

where $W(L)$ represents the weight of literal L and $\mathcal{M}(\phi)$ is the set of models of propositional formula ϕ. Note that this is simply the weight of the interpretation defined by $\mathbf{e} \cup \mathbf{q}$.

The goal of the MPE task is then to find an assignment of truth values to variables in $\mathbf{Q}$ that maximises this probability:

$$\mathbf{q}^* := \arg\max_{\mathbf{q}} P(\mathbf{Q} = \mathbf{q} \mid \mathbf{E} = \mathbf{e}), \tag{2.6}$$

where in fact we can also simply compute $\mathbf{q}^* \in \arg\max_{\mathbf{q}} \prod_{f \in \mathbf{F}} f(\mathbf{e}, \mathbf{q})$ (in the graphical model view), since $P(\mathbf{E} = \mathbf{e})$ is independent of $\mathbf{q}$. Where, for simplicity, we assume that the arg max function in the above formula returns just one assignment, even if more than one maximise $P(\mathbf{Q} = \mathbf{q} \mid \mathbf{E} = \mathbf{e})$.

Note that, because $\mathbf{S} = \varnothing$, the MPE task can be seen as finding the most likely configuration of a set of variables $\mathbf{Q}$, given evidence about the variables in the complement of $\mathbf{Q}$, namely those in $\mathbf{E}$. As such, it is a very useful task for diagnostic purposes. For example: given a set of symptoms ($\mathbf{E} = \mathbf{e}$), a physician may want to ask what the probability is that the patient has a certain disease ($\mathbf{Q} = \mathbf{q}$).

The MPE inference task of determining if there exists a configuration $\mathbf{q}$ such that $P(\mathbf{Q} = \mathbf{q} \mid \mathbf{E} = \mathbf{e}) > \theta$ (with rational threshold $0 \leq \theta \leq 1$) is shown to be $\mathcal{NP}$-complete [20, 167]. In fact, even finding a solution to the MPE problem whose quality is guaranteed to be within a constant ratio ρ_{MAP} of the optimal solution, is shown to be $\mathcal{NP}$-hard [1].

The earliest exact methods for solving MPE were *join-tree* algorithms, devised in the context of Bayesian networks, where potentials are computed one by one, in a strict order that is determined by the (structure of the) problem, often using the potentials that were computed earlier [90, 101, 140]. In the late 1990s, Dechter

proposed the *bucket elimination* framework as a generalisation of *variable elimination algorithms*, typically used for *mixed-inference* tasks, which we will briefly discuss in Section 2.3.4. Because the max-inference and mixed-inference tasks are so closely related, Dechter proposed a general framework for solving these task. It provides functionality for balancing the space and time requirements of variable elimination algorithms [53]. Note that the efficiency of variable elimination algorithms is determined by the variable order inherent to the specific problem instance.

Alternative approaches to exact MPE solving include one based on a modification of the DPLL algorithm, using a dynamic programming approach [159], and one that is based on *integer linear programming (ILP)* [153]. Another class of algorithms encodes the problem in an AND/OR diagram (which exploits the independencies in the graphical model), and uses a depth-first branch-and-bound search to traverse that diagram in order to solve the MPE task [115, 116].

Finally, we point to the existence of approximation methods for MAP, based on, *e.g.*, local search [139], mini-bucket elimination algorithm [54], (hybrid) message passing algorithms [91, 112], weighted search [67] and others [103, 119, 138, 143, 186]. A detailed discussion of these techniques is outside the scope of this work, since we focus on exact SCP solving.

Note that solving SCPs as described in Section 1.1 never requires max-inference *only*. Instead, we always require a form of inference that performs some kind of aggregation over probabilistic paths in networks, and thus *sum-inference*.

2.3.3 Sum-inference tasks

The *probabilistic logic programming (PLP)* literature identifies an inference task in which the set $\mathbf{Q}$ contains only one variable ($\mathbf{Q} = \{Q\}$), there are no observed variables, and thus there is no evidence ($\mathbf{E} = \varnothing$), but there are latent (*i.e.*, unobserved or uninteresting) variables that must be marginalised out ($\mathbf{S} \neq \varnothing$). In this case, we call $P(Q = \top)$ the *success probability* of *query* Q, which, in the graphical model context, is computed as:

$$P(Q = \top) := \frac{1}{Z} \dot{\sum}_{\mathbf{s}} \prod_{f \in \mathbf{F}} f\left(\mathbf{s}, Q = \top\right), \tag{2.7}$$

where Z is a constant needed for normalisation, often referred to as a *partition function*. In the literal-weighted propositional formula context, computing $P(Q = \top)$ corresponds to computing the weighted model count of a formula that can only be satisfied if $Q = \top$, using Equation 2.1.

This task is known in the PLP literature as the *marginal distribution (MARG)* [52, 64, 65] or simply *MAR* [47] task, or the *PROB* task [96].

This task is known to be #$\mathcal{P}$-complete in the general case [155, 176, 177]. The decision version of this problem asks if $P(Q = \top) > \theta$ holds for a given threshold $0 \leq \theta \leq 1$. This problem is $\mathcal{PP}$-complete [6, 47].

Since the task of computing the success probability of a query is such an important task in the PLP community, many methods for solving this task either exactly or approximately have been developed over the years. In particular, it has been shown that the MARG task can be reduced to the WMC problem [8, 9, 43], which we briefly referred to in Section 2.2.4. Consequently, a lot of the literature on solving this task is based on model counting techniques.

An early example of this is the suggestion by Bacchus and Dalmao to adapt the Davis-Putnam-Logemann-Loveland (DPLL) algorithm such that it can be used for model counting [9]. They observed that computing the conditional probability distributions for variables in a Bayesian Network (the *BAYES* problem), and counting the number of model of a propositional formula (#*SAT*) are instances of the same *SUMPROB* problem, as identified by Dechter [53]. They thus proposed to solve the BAYES problem with their #DPLL algorithm [8, 9], which eventually led to weighted model counter Cachet, which employed *conflict-driven clause learning (CDCL)* for efficient weighted model counting [157, 158]. Somewhat more recent model counters of this type include weighted versions of miniC2D [134] and sharpSAT [173], specifically our weighted version[1], a version called sharpSAT-TD[2], and a weighted version of GANAK [166].[3] Finally, Chakraborty *et al.* showed that, under certain assumptions, literal-weighted propositional formulae can be transformed into unweighted propositional formulae whose model counts can then be transformed back into probabilities, thus allowing any unweighted model counter to be transformed into a weighted one [29].

These model counting techniques typically require the input to be in a certain propositional language (typically CNF). However, a large part of the probabilistic inference literature assumes the probabilistic model to be encoded as a graphical model, thus requiring some kind of encoding step before solving. Several methods for encoding probabilistic graphical models into (literal-)weighted CNFs have been proposed [13, 29, 33, 43], but a detailed discussion of them is outside the scope of this work.

A weighted model counter takes a (literal-)weighted CNF as input and returns

[1] Available at `bitbucket.org/latower/weighted-sharpsat`.

[2] Available at `github.com/Laakeri/sharpsat-td`.

[3] Weighted version available at `github.com/meelgroup/ganak/tree/wmc`.

a probability. In some contexts, this might be enough. However, in some other contexts, we may want to query the same model multiple times, only with different weights, or with different decisions. For example, in the context of the power grid reliability problem, experts may have provided us with upper and lower bounds on the survival probabilities of the power lines. Using those bounds, we may want to formulate queries for best-case and worst-case scenarios. The underlying model is the same for these two scenarios, but the weights are different. Taking the spread of influence problem as another example, we may just want to determine the expected number of eventual customers given different 'seed' sets of people who receive a free sample. Again, the underlying model is the same, but some decisions (and thus the values of decision variables) are different.

In these examples, we want to compute the success probability of a literal-weighted formula multiple times, only with slightly different weights. Using the DPLL-based algorithms, we would have to run the algorithm again for each different weight function, and we cannot reuse any results.

This single-use property of model counters is a drawback if we may want to perform multiple queries on the same model. These observations are addressed by the field of *knowledge compilation* [48, 123, 165], where a propositional formula is compiled into a *decision diagram (DD)*. These data structures capture the model into a data structure that allows for repeated querying in time that is polynomial (typically linear) in the size of the DD. Since this work relies heavily on knowledge compilation, we discuss this in more detail in Section 2.4.

Due to the complexity of the MARG task, there is also a class of bounding and approximation algorithms for solving this problem. One of the earliest methods simply uses a SAT solver to generate sample solutions, whose weights can be used to estimate the total weighted model count of a literal-weighted input formula [182], taking care to take into account the weight of the samples themselves [28]. Later methods take a hashing-based approach, which adds random XOR constraints to the formula, which cuts down the solution space until it is small enough to count. The total model count is then estimated by repeating this procedure, resulting in probabilistic upper and lower bounds on the weighted model count [30]. Another method is an anytime approach from the PLP literature simply generates partial proofs by partially grounding a probabilistic logic program, generating lower and upper bounds on the success probability that get closer as the algorithm continues to run [151, 152].

Finally, we mention parallelised methods for weighted model counting, such as ones that utilise GPUs [62] or implement parallel DD compilation algorithms [40].

2.3.4 Mixed-inference tasks

Recall the probabilistic model we described above: $\mathcal{P} = \langle \mathbf{E}, \mathbf{S}, \mathbf{Q}, \mathbf{\Phi} \rangle$. For the definition of the MPE problem, we assumed that $\mathbf{S} = \varnothing$. For the definition of MARG inference, we assumed that $\mathbf{E} = \varnothing$. We now discuss a generalisation of the MPE in which $\mathbf{S} \neq \varnothing$ and $\mathbf{E} \neq \varnothing$, known as the MAP problem. As we mentioned in Section 2.3.2, some literature calls the MPE task the MAP task. In these works, the mixed-inference task is called *marginal MAP (MMAP)* (*e.g.*, in [1, 35, 91, 104, 112, 126, 153, 167, 184]) or partial MAP [100].

The MAP task is to find an assignment of truth values $\mathbf{q}$ to the variables in $\mathbf{Q}$ that maximises $P(\mathbf{Q} = \mathbf{q} \mid \mathbf{E} = \mathbf{e})$, and thus given only *partial* evidence on the variables in the complement of $\mathbf{Q}$.

In the MAP setting, we have to marginalise out the variables in $\mathbf{S}$, which present the 'hidden' variables that we neither know, nor care about, resulting in the following expression for the probability of an instantiation $\mathbf{q}$ of the variables in $\mathbf{Q}$, in the context of graphical models:

$$P\left(\mathbf{Q} = \mathbf{q} \mid \mathbf{E} = \mathbf{e}\right) := \frac{1}{P\left(\mathbf{E} = \mathbf{e}\right)} \cdot \sum_{\mathbf{s} \in \{\top, \bot\}^{|\mathbf{S}|}} \prod_{f \in \mathbf{F}} f\left(\mathbf{e}, \mathbf{q}, \mathbf{s}\right). \tag{2.8}$$

In the context of literal-weighted propositional formulae, this would correspond to computing the weighted model count of the residual formula $\phi|_{\mathbf{q},\mathbf{e}}$, obtained by substituting the variables in $\mathbf{Q} \cup \mathbf{E}$ with their truth values according to $\mathbf{q}$ and $\mathbf{e}$, and simplifying the resulting formula. Consequently, the sum in Equation 2.1 then only runs over interpretations $\mathbf{s}$ that are models of $\phi|_{\mathbf{q},\mathbf{e}}$. Similar to the MPE task, MAP aims to find the solution to Equation 2.6, but uses Equation 2.8 to compute $P\left(\mathbf{Q} = \mathbf{q} \mid \mathbf{E} = \mathbf{e}\right)$ instead of Equation 2.4:

$$\mathbf{q}^* := \arg\max_{\mathbf{q}} \sum_{\mathbf{s} \in \{\top, \bot\}^{|\mathbf{S}|}} \prod_{f \in \mathbf{F}} f(\mathbf{e}, \mathbf{q}, \mathbf{s}), \tag{2.9}$$

where we note that, again, the factor $P(\mathbf{E} = \mathbf{e})$ is unimportant for the purposes of finding $\mathbf{q}^*$.

A generalisation of the MAP task is the *maximum expected utility (MEU)* task [53], which is formulated on a probabilistic model $\mathcal{P} = \langle \mathbf{E}, \mathbf{S}, \mathbf{Q}, \mathbf{\Phi}, \mathbf{U} \rangle$, where we are given a utility function that associates a utility $u(\mathbf{e} \cup \mathbf{s} \cup \mathbf{q})$ with each possible instantiation of the variables in $\mathbf{E} \cup \mathbf{S} \cup \mathbf{Q}$. Instead of maximising the marginal probability, this problem maximises the expected utility. As such, it is a popular setting in the field of optimisation, planning and scheduling [5, 102]. Clearly, we can formulate SCPs as MEUs, provided that we can encode the constraints into the probabilistic model.

Note that solving the MAP problem involves both maximisation and summation. Since the maximisation and summation operators do not commute, mixed-inference tasks are typically harder than either sum-inference or max-inference alone. In fact, Park and Darwiche proved that MAP is $\mathcal{NP}^{PP}$-complete in the general case [139], whereas MPE is 'only' $\mathcal{NP}$-complete.

There is a rich literature on solving the MAP task, with methods for solving the problem exactly, approximately, or for computing bounds on the solution. We highlight a few common methods.

The MAP task is naturally solved by using some form of variable elimination [53]. Early exact algorithms for MAP use variable elimination in a branch-and-bound algorithm to find an optimal solution [138, 186], in some cases even in combination with knowledge compilation [83]. Just like AND/OR diagrams can be used in algorithms to solve the MPE problem, they can also be used to solve MAP, by combining them with depth-first search [117] or best-first search [118].

In order to employ branch-and-bound algorithms, we need to compute actual upper bounds on the conditional probability. Perhaps unsurprisingly, techniques that have been developed for this are quite similar to those that have been developed for solving MPE tasks, and the MPE approximation methods that we mentioned in Section 2.3.2 can be used for this purpose.

Given the applicability of AND/OR diagrams (combined with a search algorithm) to solve both MPE and MAP exactly, it is unsurprising that anytime variants of these algorithms have also been developed over the last years [104, 119, 120, 122]. Some methods not only use MPE approximation methods to obtain meaningful upper bounds on the optimal probability of a MAP, but also employ weighted search methods [67] to also obtain meaningful lower bounds [121].

Other approximation methods rely on decomposition and approximate variable elimination [35, 143], on repeatedly performing the MARG task on each variable in the MAP to compute a lower bound [4], gradient-based methods [39], to name a few. Because of its hardness and usefulness to model a wide range of problems, many MAP approximation methods have been developed over the years, and still are being developed. Since our focus is on exact solving, we do not expand further on approximate MAP, considering it to be outside the scope of this work.

In this section, we gave a very brief overview of the main inference tasks in the probabilistic inference literature. This literature displays a strong focus on graphical models of probability distributions. As we motivated earlier, in our work we focus on propositional representations of probability distributions. In the next

section, we give a brief introduction to a useful tool in the realm of propositional inference: knowledge compilation.

2.4 Knowledge compilation

Historically, *knowledge compilation* [48, 123, 165] has been a popular method for making online WMC computation more tractable in the field of probabilistic inference and planning [33, 45, 52, 64, 82]. As we mentioned in Section 2.3.3, computing the success probability of a query is a well-known task in the field of PLP. In stochastic optimisation problems, it is only natural to want to repeatedly compute a conditional success probability, conditioned on different evidence (or strategies). However, recall from Section 2.3.3 that WMC computation can be done by using a 'single-use' weighted model counter. Consequently, if we would want to recompute the weighted model count of a probabilistic model after changing the evidence, we would also have to rerun the solver, discarding any partial results that might have been reusable.

2.4.1 Decision diagrams

Knowledge compilation represents a solution to this problem. Most knowledge compilers are essentially DPLL-based model counters that record their *trace* (the search tree) while counting. Taking care to create the trace in such a way that it has specific properties, the result is saved in a language that supports tractable (meaning "in time that is polynomial in the size of the string in that language") inference operations. A formula (or *sentence*) in this language can be represented as a *decision diagram (DD)*, and is then said to be 'compiled'. This DD can be seen as a compact representation of the truth table of the input formula.

Recall Table 2.1 in Section 2.2.3. In essence, this table is a truth table (albeit one that only lists the rows that represent models of ϕ_d). It will not surprise the reader that, by adding appropriate *weights* to a DD representation of the truth table of a literal-weighted propositional formula, we can use that DD to compute the WMC of that formula. Note also that, once the diagram is compiled, these weights can be changed to, *e.g.*, represent different assumptions about the exact probabilities in the probabilistic model, or to represent different strategies that we need to evaluate in order to solve a SCP.

Examples of DDs that are used for tractable MARG inference include *binary decision diagrams (BDDs)* and *ordered binary decision diagrams (OBDDs)* [42, 45], *negation normal forms (NNFs)* [32], *deterministic decomposable negation normal forms*

(d-DNNFs) [34], *sentential decision diagrams (SDDs)* [37], *weighted positive binary decision diagrams* [41], and *algebraic decision diagrams (ADDs)* [60]. In this work, we focus on *ordered binary decision diagrams (OBDDs)* and *sentential decision diagrams (SDDs)*, specifically. For a good overview of the different properties of some of these languages and on how they relate to each other, we refer the reader to Darwiche's *A knowledge compilation map* [48].

Note that, while all these diagrams support inference operations that take time polynomial in the size of the diagram, this diagram must still be compiled. Consequently, for a single query, the total time complexity would not be reduced if using knowledge compilation instead of running a model counting algorithm for that query. However, for repeated querying under different assumptions, this computational effort is 'shared' among all those queries.

An additional potential time saver is the fact that propositional formulae can have sub formulae in common. This is particularly likely to happen when these formulae originate from the same system. In Section 4.2 we will show how queries about real-world systems can be translated into propositional formulae. Because these formulae represent questions about the same stochastic system, they are likely to overlap in part. In this case, we can choose not to construct an individual DD for each formula, but instead compile one DD with multiple roots, each root corresponding to a different formula. This way, we can potentially save compilation time and memory, by avoiding to repeatedly recompile the same sub formulae, but instead re-using those compiled sub formulae. For the sake of simplicity, the discussion below will be limited to single-rooted DDs.

While inference operations can be done in time polynomial in the size of the diagram, the size of the diagram may still be exponential in the size of the input CNF in the worst case [48]. The task of finding a minimal-sized DD is typically also hard. In fact, finding a minimal-size OBDD is known to be $\mathcal{NP}$-hard [22], and we expect the same to hold for SDDs, although we are not aware of a published proof of this. There is a rich literature on how to compile succinct diagrams, the discussion of which is outside the scope of this work.

Note also that, since the knowledge compilation process involves storing the full trace (search tree) of the DPLL-based model counter that the compiler is built on, it may require a lot of memory, which can be prohibitive.

To summarise: compiling CNFs to DDs gives us data structures that we can use for tractable inference *if* we are able to make these DDs compact enough, and *if* we have enough memory to compile the diagram. Additionally, this effort is only useful if we need to answer multiple queries. Taking this into consideration, the question arises of why we use knowledge compilation in this work. In short:

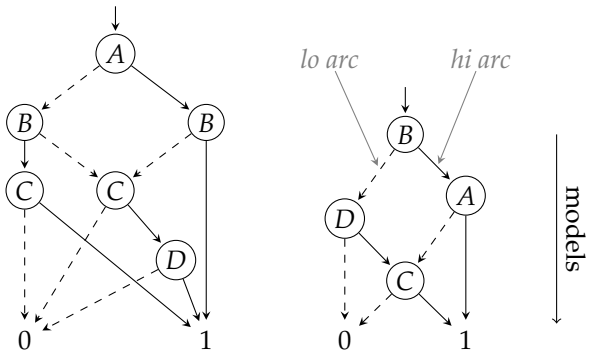


Figure 2.1: *Two small OBDDs, each encoding the propositional logic formula $\phi = (A \wedge B) \vee (B \wedge C) \vee (C \wedge D)$. The left OBDD has variable order $A \prec B \prec C \prec D$, while the right one has variable order $B \prec A \prec D \prec C$.*

DDs help us to model relationship between variables and exploit those relationships to solve SCPs. We will answer this question in detail in Chapters 5 and 6.

In the remainder of this section, we will first provide some background on two specific types of DDs: *ordered binary decision diagrams (OBDDs)* and *sentential decision diagrams (SDDs)*. In the next section, we show how to use them specifically for the task of WMC for probabilistic inference.

2.4.2 Ordered binary decision diagrams

Figure 2.1 shows two examples of OBDDs, each representing the truth table of the same propositional formula. An OBDD is a *directed acyclic graph (DAG)* with two leaf nodes that represent the values *true* (1) and *false* (0). In an OBDD $\Delta(\phi)$ that encodes a formula ϕ, each internal node n is labelled with a variable $X \in sc(\phi)$. There can be multiple nodes with the same label, but never multiple nodes with the same label *on a path* from root to leaf. Each internal node n has two outgoing arcs: a *lo arc* that corresponds to $X = \bot$ and a *hi arc* that corresponds to $X = \top$, where $X \in sc(\phi)$ labels n.

A path from the root of the diagram to the leaf node labelled with 1 corresponds to a *model* of ϕ, a mapping $sc(\phi) \mapsto \{\bot, \top\}$ of truth values on the variables in the scope of ϕ. Note that not all variables may be encountered in a path from root to leaf, since assignments of truth values to variables may make the satisfiability of the resulting residual formula agnostic to the truth values of some of the other variables. As a consequence, OBDDs can be used to very compactly encode all the models of a propositional formula.

The size and shape of an OBDD are determined by its *variable order* $\mathcal{O}$, which indicates in which order we encounter the variables in $sc(\phi)$ on a path from the root of the OBDD to a leaf. The two OBDDs in Figure 2.1 are shaped by two different variable orders.

2.4.3 Sentential decision diagrams

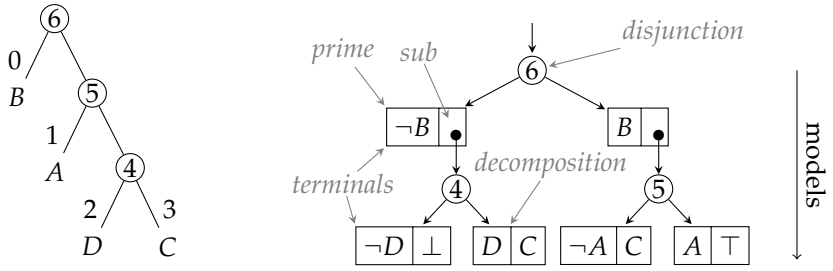
Figure 2.2 shows two examples of SDDs. Like OBDDs, SDDs are compact representations of truth tables.

In Figure 2.2 the circular nodes represent disjunctions and rectangular nodes represent decompositions of a *prime* p (rooted in the left half of the node) and a *sub* s (rooted in the right half), such that a decomposition node represents the formula $p \wedge s$. A single variable or constant in a prime or a sub is called a *terminal*. Naturally, a disjunction node is *true* if at least one of its children is *true*. A conjunction node is *true* if both the prime and the sub evaluate to *true*.

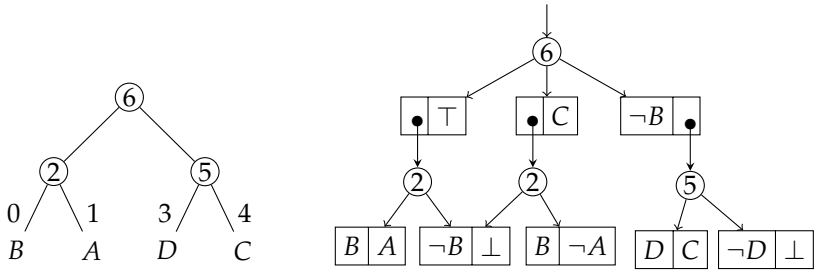
As with OBDDs, there is typically no unique SDD representation for a propositional formula. Rather, the shape and size of an SDD is determined by the *vtree* [144] that it *respects*. A vtree is a full binary tree that generalises the concept of a variable order. In particular: each disjunction node in an SDD respects a subvtree of the subvtree that the entire SDD respects, rooted at an internal node in that vtree. The left and right children of that vtree node determine the scopes of respectively the primes and subs of the children of the disjunction node in de SDD.

Consider the SDDs and corresponding vtrees in Figure 2.2. Each disjunction node in the SDD is labelled with the index of the internal vtree node that is the root of the subvtree respected by that disjunction node. The root disjunction node of the SDD respects the entire vtree. Let ϕ_p and ϕ_s be the propositional formulae represented by the prime and the sub of any decomposition node that is a descendant of a disjunction node Δ. Let ℓ and r be the left child and the right child of an internal vtree node n, respectively. We say that Δ respects n if the following holds: $sc\left(\phi_p\right) \subseteq \{X_l \in \mathcal{T}_\ell\}$ and $sc\left(\phi_s\right) \subseteq \{X_l \in \mathcal{T}_r\}$, where $\mathcal{T}_\ell$ and $\mathcal{T}_r$ represent the sub vtrees rooted at ℓ and r, respectively, and X_l represents the variable that labels a leaf in those sub vtrees.

Thus, in the (sub) SDD rooted at Δ, the sub formulae corresponding to the *primes* of the decomposition nodes that are Δ's children only contain variables that occur in the vtree rooted at the *left* child of internal vtree node n, and the sub formulae corresponding to the *subs* of the decomposition nodes that are Δ's children only contain variables that occur in the vtree rooted at the *right* child of n.



(a) *An SDD (right) that respects a right-linear vtree (left).*



(b) *An SDD (right) that respects a balanced vtree (left). Example from Darwiche [46]*

Figure 2.2: *Two examples of SDDs encoding the truth table of propositional formula $\phi = (A \wedge B) \vee (B \wedge C) \vee (C \wedge D)$, and their corresponding vtrees. Both internal nodes and leaves in the vtrees are labelled with an index to indicate their place in the variable order. Disjunction nodes in SDDs are labelled with the index of the internal vtree node they respect.*

The figure shows two examples of how vtrees influence the size and shape of SDDs. In particular, we distinguish three types of vtrees: *right-linear*, *left-linear* and *balanced*. Figure 2.2a shows an example of an SDD that respects a *right-linear* vtree, while Figure 2.2b shows an example of an SDD that respects a *balanced* vtree.

Note that we can distill a total order from vtrees by doing a left-right traversal and noting the order in which we encounter the variables. Unlike OBDDs, the size and shape of SDDs are not defined by a total order, but by a vtree. This is also illustrated in Figure 2.2. Both vtrees have the total order $B \prec A \prec D \prec C$, but they have different shapes and thus correspond to differently shaped SDDs.

In a top-down traversal of an SDD, we can interpret the primes as conditions: if the prime evaluates to *true*, then the condition in the sub must evaluate to *true* in order to make the formula evaluate to *true*. Right-linear vtrees have the special property that the SDDs that respect them are equivalent to OBDDs. Comparing the right OBDD in Figure 2.1 and the SDD in Figure 2.2a, we see that the primes

in the SDD correspond to the values of the outgoing arcs in the OBDD.

SDDs are a strict superset of OBDDs [46]. When an SDD respects a right-linear vtree, its primes can only condition on truth assignments to single variables (as is shown in Figure 2.2a). For other vtrees, however, primes might represent entire sub formulae (as is shown in Figure 2.2b), which are called *sentences* in the knowledge compilation literature (hence the name *sentential* decision diagrams). Because of this, truth tables can potentially be encoded more efficiently when the vtree is not right-linear, and thus SDDs can be made at least as small as OBDDs [24].

We remark that SDDs and OBDDs are not the only DDs that can be used for conditional probability computation in time that is linear in the size of the diagram. Other examples include *negation normal forms (NNFs)* [95], d-DNNFs [33, 145], *smooth deterministic decomposable negation normal forms (sd-DNNFs)* [95] and *affine decision trees (ADTs)* [98]. A detailed discussion of these is outside the scope of this work. We point the interested reader to Darwiche & Marquis's *A Knowledge Compilation Map* [48] on how most of these languages relate to each other.

2.5 Inference with decision diagrams

We now describe, mainly with help of examples, how we can use OBDDs and SDDs for probabilistic inference. The propositional formula that we will be using in these examples is the following:

$$\begin{aligned}\phi_d(\mathbf{D},\mathbf{T}) := {} & D_d \vee (D_c \wedge T_{cd}) \vee (D_b \wedge T_{bc} \wedge T_{cd}) \vee (D_a \wedge T_{ac} \wedge T_{cd}) \vee \\ & (D_b \wedge T_{ab} \wedge T_{ac} \wedge T_{cd}) \vee (D_a \wedge T_{ab} \wedge T_{bc} \wedge T_{cd}),\end{aligned} \tag{2.10}$$

with $\mathbf{D} = \{D_a, D_b, D_c, D_d\}$ a set of Boolean decision variables and $\mathbf{T} = \{T_{ab}, T_{ac}, T_{bc}, T_{cd}\}$ a set of Boolean stochastic variables. We will discuss the origin of this formula in more detail in Section 4.2. Recall that $\phi_d|_\sigma(\mathbf{T})$ is the residual formula obtained by taking $\phi_d(\mathbf{D},\mathbf{T})$ and replacing the variables in $\mathbf{D}$ by their values specified by strategy σ and simplifying. Since the goal of solving Equation 1.1 is finding a strategy σ that satisfies that constraint, we are going to describe how to use OBDDs and SDDs to compute $P(\phi_d|_\sigma)$, particularly.

As discussed in the previous section, DDs are data structures that summarise truth tables of propositional formulae, and we can use DDs to compute the (weighted) model count of a formula in time that is linear in the size of the DD, instead of simply listing all the models of the formula, determining their individual weights and summing those, as we did in Table 2.1. In order to employ a DD to compute the success probability of a probability-weighted propositional

formula, we must transform it into an *arithmetic circuit (AC)*. In the following, we will not explicitly show the *arithmetic circuits (ACs)*, but describe how to transform an OBDD or SDD into one.

Note that we compile the entire formula ϕ, and then compute $P(\phi|_\sigma)$ by setting the appropriate weights in the AC to reflect σ. Once those weights are defined, a bottom-up traversal of such an AC computes the success probability of $\phi|_\sigma$. For the sake of brevity, we will sometimes abuse terminology and say that we traverse the DD to compute that probability, where we actually mean that we traverse the AC that is obtained from the DD. Note that, for OBDDs and SDDs, the size of the AC is linear in the size of the DD it was constructed from.

As we discussed in Section 2.4.1, a DD may have multiple roots, each corresponding to a different formula. Consequently, an AC constructed from such a multi-rooted DD may also have multiple roots, where each returns the success probability of a different query. For the sake of simplicity, the discussion below will be limited to single-rooted ACs.

2.5.1 Inference with OBDDs

In this section, we briefly discuss how to compute conditional probabilities using an OBDD representation of a (weighted) propositional formula, in time that is linear in the size of the OBDD.

To see how we can compute $P(\phi_d|_\sigma)$ using an OBDD [42, 45] in linear time, consider Figure 2.3. This OBDD has two types of internal nodes. The square nodes are labelled with decision variables $D_i \in \mathbf{D}$, and we refer to those nodes as *decision nodes*. The circular nodes are labelled with stochastic variables $T_{ij} \in \mathbf{T}$, and we refer to those nodes as *stochastic nodes*. The two leaf nodes are labelled with 0 and 1, which represent the values *false* and *true*, respectively. A path from the root of an OBDD to the leaf labelled with 1 corresponds to a (sub)set of the set of (variable, truth value) pairs that form a model for the formula encoded by the OBDD. This subset is sufficient for satisfying the formula, and its weight equals the sum of the weights of all models that are its supersets. Each model of the formula is defined by exactly one such path/(sub)set.

We map this OBDD to an AC to compute the probability that ϕ_d in Example 4.2.3 evaluates to *true* under a strategy σ as follows. The weights on the outgoing arcs of a stochastic node correspond to the probability that the variable that labels that node is *true* (for the solid, or *hi*, arcs) or *false* (dashed, or *lo*, arcs). We add a strategy σ on the OBDD by adding weights of 0 and 1 to the appropriate outgoing arcs of the decision nodes. The OBDD in Figure 2.3 does not reflect any specific strategy.

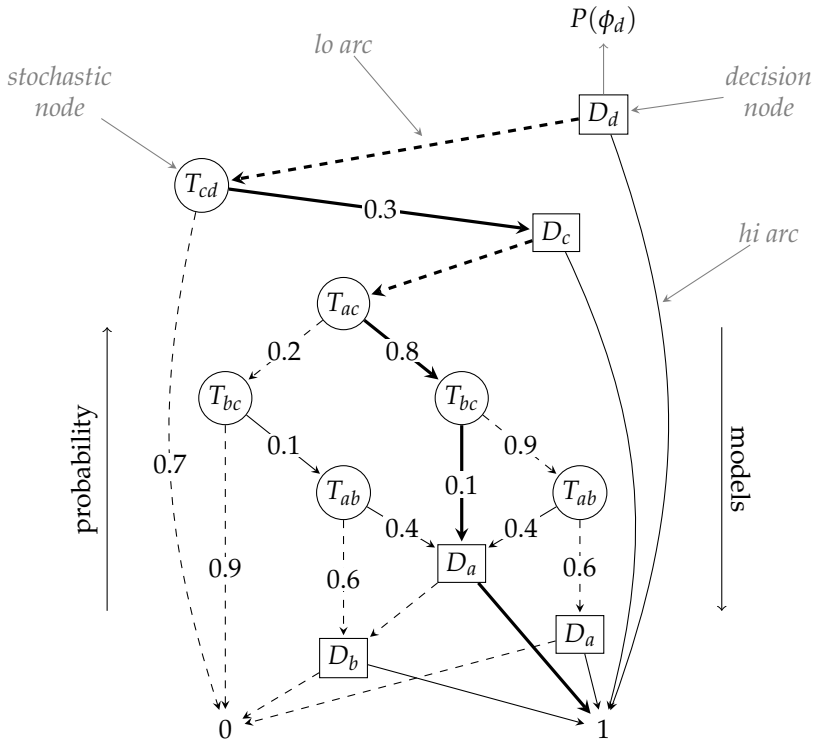


Figure 2.3: *An OBDD representation of Equation* 2.10*, mapped to an AC. This OBDD has variable order* $D_d \prec T_{cd} \prec D_c \prec T_{ac} \prec T_{bc} \prec T_{ab} \prec D_a \prec D_b$*. Bold arcs represent one of the models of* ϕ_d*:* $\{D_d := \bot, D_c := \bot, T_{ac} := \top, T_{bc} := \top, D_a := \top\}$*.*

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

$$v(r) := w(r) \cdot v\left(r^+\right) + (1 - w(r)) \cdot v\left(r^-\right), \tag{2.11}$$

where $0 \leq w(r) \leq 1$ represents the weight of the variable that labels r, 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 $v(root) = P\left(\phi|_\sigma\right)$. Note that it takes one *bottom-up* traversal of this AC to compute the score of the root.

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

Example 2.5.1 (WMC on an OBDD)**.** *Consider the OBDD in Figure 2.3. Suppose we want to compute* $P(\phi_d|_\sigma)$*, with* $\sigma := \{D_d := \bot, D_c := \bot, D_a := \top, D_b := \top\}$*).*

We label the dashed outgoing arcs of nodes labelled with D_d and D_c, as well as the solid outgoing arcs of nodes labelled with D_a and D_b with the value 1. Similarly, we label the solid outgoing arcs of nodes labelled with D_d and D_c, as well as the dashed outgoing arcs of nodes labelled with D_a and D_b with the value 0. Then, we perform a bottom-up sweep of the diagram to compute the score of each node, by computing the weighted sum of its children, as per Equation 2.11.

This yields a score of 1 *for the nodes labelled with D_a, D_b and T_{ab}, and for the right node labelled with T_{bc}. The left T_{bc} node has a score of* 0.1. *The T_{ac} and D_c nodes each have a score of* 0.82, *and the nodes labelled with T_{cd} and D_d each have a score of* 0.246. *Because the node labelled with D_d is the root node of the diagram, we conclude that* $P(\phi_d \mid \{D_d := \bot, D_c := \bot, D_a := \top, D_b := \top\}) = 0.246$.

2.5.2 Inference with SDDs

Just like OBDDs, we can use SDDs to compute the success probability of a residual propositional formula, by mapping the SDD onto an AC. Note that here, too, the time it takes to compute those probabilities is linear in the size of the DD, in this case an SDD. Since SDDs can be made more succinct than OBDDs through minimisation [24], the ACs we derive from them can also be more succinct, and therefore more efficient tools for repeated querying, than ACs obtained from OBDDs.

To compute $P(\phi_d|_\sigma)$ with an SDD, we construct an AC as follows. We replace the subs and the primes in Figure 2.4 with their weight according to the corresponding probability (in case of stochastic variables T) or their assignment (in case of decision variables D). We compute $P(\phi_d|_\sigma)$ in a bottom-up traversal of the SDD, where each disjunction node r takes score

$$v(r) := v\left(p^\ell\right) \cdot v\left(s^\ell\right) + v\left(p^r\right) \cdot v\left(s^r\right), \tag{2.12}$$

where p^ℓ (p^r) denotes the prime of the left (right) child of r, and s^ℓ (s^r) the sub of the left (right) child, and $v(p)$ or $v(s)$ is the weight of the terminal in the corresponding prime or sub, or the score of the sub formula rooted in that prime or sub. Again, $v(root) = P(\phi_d|_\sigma)$. And again, in the interest of brevity, in the remainder of this work, we will sometimes abuse terminology and refer to the SDD when we actually mean the AC that the SDD is mapped onto.

Example 2.5.2 (WMC on an SDD)**.** *Consider the SDD in Figure 2.4. Suppose that, again, we want to compute* $P\left(\phi_d|_{\{D_d:=\bot, D_c:=\bot, D_a:=\top, D_b:=\top\}}\right)$.

The disjunction node indexed with 10 *has score* $0 \cdot 0.4 + 1 \cdot 1 = 1$, *and the one indexed with* 11 *has score* $1 \cdot 0.4 + 0 \cdot 0 = 0.4$. *Continuing in our bottom-up traversal*

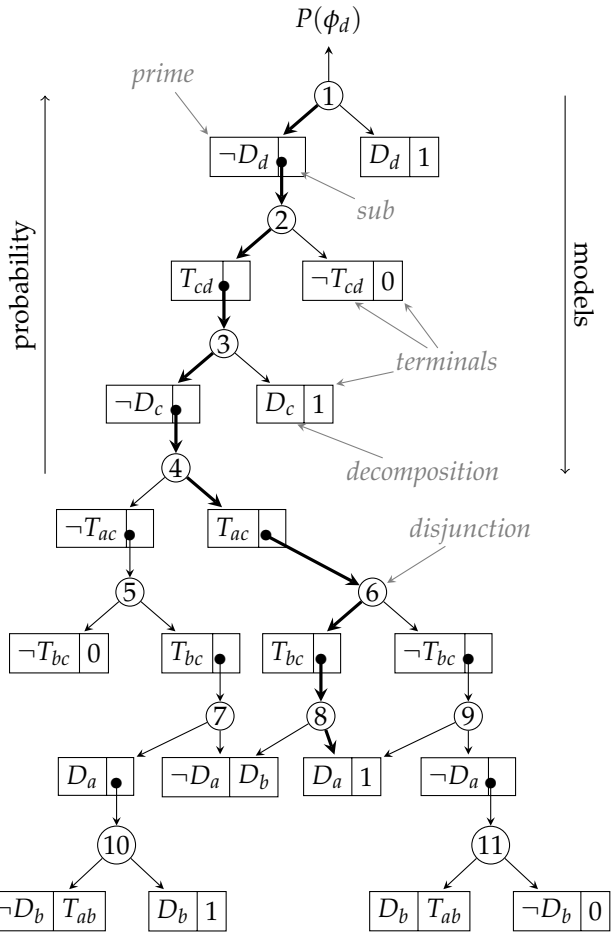


Figure 2.4: *An SDD representation of Equation* 2.10 *with variable order* $D_d \prec T_{cd} \prec D_c \prec T_{ac} \prec T_{bc} \prec D_a \prec D_b \prec T_{ab}$. *Bold arcs represent one of the models of* ϕ_d: $\{D_d := \bot, D_c := \bot, T_{ac} := \top, T_{bc} := \top, D_a := \top\}$. *Probabilities are omitted. Disjunction nodes are indexed for reference.*

and applying Equation 2.12, *disjunction nodes* 6–9 *each have score* 1, *while disjunction node* 5 *has score* 0.1. *Disjunction node* 4 *has score* $0.2 \cdot 0.1 + 0.8 \cdot 1 = 0.82$, *and so does node* 3. *Finally, node* 2 *has score* $0.3 \cdot 0.82 + 0.7 \cdot 0 = 0.246$, *and so does node* 1. *Since node* 1 *is the root of the SDD, we find that* $P\left(\phi_d|_{\{D_d:=\bot, D_c:=\bot, D_a:=\top, D_b:=\top\}}\right) = 0.246$.

2.6 Probabilistic logic programming

As the previous section shows, DDs such as OBDDs and SDDs represent convenient data structures for performing tractable probabilistic inference. This begs the question: how do we go from a mathematical model of a probability distribution to a DD-representation of that distribution?

Note that this question really has two components. First, we need some convenient way to model the probability distribution in a computer. Second, we then need to compile this model into a DD.

To illustrate the first challenge, recall the two example problems described in Section 1.1: spread of influence and power grid reliability. Both of them are formulated on probabilistic networks (where edges exist with a certain probability), and both of them require some form of reasoning about paths in those networks. As a consequence, the resulting probability distributions can become quite complex, since there are often many different paths between two nodes in a real-world network, especially in the case of social networks, which are the types of network that spread of influence problems are formulated on. Additionally, these paths may partially overlap, divert from each other, and maybe even join again later on. This makes the resulting probability distributions quite non-trivial to define.

This is a problem, since we aim to develop solving methods for problems like the spread of influence problem, that are *convenient* to use. We want it to be easy for a user to specify them, and to not require (much) technical knowledge, aiming to democratise the technology that we develop in this dissertation as much as possible, making it as accessible as possible to anyone who needs it.

In this section we discuss a convenient tool for modelling probability distributions: the probabilistic logic programming language ProbLog [52]. This Prolog-based language provides a simple way for users to model probability distributions, and is particularly suited for modelling distributions that arise from probabilistic networks, thus addressing the first challenge. Conveniently, ProbLog has functionality for compiling these distributions into several different DDs built in, thus addressing the second challenge. In the remainder of this section, we give a brief introduction to the logic programming Prolog, which is a precursor to ProbLog, and then to ProbLog itself.

2.6.1 Logic programs

Prolog is a *rule-based*, *logical* programming language that is *declarative* in nature. Declarative programming paradigms are very user-friendly in the following way. *Imperative* programming languages (such as C, C++, Java, Python, Go, etc.) require

the user to specify *how* the computer should solve a problem. Declarative programming languages (like Prolog, Datalog and SQL), on the other hand, only ask the user to specify everything they *know* about the problem, and then simply ask a question. The computer will then figure out how to get to the answer.

Our choice to build on Prolog is motivated by the fact that it is declarative. While a study of how different types of intended users experience modelling problems in Prolog compared to other languages is outside the scope of this work, we are confident that its declarative nature can appeal to a wide range of users. In Section 4.3, we present a language to model constraint optimisation problems by querying databases. Arguably the most-used programming paradigms for these elements are *constraint programming (CP)* for modelling linear and non-linear constraints (see Section 3.3), *mixed integer programming (MIP)* for modelling linear constraint optimisation problems (see Section 3.4) and SQL for querying databases. Since all three of these paradigms are declarative in nature, we expect our target audience to find a declarative programming tool convenient to use and easy to learn. After all, there is probably a good reason that declarative languages are so popular, and due to their popularity, a potential user is likely to be familiar with the declarative programming paradigm.

It is important to us that our tools are easy to use by a wide variety of people, even if they have no previous coding experience, because that helps to democratise computing power and technology. However, there are possible downsides such as added expressive complexity for specifying certain tasks and less speed. At the same time, it is a very common practice in Computer Science to design languages for specific goals, and thus with limited applicability, so these downsides are only a natural consequence of the fact that we design task-specific tools.

Recall the goal that we specified in Section 1.3: to find SCP solving methods with reasonable trade-offs between convenience, generality and speed. Our reason for choosing the declarative ProbLog (probabilistic Prolog) language as a basis for our new SCP modelling language is primarily rooted in the first criterion: convenience. As mentioned above, the basics of Prolog, which provide the user with enough expressibility to model a wide range of problems, are quick and easy to learn. Additionally, it is very convenient for modelling *relations* between entities. In this work, our focus is on SCPs that are formulated on probabilistic networks. Networks are very easy to model in Prolog, because edges can be seen as relations between nodes. To illustrate this, we now first present a few basics about the syntax and semantics of Prolog, and then provide a small example.

Prolog is a *rule-based* programming language. To construct rules, we can use *terms* and *predicates*. Terms are either *constants*, or *variables*. Constant symbols in

Prolog start with a lower case letter, or are a string enclosed in single quotes. Variable symbols are capitalised and can take arbitrary constants as values. Prolog uses *predicates* to express relationships between constants or variables, in the same way as the predicates in first-order logic (Section 2.2.2). The name of a predicate also starts with a lowercase letter, and is followed by comma-separated terms in brackets: the *arguments* of the predicate. Any Prolog predicate can also be negated.

Each rule is of the form: `Head :- Body.`, which means "`Head` is *true* if `Body` is *true* ." The body contains one or more predicates and terms, which are known as *goals*, separated either by "`,`" for conjunction, or "`;`" for disjunction. We also refer to these rules as *clauses*. A special property of the rules in Prolog is that the head only contains one predicate, which makes these clauses *Horn clauses* [81].

In addition to rules, there are also facts. An example is: `dir(alexa,behrouz).`, which is a shorthand for `dir(alexa,behrouz) :- true.`, and means that there is a directed relationship between the constant `alexa` and the constant `behrouz`.

The user can use rules and facts to describe everything they know about the problem, and ask questions by specifying *queries* of the form `?- influences(alexa,daniel).`. Queries can be seen as rules without a head, where we ask if the goals in the body are *true*. The Prolog system then uses an inference process called *selective linear definite (SLD) clause resolution*. The resolution process tries to find constants in the rest of the program that can be used to substitute variables such that rules are satisfied (the arguments in the predicates in the head and the body match) and new facts are proven from these rules.

We can now formulate a small Prolog program that describes some relationships between some people.

Program 2.1: *A Prolog program describing influence relationships between four people.*

```
   % Relation facts
1. dir(alexa,behrouz).
2. dir(alexa,claire).
3. dir(behrouz,claire).
4. dir(claire,daniel).

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

   % Query
7. ?- influences(alexa,daniel).
```

Example 2.6.1 (A simple Prolog program)**.** *Consider the small Prolog program in Program 2.1. In lines 1–4, it describes the direct influence relationships between four peo-*

ple who are represented by the constants `alexa`, `behrouz`, `claire` *and* `daniel`. *Line 6 serves to make the relationships symmetric, and line 7 is a query asking if Alexa influences Daniël. Evaluating this program tells us that she does not, since neither the predicate* `dir(alexa,daniel).` *nor* `dir(daniel,alexa).` *is in the knowledge base of relation facts, meaning that the rules in line 5 and 6 cannot be used to prove that Alexa influences Daniël.*

The reader may have noticed that the fact that Alexa *does not* influence Daniël is proved by *failing to prove* that she *does*. This property, absence of truth meaning negation of truth, is called the *closed-world assumption* and central to the semantics of Prolog. The closed-world assumption is very powerful because it drastically limits the size of program you need to model a problem.

Note that in Program 2.1, lines 1–4 are essentially first-order logic formulae. Using predicates, we could rewrite these lines as $R(a,b) \wedge R(a,c) \wedge R(b,c) \wedge R(c,d)$, where $R(a,b)$ indicates that there is a directed relationship between constants a (Alexa) and b (Behrouz), and analogously for the other predicates and constants. We could replace each predicate by a Boolean variable, and then rewrite these lines as as the *propositional* logic formula $R_{ab} \wedge R_{ac} \wedge R_{bc} \wedge R_{cd}$, where R_{ab} is a Boolean variable that is *true* iff there is a directed relationship from Alexa to Behrouz, and analogously for the other variables. This latter expression is not written in first-order logic, since it does not use any predicates or quantifiers.

Lines 5 and 6, on the other hand, are written in *first-order* logic and cannot be rewritten by replacing predicates with Boolean variables. We can rewrite line 5 as $\forall X, Y.\,(I(X,Y) \vee \neg R(X,Y))$, where X and Y are variables that could be replaced by a, b, c or d, and $I(X,Y)$ is a predicate that is *true* iff X influences Y.

It helps our discussion later on to introduce the concept of *grounding* a logic program. A predicate that does not contain any variables, is called a *ground predicate*. A ground logic program is one in which all predicates are ground by substituting the variables by constants. Note that this corresponds to turning the first-order logic formula that is implied by the program into a propositional formula.

Program 2.2: *Ground relation rules.*

```
5.1. influences(alexa,behrouz).
6.1. influences(behrouz,alexa).
5.2. influences(alexa,claire).
6.2. influences(claire,alexa).
5.3. influences(behrouz,claire).
6.3. influences(claire,behrouz).
5.4. influences(claire,daniel).
6.4. influences(daniel,claire).
```

Example 2.6.2 (Ground Prolog program)**.** *For example, we could ground the program in Program 2.1 by replacing lines 5 and 6 by Program 2.2. Here, we have replaced the* `Xs` *and* `Ys` *in these lines by combinations of constants that are allowed according to the relation facts in lines 1–4. Note again that* `influences(alexa,daniel).` *is not in the ground program, and thus we cannot prove that Alexa influences Daniël, as is asked in line 7 of Program 2.1.*

Note that in the example above, we can replace the ground predicates of lines 5.1–6.4 by converting them into Boolean variables, and write these lines as $I_{ab} \wedge I_{ba} \wedge I_{ac} \wedge I_{ca} \wedge I_{bc} \wedge I_{cb} \wedge I_{cd} \wedge I_{dc}$, which is a propositional logic formula.

Grounding an entire Prolog program requires substitution of variables into all possible (combinations of) constants that are allowed by the program. This typically results in not only a large program, but one with many ground facts that are irrelevant for answering the query, as is very clearly shown in Example 2.6.2, where all eight ground relation rules turn out to be irrelevant for answering the query. Therefore, grounding typically happens in a top-down rather than bottom-up manner, such that only those predicates are ground that are needed for answering the query.

A detailed description of Prolog, and of SLD and techniques for efficient grounding, is outside the scope of this work. For more information on the syntax, semantics and inner workings of Prolog, we refer the reader to the literature, *e.g.*, Peter Flach's *Simply Logical: Intelligent Reasoning by Example* [66].

2.6.2 Probabilistic logic programs

The probabilistic logic programming ProbLog [52] is a language for programming probability distributions, built on Prolog. We can turn the Prolog program in Program 2.1 into a ProbLog program by adding probabilities to facts and rules. For example, by replacing lines 1–4 by the following, we can make the influence relationships between people probabilistic:

Program 2.3: *Probabilistic facts.*

```
1. 0.4::dir(alexa,behrouz).
2. 0.8::dir(alexa,claire).
3. 0.1::dir(behrouz,claire).
4. 0.3::dir(claire,daniel).
```

Here, we assume that the associated probabilities are independent of each other. We can think of these probabilistic facts as the (positive) literals in a probability-weighted propositional formula, as described in Section 2.2.3.

We can think of a ProbLog program as one that defines a distribution of 'underlying' Prolog programs, or *possible worlds*, where each probabilistic fact is non-deterministically present in a program that was randomly sampled from this distribution, according to that fact's associated probability. This probability is the one that annotates the corresponding rule or fact. A probabilistic version of the rule in line 5 in Program 2.1 would be, *e.g.*, `0.2::buys(X) :- influences(Y,X), buys(Y).`. This is shorthand for `0.2::buys(alexa) :- influences(alexa,alexa), buys(alexa).`, `0.2::buys(alexa) :- influences(behrouz,alexa), buys(behrouz).`, and so on. Each of these rules has a probability of 0.2 to be included in a Prolog program that is randomly selected from the distribution defined by the ProbLog program. In other words: if there is a way in which the body of the rule can be made *true*, the chance that the head is *true*, is 0.2.

Following the closed-world assumption, we assume that the probability that a fact is true (and thus present in a Prolog program randomly sampled from the distribution) and the probability that the negation of that fact is true (and thus not present in that Prolog program), sum up to one. For example: we assume that the probability that Alexa *does not* influence Behrouz is $1 - 0.4 = 0.6$. Now, instead of asking *if* Alexa influences Daniël, line 7 now asks *with what probability* she does. We also refer to this probability as the *success probability* of the query.

The success probability of a query Q is defined as follows:

$$P\left(Q \mid \mathscr{P}\right) = \sum_{v \in \mathscr{P} \mid v \models Q} P(v), \tag{2.13}$$

where $\mathscr{P}$ is a ProbLog program, v is a *possible world*, *i.e.*, a Prolog program that can be obtained from $\mathscr{P}$ by including all deterministic facts and rules and including a subset of the probabilistic facts and rules, and $P(v)$ is the probability that v is randomly sampled from $\mathscr{P}$. Here, we use the notation $v \in \mathscr{P}$ to indicate that v is a possible world that can be obtained from $\mathscr{P}$, and the notation $v \models Q$ to denote that Q is a *logical consequence* of v. Note that we simply sum the probabilities of all possible worlds in which $Q = \top$. In the example above, there are no world in which the query is *true*, so the probability that Alexa influence Daniël is 0.

Because of the assumption that the probabilities annotating the probabilistic facts are mutually independent, we can follow Sato's distributions semantics [160] and define the probability of a possible world as:

$$P(v) = \prod_{f \in v} P(f), \tag{2.14}$$

where f is a (probabilistic) fact in the *ground* Prolog program v, and $P(f)$ its associated probability, according to $\mathscr{P}$.

Equations 2.13 and 2.14 should look familiar to the reader, as they bear a clear and non-accidental similarity to Equation 2.1, which defines the weighted model count of a literal-weighted propositional formula. Indeed, we can think of ProbLog as a tool to program literal-weighted propositional formulae, whose model counts reflect success probabilities of the associated queries.

This is also reflected in the inner workings of ProbLog. Like Prolog, ProbLog uses SLD resolution in order to provide an answer to the query, which comes in the form of that query's success probability. Conceptually, in doing so, it generates all possible ground logic programs that one could create from the probabilistic logic program, and summarises them into a DD, which can then be used to compute a query's success probability, as described in Section 2.5. That knowledge compilation step helps us to achieve our requirement that SCP solving methods are fast, as well as convenient.

However convenient probability distributions are to model using ProbLog, they still require the user to be smart about *how* they model the problems exactly. As with many problems in Computer Science, we expect the way that we model an SCP to have a large impact on the speed with which we can solve it. We will reflect on this some more in the next chapter.

2.7 Conclusion

In this chapter we discussed topics related to probabilistic inference and how to express probabilistic models and queries using logic and a probabilistic logic programming language. We motivated why, in this work, we chose to model probability distributions using probability-weighted propositional formulae, which we discussed in Section 2.2, along with related topics such as first-order logic and relevant complexity classes. We specifically described how we can apply *weighted model countings (WMCs)* to weighted propositional formulae to compute the probability that our probabilistic model is in a certain configuration.

In order to put the (sub-)problems described in this dissertation into context, we also gave a high-level overview of the different probabilistic inference tasks known in the probabilistic reasoning literature. Specifically, we discussed the $\mathcal{NP}$-complete task of determining the most likely truth evaluation of each of a set of queries, given a truth value assignment to the complementary set of variables in the probabilistic model. The decision version of this max-inference task, where we do not ask what the most likely truth evaluation is, but rather if a given assignment is more likely than a certain threshold value, is relevant to this work, as it can be seen as asking if a specific strategy σ satisfies Equation 1.1.

We then discussed the $\#\mathcal{P}$-complete sum-inference task, which is to compute the success probability of a query, such as the ones described in Section 2.6.2. Finally, we discussed the $\mathcal{NP}^{PP}$-complete mixed-inference task that aims to find an assignment to a subset of variables, given truth assignments to another, disjoint set, and without knowing the truth values of the variables in the complement of these two sets. Provided we can encode the constraints into the probabilistic model, we can cast SCPs as instances of MMAP.

We then argued that knowledge compilation techniques can help us achieve our goal of developing exact SCP solving methods that strike a reasonable balance between convenience, generality and speed. Specifically, we argued that DD representations of probability distributions can be compact and can be used for tractable probabilistic inference. We closed this chapter with a description of probabilistic logic programming language ProbLog, a tool that not only allows us to conveniently model probability distributions that arise from the probabilistic networks on which the SCPs that we study in this work are formulated, but also provides support for converting the resulting probabilistic logic programs into DDs for fast inference.

The focus of this chapter was on the probabilistic reasoning part of SCP solving. In the next chapter, we focus on the constraint optimisation part of SCP solving. Together, the techniques and frameworks described in this chapter and the next will help us develop SCP solving tools in Part II of this dissertation.

3

Programming paradigms for optimisation

3.1 Introduction

Recall from Section 1.3 that exact *stochastic constraint (optimisation) problem (SCP)* solving involves two components: probabilistic inference and search. While in the previous chapter we discussed how to model probability distributions and typical inference tasks, in this chapter we discuss programming paradigms that help us model probabilistic optimisation problems, and on programming paradigms that help us traverse the search space of possible strategies efficiently. Specifically, in the next section, we discuss a ProbLog-derived tool for solving decision problems that involve probabilities, DT-ProbLog [178]. Then, in Sections 3.3 and 3.4, we discuss the fields of *constraint programming (CP)* and *mixed integer programming (MIP)* as optimisation paradigms that allow us to efficiently traverse search spaces.

Recall also from Section 2.1 that the work presented in this dissertation is broad in scope, meaning that we can build on many and varied existing tools, each with its own set of tunable parameters. Consequently, bringing these tools together in an effort to solve SCPs may result in complex systems that may require specialised configurations for different application domains for them to perform well. Therefore, later in this chapter, we provide a brief introduction to the paradigms of *programming by optimisation (PbO)* and *automated algorithm configuration (AAC)*, which enable us to find these configurations. We close this chapter in Section 3.6 with a brief conclusion.

3.2 Decision-theoretic probabilistic logic programming

We start this overview of programming paradigms for optimisation with yet another Prolog-based modelling language (recall Section 2.6): DT-ProbLog (short for *decision-theoretic ProbLog*) [178]. This extension of ProbLog [52] was proposed in 2010 in order to solve decision problems in which the aim is to maximise some kind of utility that is associated with (derived) facts. To this end, DT-ProbLog extends the ProbLog syntax and semantics by adding *decision variables* and functionality to assign a *utility* to events.

By setting decision variables to *true*, we can introduce facts to the program. Utilities associate with facts can be used to define the relative costs and benefits of facts and consequences. We illustrate this with the following example.

Program 3.1: *A DT-ProbLog program describing a spread of influence problem.*

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

    % Probabilistic relation facts
3.  0.4::dir(alexa,behrouz).        0.8::dir(alexa,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).

    % Utilities
```

```
 8. utility(buys(P), 1) :- person(P).
 9. utility(gets_free_sample(P), -1) :- person(P).

    % Probabilistic customer conversion rules
10. buys(X) :- gets_free_sample(X).
11. buys(X) :- influences(Y,X), buys(Y).
```

Example 3.2.1 (A simple DT-ProbLog program)**.** *We can modify and extend the probabilistic logic program described by Programs 2.1 and 2.3 by adding decision variables and utilities, resulting in Program 3.1. Note that this program describes a spread of influence problem like the one described in Section 1.1.*

Here, line 7 specifies the decision variables — a functionality that is new in DT-ProbLog. Lines 8 and 9 specify utilities for (derived) facts. Specifically, they state that converting a person into a customer who buys our product, has a utility of 1 per person (line 8). However, it costs 1 to give a person a free sample of our product (line 9).

DT-ProbLog does not provide a specific syntax for querying the program, but returns grounded facts that represent the decisions that have to be made in order to maximise the expected utility, along with the value of that utility. This utility is computed by simply summing the (expected) utilities present in or derived from the DT-ProbLog program. In the example above, the output of DT-ProbLog is `gets_free_sample(alexa) := True, gets_free_sample(behrouz) := False, gets_free_sample(claire) := False, gets_free_sample(daniel) := False, Expected utility: 1.4984`.

It comes to this conclusion based on an inference process that uses *algebraic decision diagrams* (ADDs) [11], data structures that are very similar to the *arithmetic circuits (ACs)*, described in Section 2.5, and can be used for optimisation problems. These can be constructed from *decision diagrams (DDs)* like *ordered binary decision diagrams (OBDDs)* and *sentential decision diagrams (SDDs)*.

These properties make DT-ProbLog an attractive programming paradigm for stochastic optimisation problems, for us to build on in our efforts of designing SCP solving methods.

Note that we do have to build on DT-ProbLog before we can use it to solve SCPs, since DT-ProbLog itself does not support constraints, only a maximisation criterion.

3.3 Constraint programming

As discussed in Section 1.3, in order to solve SCPs we do not just need efficient probabilistic inference, we also need an effective search mechanism. One of the

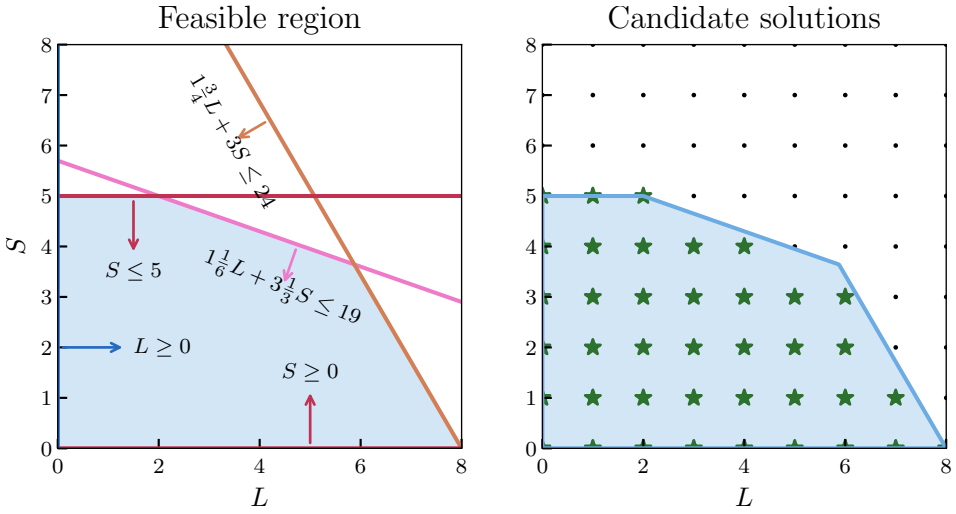


Figure 3.1: *Graphical representation of the constraints in the power grid reliability problem described in Example 3.3.1. The horizontal axis shows the number of power lines that we can choose to reinforce, and the vertical axis shows the number of stations. Stars indicate (L, S) combinations that satisfy the constraints and are thus candidate solutions of the optimisation problem.*

paradigms we employ in this work is CP. Here, we recall basic concepts of CP. For details we refer the reader to the literature, *e.g.*, the *Handbook of Constraint Programming* [154].

Constraint satisfaction problems (CSPs) are modelled using a set of variables $\mathbf{X} = \{X_1, \ldots, X_{|\mathbf{X}|}\}$, each of which is associated with a domain $dom(X_i)$, a set of constraints C on (subsets of) these variables and an objective function $f(\mathbf{X})$. In the context of this work, the objective function can be, *e.g.*, $\arg\max_\sigma P(\phi|_\sigma)$, or the linear constraint optimisation function in the next example, which we will consider for the scope of this section and the next, to illustrate some key concepts of optimisation techniques.

Example 3.3.1 (A power grid reliability problem)**.** *Recall the power grid reliability problem from Section 1.1. Suppose we drastically simplify this problem by defining some measure of 'reliability' and assuming that this is some linear combination of the number*

of power lines and stations that we have reinforced.

$$\textit{maximise reliability} \quad R = \frac{5}{4}L + S \tag{3.1}$$

$$\textit{subject to budget constraint} \quad \frac{7}{6}L + \frac{10}{3}S \leq 19 \tag{3.2}$$

$$\textit{and time constraint} \quad \frac{7}{4}L + 3S \leq 24 \tag{3.3}$$

$$\textit{with} \quad L, S \in \mathbb{N}_0, L \leq 10, S \leq 5 \tag{3.4}$$

where L and S are the numbers of power lines and power stations that we have reinforced. They require different amounts of money ($\frac{7}{6}$ million and $\frac{10}{3}$ million of your favourite currency, respectively), with a budget of 19 million. They also require different amounts of time per unit to reinforce ($\frac{7}{4}$ and 3 months, respectively), where we assume that there is only one team that does the reinforcements and they can only reinforce one unit at a time, and we have two years to complete the project. Finally, our small network has ten power lines and five power stations.

We provide a graphical representation of this problem in Figure 3.1.

In general, variables can have different kinds of domains (typically Boolean, categorical, ordinal, integer- and real-valued). In the example above, the variables have integer domains. In the SCP problems studied in this work, the relevant variables are all Boolean decision variables. However, the choices we make in Chapter 5 to model SCPs in CP solvers, result in constraint programs that also contain variables with real-valued domains.

In the next subsections, we continue with a discussion of two orthogonal solving techniques for CP: *search* and *inference*. Intuitively, the search process determines how the solver traverses the search space, by assigning values to variables to see if those variable assignments can be extended to a solution. The inference process, called *propagation*, helps to prune the resulting search tree by efficiently *inferring* consequences of these assignments. Propagation detects which values in the domains of free variables have to be removed from those domains because they are *inconsistent* with the assignments made by the search process, and thus can never lead to a solution.

This brings us to another important dichotomy that is identified in the CP literature: modelling versus solving. CP solvers provide the user with a range of different constraints to choose from. Each of these constraints has an associated *constraint propagator*: an algorithm that can efficiently *infer* consequences when variable domains change, due to choices made during the search process. We will describe propagation in more detail later in this section.

Since not all propagators are equally powerful, meaning that they cannot always make the same inferences using the same amount of computational effort, the actual choice of which constraints to use in *modelling* the problem can have a significant impact on how efficiently the problem can then be *solved*. Therefore, in order to successfully employ CP techniques, the user has to be smart about how they model the problem. A detailed discussion of (how to make good) modelling choices is outside the scope of this work, but we do now continue with a brief overview of search and propagation techniques.

3.3.1 Backtracking search

CP solvers employ one core algorithmic approach to finding a solution (or refutation) to the input problem: *backtracking search*. While CSPs can also be solved by techniques like *local search* and *dynamic programming*, we focus on backtracking search.

In this work, we require the search algorithm to be *complete*, meaning that it guarantees that a solution will be found if one exists, that it will show that a problem does not have a solution if none exists, and that it can be used to find a provably optimal solution. The backtracking search approach is complete (unlike most local search algorithms) and typically preferred over the (complete) dynamic programming approach, because backtracking requires only a polynomial amount of space, while dynamic programming might require exponential amounts of time and space [154, Chapter 4]. Backtracking search was first proposed in the 1960s [49, 73] and is still the main driver of modern-day CP solvers.

Typical backtracking search uses a depth-first search, inducing a *search tree* to find a solution to the CSP (or an optimal solution in an optimisation setting). The solver repeatedly selects an *unbound* (or *uninstantiated*, or *free*) variable X and assigns to it a value $a \in dom(X)$ (or a range or interval of values in case, *e.g.*, $dom(X) \subseteq \mathbb{R}$), thus building a *partial solution*. Repeatedly selecting an unbound variable and assigning a value to it is called *branching* and induces a *search tree*.

If the domain of a variable X is reduced to a single value in this process, we consider X to be *fixed* (or *bound*) to that value. If a fixed variable or set of fixed variables violate(s) a constraint, we have encountered a *failure*: a partial assignment that cannot be extended to a solution. When this happens, the solver *backtracks* to an earlier point in the search tree by undoing variable assignments. The two main backtracking methods are *chronological backtracking*, where the solver simply returns to the closest node on the current path in the search tree where not all outgoing branches have been explored yet, and *non-chronological backtracking* or *backjumping*, where the solver 'jumps' back to a higher level in the search

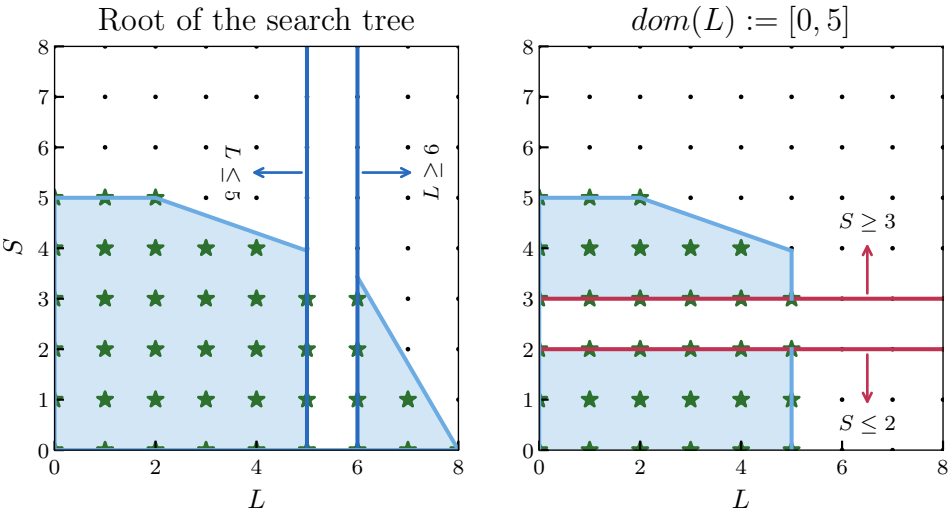


Figure 3.2: *Graphical representation of branching in the root and in the left child of the root in Figure 3.3.*

tree, thus skipping one or more nodes whose outgoing branches have not all been explored yet.

Let us now use the constraint optimisation problem in Example 3.3.1 to illustrate backtracking search in CP.

Example 3.3.2 (Branching and backtracking in CP)**.** *The constraint optimisation problem described in Example 3.3.1 has two variables: the number of power lines that we choose to reinforce L, and the number of power stations that we choose to reinforce S. A search algorithm will thus have to traverse the search space of (L, S) combinations to find the combination that satisfies Equations* 3.2 *to* 3.4 *and maximises Equation* 3.1*.*

Figure 3.3 illustrates just one way of traversing that search space. In each node, the domain of either L or S is split (roughly) in half, branching left on the lower side, and right on the higher side of the domain. Going down along a branch on the tree, nodes in which we split on the domain of L and the domain of S alternate. Let us assume we traverse the tree from left to right, and let us in this example only consider the left part of the search tree (rooted at the left child of the root).

Following the left-most branch, we first branch on $L \leq 5$ and then on $S \leq 2$. We can visualise this as adding extra constraints to focus on a specific region of the search space, as is illustrated in Figure 3.2. Note that, at this point in the search tree, with $dom(L) = [0, 5]$ and $dom(S) = [0, 2]$, all (L, S) combinations satisfy the constraints in Equations 3.2 *to* 3.4*. For the sake of legibility, we have omitted further descendents of this node and just listed the optimal combination in these domains: $R(L = 5, S = 2) = 8.25$.*

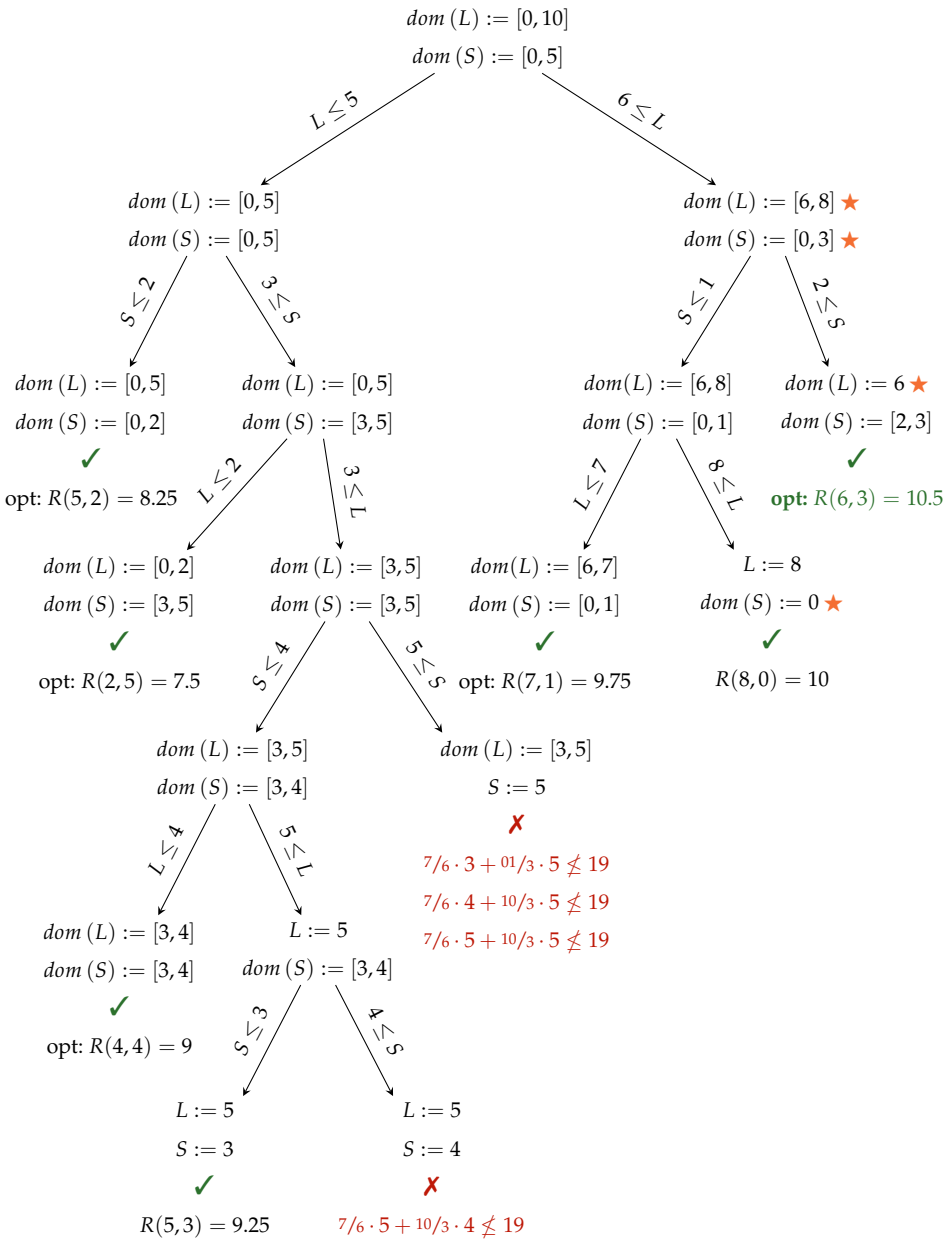


Figure 3.3: *An example of a small search tree for the constraint optimisation problem described in Example 3.3.1. We use ✗ to indicate failures and ✓ to indicate a solution to the constraints in Equations 3.2 to 3.4. Domains marked with ★ are relevant for Example 3.3.3.*

After finding this solution, we backtrack to the parent node and branch right, on $3 \leq S$ *to continue the search. We continue branching and finding solutions, until the node in which both domains have been reduced to just one value:* $L := 5$, $S := 4$, *which is a combination that violates the budget constraint in Equation* 3.2. *We backtrack and continue the search for a better solution than the best one found so far (*$R(L = 5, S = 3) = 9\frac{1}{4}$*), until we have traversed the entire search tree.*

Typically, CP solvers employ a process called *branch and bound*, which uses cheap-to-compute heuristics to determine if branches of the search tree can still contain a better solution than the best solution found so far, or even any solution at all. This helps to keep the size of the search tree from becoming too large.

In the above example, we made a rather arbitrary choice to alternate between splitting the domains of L and S and to always split them (roughly) in half. Which variable (and value or domain) to branch on next, is typically decided by a *branching heuristic*. The aim of these heuristics is to find a variable and value/domain branching order that minimises the size of the search tree, and thus likely also the running time of the solver. Since even finding the *first* variable of an *optimal* variable order is at least as hard as solving the input problem itself [107], finding and using the optimal variable order is infeasible.

Therefore, CP solvers use heuristics that give no guarantees of optimality to decide which (variable, value/domain) pair to branch on next during the search. These heuristics can either be universal or domain-specific, and can be either static (determined before the search starts) or dynamic (determined during the search). In this work we use existing universal branching heuristics, and present new ones that are either static or dynamic and designed specifically for a new constraint propagator that we introduce in Chapter 6. Additionally, we introduce static, domain-specific branching heuristics in Chapter 7.

3.3.2 Constraints, consistency and pruning

The second mechanism that drives typical CP solvers is *constraint propagation*, or *inference*, which is orthogonal to search. Informally, constraint propagation helps the solver to eliminate *inconsistencies*, which are values in domains of free variables that cannot be part of a solution to the CSP, given the current assignment to the bound variables. By enforcing consistency, constraint propagation helps the solver to avoid branching on (variable, value) pairs that are inconsistent with the current partial assignment and the constraints, and thus to prune parts of the search space that do not contain any solutions, reducing the size of the search tree.

Constraint propagation operates as follows. After each time a backtracking search solver branches on a (variable, value/domain) pair, propagators update the domains of the remaining unbound variables by removing values that would violate the constraints of the problem instance, given the current partial solution. This helps to shrink the domains of the remaining free variables, and thus to prune the search tree. If the domain of a variable X is reduced to a single value in this process, this variable is automatically bound to that value. If for any variable X we find that $dom(X) = \varnothing$ after propagation, we have found a failure and must backtrack. Note that constraint propagation can also be called *before* the search starts, in which case it serves as a preprocessing step to reduce the sizes of variable domains [154, Chapter 4].

An important type of consistency is that of *generalised arc consistency (GAC)*. A (variable, value) pair (X, x) with $x \in dom(X)$ is considered *generalised arc consistent (GAC)* with respect to a constraint $c \in C$ iff there exists an assignment in the current domains of the other variables in the scope of c that satisfies c and in which $X = x$ [114]. Propagation establishes GAC for a constraint c if all remaining values of all variables in the scope of c are GAC.

Example 3.3.3 (Search space pruning in CP)**.** *Note that the left side of the search tree in Figure 3.3 is quite large, despite us omitting many nodes for reasons of legibility and instead simply giving the optimal value of the objective function possible for the domains of L and S in that node. Crucially, in Example 3.3.2, we did not perform any propagation. Had we done any propagation, we could have pruned the search space.*

Consider the right branch of the root of the search tree in Figure 3.3. It branches on $6 \leq L$*, reducing L's domain from* $[0, 10]$ *to* $[6, 10]$*. A quick glance at Figure 3.1 tells us that any L that exceeds* 8*, violates the time constraint in Equation* 3.3*. A constraint propagation algorithm may detect that this is the case, and exclude the values* 9 *and* 10 *from L's domain, which we have done in the right child of the root of the search tree. Similarly, for values of L that are larger than* 4*, there are no solutions in which* $S = 4$ *or* $S = 5$*. Since we branched on* $L \geq 6$ *in the right child of the root, we can also exclude the values* 4 *and* 5 *from the domain of S, which we have also done in that node.*

Note that these actions kept the domains of L and S GAC with respect to the budget constraint in Equation 3.2*, and the time constraint in Equation* 3.3*, respectively. The domains in Figure 3.3 that have been pruned using propagation, are marked with* ★*.*

3.3.3 Local and global constraints

A detailed discussion of backtracking and propagation, as well as of other techniques, such as randomisation, restarts, local search, value selection heuristics

and more, is outside the scope of this work. However, we must mention a few important concepts about *local* and *global* constraints.

The main difference between local and global constraints is their scope. Local constraints are between a *fixed* number of variables. For example: $X < Y$ is a local constraint, since it is always between two variables.

Global constraints, on the other hand, can involve an arbitrary subset of the variables present in a problem. Arguably the best-known global constraint is the *AllDifferent* constraint, which requires all variables in the scope of the constraint to have a different value. Note that, contrary to the constraint above, the size of the scope of an AllDifferent constraint is not determined by the form that the constraint takes.

The constraint that is central to this work, the one in Equation 1.1, is a global constraint. The constraints in Equations 3.2 to 3.4 on the other hand, are local constraints.

Note that enforcing GAC for global constraints can be more computationally expensive in both time and space than enforcing consistency on local constraints, but also potentially more powerful in the amount of pruning it makes possible. Some work has been done on *decomposing* global constraints such that the decomposition has the same propagation power as the original global constraint, meaning that the decomposition prunes the same values from the domains of the involved variables as does the original global constraint. This is possible for some global constraints, but not for all (at least not in polynomial time and polynomial space). We refer the interested reader to the literature on which global constraints can be decomposed, whether those decompositions preserve the solutions to the CSP, whether they preserve GAC, and whether they preserve the time and space complexity of enforcing GAC, *e.g.*, [10, 15] and [154, Chapter 3].

3.3.4 Advantages of CP technology

CP solvers typically support many different types of constraints, where each constraint has a dedicated propagator, designed specifically to propagate changes in domains of variables in the scope of that constraint efficiently. As one propagator removes values from a variable's domain, this may trigger other propagators to also remove values from domains. Thus, even though propagators themselves are designed to solve specific constraints, their interaction can be quite powerful in finding solutions to the input problem very quickly. A user simply has to specify the relevant constraints, and the dedicated propagators take care of the inference tasks.

Note that, while we have so far only discussed CSPs, we can also use CP for

solving constraint *optimisation* problems. We can straightforwardly turn a constraint optimisation problem into a CSP as follows.

Suppose we have an optimisation problem with non-negative objective function $f(\mathbf{X})$ and constraint $c(\mathbf{X})$. The first step is to turn the objective function into a constraint, by setting $f(\mathbf{X}) > 0$. If there exists a solution to the resulting CSP, $\mathbf{X}_{\text{sol}}$, it has value $\theta_{\text{sol}} := f(\mathbf{X}_{\text{sol}})$. We now update the constraint we derived from the objective function to $f(\mathbf{X}) > \theta_{\text{sol}}$ and *continue* the search. Note that, because $\mathbf{X}_{\text{sol}}$ represents the first solution we found, we do not have to restart the search, but can simply continue building the existing search tree, now in pursuit of a new solution $\mathbf{X}_{\text{new}}$ such that $f(\mathbf{X}_{\text{new}}) > \theta_{\text{sol}}$. We continue this process until the CSP becomes infeasible, in which case the last solution that was found represents the solution to the original constraint optimisation problem. This makes CP a declarative, flexible, convenient, general and fast programming paradigm for modelling and solving a wide range of problems.

Unsurprisingly, therefore, the CP community has produced a wide range of tools for modelling and solving CSPs, of which we name a few here. First of all, as described above, CSPs must be modelled before they can be solved. MiniZinc is a free and open source tool, especially designed to model CSPs in a high-level and solver-independent way.[1] All solvers that we name in this section can not only solve CSPs modelled with MiniZinc, but also have an interface that directly connects MiniZinc to the solver.

The powerful open source C++ toolkit Gecode[2] has proven to be a time- and memory-efficient CP solving tool for well over a decade, winning all gold medals in all categories of the MiniZinc Challenge[3] five years in a row. IBM's commercial ILOG CP Optimizer provides state-of-the art support for both real-world, and purely academic constraint optimisation problems.[4] The ILOG-inspired, and recent MiniZinc Challenge gold medallist, Scala library OscaR [132] provides an open source toolkit for constraint solving and constraint optimisation, including functionality for visualising the search tree.[5] Google's OR-Tools provides Python, C++, Java and C# interfaces for users to model CP problems and then solve them by solvers such as CP-SAT.[6] The open-source constraint logic programming system ECLiPSe was particularly designed to be a generic programming tool, especially suitable for rapid prototyping, and provides a Python interface.[7]

[1] Available at `www.minizinc.org`.

[2] Available at `www.gecode.org`.

[3] See `www.minizinc.org/challenge.html`.

[4] Available at `www.ibm.com/analytics/cplex-cp-optimizer`.

[5] Available at `bitbucket.org/oscarlib/oscar`.

[6] Available at `developers.google.com/optimization`.

[7] Available at `eclipseclp.org` and `pyclp.sourceforge.net`.

The discussion above represent just a small selection of the wide range of CP solvers available. Later in this work, in Chapter 5, we will explore how we can use off-the-shelf CP solvers to solve SCPs. Then, in Chapter 6, we present two variants of a propagation algorithm that is specifically designed for a special kind of stochastic constraint.

3.4 Mixed integer programming

As an alternative to CP solvers, we can also employ *mixed integer programming (MIP)* solvers to solve the SCPs studied in this work. We recall basic concepts of MIP. For details we refer the reader to the literature, *e.g.*, Bradley *et al.*'s *Applied Mathematical Programming* [25].

3.4.1 Mixed-integer linear programs

Again, we can model discrete optimisation problems with a set of variables, corresponding domains, a set of constraints and an objective function. Unlike CP solvers, MIP solvers support a limited range of different constraints. *Mixed integer-linear programming (MILP)* solvers – arguably the most widely used type of MIP solvers – support only linear constraints; as a result, they can only be used for solving problems that can be modelled using linear constraints. Note that even MILP is $\mathcal{NP}$-hard [71].

The constraint optimisation problem in Example 3.3.1 also happens to be a MILP, since both the constraints and the objective function are simply linear combinations of variables.

3.4.2 Solving a MILP

Despite the limitation of only being able to deal with *linear* constraints, MILP solvers can be more powerful than CP solvers in solving linear programs, because of their ability to *relax* MILPs. Relaxing a MILP instance means relaxing the integrality constraint of the decision variables with integer domains, meaning that they are allowed to take real values. The resulting linear program has an optimal solution that is guaranteed to be on one of the corner points of the convex hull of all feasible solutions. This is illustrated in the left figure of Figure 3.4. It shows the feasible region, where all constraints are satisfied, and shows the only allowed solutions, which are the integer ones. The optimal continuous solution is on the outer hull of the feasible region, and is indicated in the figure. The figure

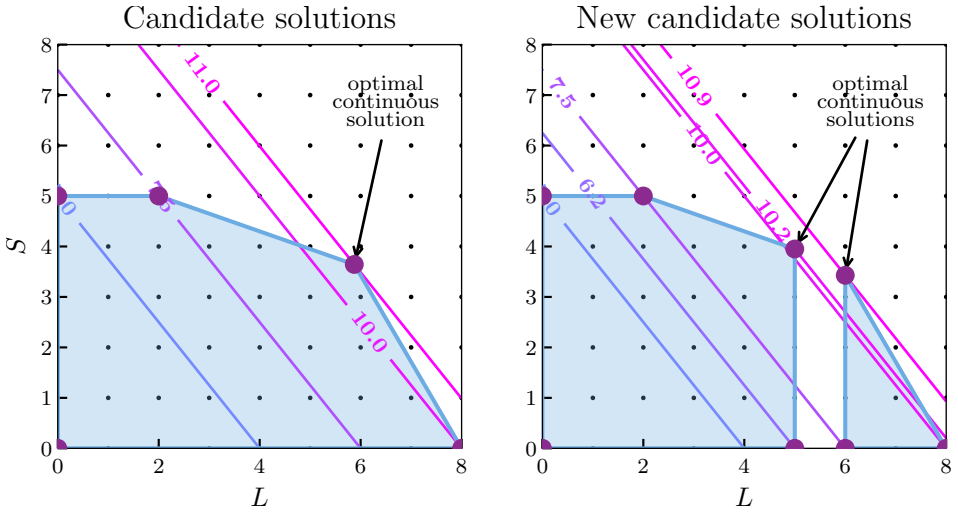


Figure 3.4: *Candidate solutions (solid disks) on the outer hull of the feasible region of the MILP in Example 3.3.1, along with the corresponding values of the optimisation criterion in Equation* 3.1 *(diagonal lines). In the left figure there is one optimal continuous solution. In the right figure, there is one optimal continuous solution for each sub-domain of* L *that is obtained after branching.*

also shows the values of the reliability function (Equation 3.1) for the different corner points of the convex hull.

Since the optimal continuous solution is usually not a valid solution because of the integrality constraints on L and S in Example 3.3.1, a MILP solver must narrow down the space of feasible optimal solutions, until it finds an integer one. Note that simply rounding the continuous solution to an integer one may not be feasible, since that might violate constraints. MILP solvers employ three main techniques for this [25, Chapter 9]: *branch-and-bound*, *cutting planes* and *group-theoretic* approaches. We will give a short intuition for how the first two techniques work, again using the example problem in Example 3.3.1.

The branch-and-bound technique is also employed by CP solvers and simply adds extra constraints to recursively narrow down the search space to an integral one. This process induces a search tree, which allows for pruning similar to what we described in Section 3.3.1.

Example 3.4.1 (Branch-and-bound)**.** *Consider the left figure in Figure 3.4. The MILP solver has used the knowledge that the optimal continuous solution is on a corner points of the convex outer hull of the feasible region, which in this case happens to be at* $(L = {}^{1122}/_{191}, S = {}^{696}/_{191})$*. Using this as a starting point for the search, it might make*

sense to split the problem into two parts by adding two more constraints, either:

$$S \leq 3 \quad \textit{and} \quad S \geq 4, \quad \textit{or} \quad L \leq 5 \quad \textit{and} \quad L \geq 6.$$

Note that neither choice excludes any integer solutions from the feasible region.

In the figure, we have chosen to first branch on the latter set of constraints, similar to what we did in Example 3.3.2 and the left plot of Figure 3.2. This divides the search space up into two sub problems, that can be solved individually. In the left part of Figure 3.4, we have indicated the reliability values of Equation 3.1 *for the corner points of the convex hulls, and the new optimal continuous solutions.*

Note that, if we now continue with the $L \geq 6$ part of the sub-problem, we can immediately refine the optimal continuous solution of $(L = 6, S = {}^{25}/_{7})$ to $(L = 6, S = 3)$. We do this by noting that the optimal continuous solution is not in between two integer solutions, but that its value for S can be rounded down to an integer solution, and obtain $R(6, 3) = 10.5$. Since this value is higher than the optimal continuous solution of the $L \leq 5$ part of the search space ($R\,(5, {}^{79}/_{20}) = 10.2$), we have found the optimal integer solution and do not have to explore that part of the search space.

An alternative way of narrowing down the space of feasible optimal solutions, is the *cutting plane* technique, introduced by Gomory in the 1950s [76], which is used in all modern MILP solvers. After finding an optimal solution to the relaxed MILP, the MILP solver checks if the decision variables in that solution take integer values. If not, it introduces a new linear constraint (cutting plane), separating this solution from the convex hull of feasible solutions to the MILP. Then, it solves the resulting (relaxed) linear program, obtains a new optimal solution, checks it for integrality, and so on. Plenty of research effort has been spent on creating techniques for finding cutting planes that are fast to compute and that reduce the feasible region by as much as possible, without excluding any integer solutions. While a detailed discussion of these efforts is outside the scope of this work, we illustrate the cutting planes technique with the following example.

Example 3.4.2 (Cutting planes)**.** *We illustrate the cutting planes technique in Figure 3.5. The red line in the left figure represents the cut. The part of the feasible region that is above the cut does not contain any integer solutions, and can thus safely be cut off from the feasible region, thus allowing the solver to narrow down its search.*

In the right figure we have indicated the reliability scores for the corner points of the new outer hull of the feasible region. Note that the new optimal continuous solution, $R\,({}^{108}/_{17}, {}^{48}/_{17}) = 10.8$, is smaller than the optimal continuous solution in the right plot of Figure 3.4.

Many MIP solvers grew from extending CP solving techniques such as

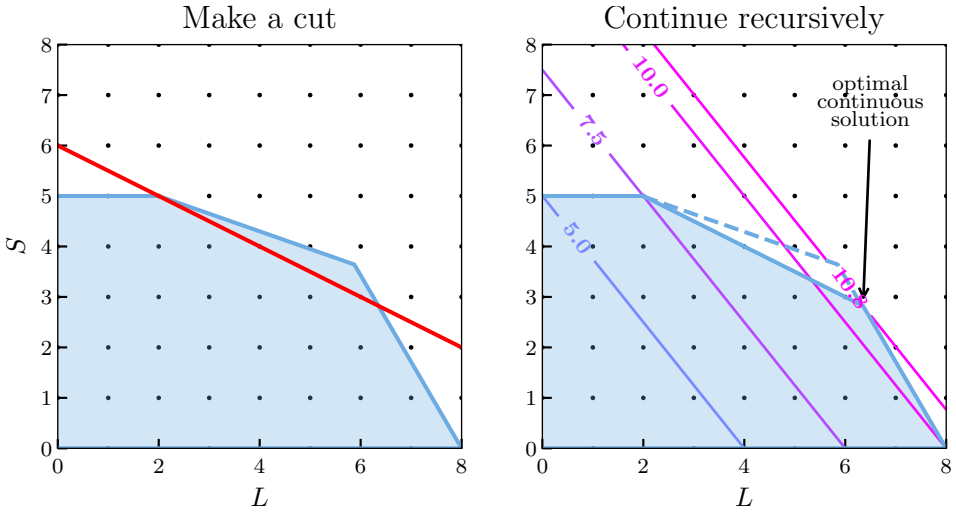


Figure 3.5: *Visualisation of the cutting planes technique. The red line in the left figure cuts off a part of the feasible region, without excluding integer solutions.*

branch-and-bound with MIP solving techniques such as cutting planes. One example of a (commercial) solver (with free academic license) that evolved in this way is IBM's CPLEX Optimizer.[8] After extensive integration of cutting planes techniques into this solver in 1999, it showed a dramatic decrease both in solving time and in optimality gap on MIPLIB examples.[9] Another commercial MIP solver with free academic license, Gurobi, provides a wide range of cutting plane techniques, whose parameters can be tuned either by hand or by Gurobi's automated parameter optimiser.[10] Some systems, such as the non-commercial SCIP Optimization Suite[11], or Google's OR-Tools[12] offer a general framework for modelling MIPs and then providing an interface to the user to have the resulting programs solved by other MIP solvers.

3.4.3 Quadratic programs and linearisation

Later, in Chapter 5 we will demonstrate how to encode SCPs as MIPs. The encodings that we use result in MIP models that are *not* linear, but contain quadratic constraints. While state-of-the-art MIP solvers, such as CPLEX and Gurobi, can

[8] Available at `www.ibm.com/analytics/cplex-optimizer`.
[9] Available at `miplib.zib.de`.
[10] Available at `www.gurobi.com`.
[11] Available at `scipopt.org`.
[12] Available at `developers.google.com/optimization`.

also deal with quadratic constraints under certain conditions, we limit ourselves to MILPs; we do this, because those conditions are currently not guaranteed by all MIP encodings obtained from SDD representations of probability distributions.

Under certain circumstances, it is possible to *linearise* quadratic constraints. We will reflect on how to linearise quadratic constraints that are obtained from OBDD or SDD representations of constraints on probability distributions in Chapter 5. While linearisation typically comes at the cost of increasing the size of the model, it may very well be worth it, because linearised models are potentially very quick to solve by a MILP solver, because of the relaxation, branch-and-bound and cutting planes techniques described above.

3.5 Programming by optimisation

In this work, specifically in Chapters 5 and 6, we introduce *modular* methods for solving SCPs. This is in large part a consequence of the fact that this work is broad in scope. In this and the previous chapter, we have discussed relevant techniques from the fields of propositional logic, probabilistic inference, knowledge compilation, logic programming, CP solving and MIP solving. These different fields have their own states of the art, implemented in different tools and solvers. Thus, if we want to combine the crème de la crème of the technologies brought forth by these fields, a promising attempt at combining them into SCP solvers may be to click them together like LEGO bricks, building SCP solvers in a modular manner.[13]

3.5.1 One size does not really fit anybody

This prompts us to employ yet another programming paradigm for optimisation: *programming by optimisation (PbO)* [80].

Just like LEGO bricks come in different colours, so we can choose which colour to use every time we add one to the thing we are constructing, there are of-

[13] LEGO bricks are things that, when stepped on while barefoot, induce a hellish pain that requires excessive screaming to soothe. Many wheelchair users remain blissfully unaware of the pain inflicted by these specific instruments of torture. Optionally, the small, brightly coloured, interlocking plastic bricks can be used by children and adults alike to construct various objects. Anything constructed can be taken apart again, and the pieces reused to make new things. Much like this dissertation, a LEGO brick presents a choking hazard to anyone unwise enough to stick it in their mouth. Unlike a LEGO brick, however, this dissertation presents a challenge to anyone attempting to stick it far enough up their nose (or anybody else's) to require a hospital visit for its removal from the relevant nose. At the time of writing, these last two statements remain purely speculative, since this dissertation has not been printed yet. That being said, we do not encourage the reader to attempt an empirical verification or falsification of the truth of those statements, even after this dissertation has gone to print.

ten multiple available solutions for solving the same sub task in our SCP solving pipelines. For example, for modelling probability distributions, we could either choose to model them with an OBDD or with an SDD. These *design choices* have no effect on correctness, but can affect performance, especially for computationally challenging problems, such as SCPs.

Note also that one-size-fits-all solutions are rare in this world. There is a reason that LEGO bricks come in different shapes and sizes. We often find that certain approaches work well for solving problems from one domain, but are much less suited to solve problems from another domain. For example: branching heuristics in CP solvers may be domain-specific. However, in practice, only one of these *design choices* is implemented in the final version of an algorithm or software system. The choice is often made based on limited experimentation, with a specific application in mind.

In this work, we try to avoid making that mistake. Rather, we want to exploit the fact that there are often multiple possible ways of achieving (sub)tasks readily available for us to use. As we will describe in Chapter 7, we have therefore constructed various parts of our SCP solving pipeline in such a way that it has access to different methods for solving subtasks and can be tuned for problems from specific application domains.

This approach, implementing different design choices such that the configuration of the resulting solver can be optimised for specific problem types is called PbO [80].

3.5.2 Automated algorithm configuration

Taking a PbO-based approach to software or algorithm design, developers provide the end user with the choice between these options, by exposing them as configurable parameters. A potential downside of this is that the user is left with myriad choices of possible parameter settings (the *configuration* of the algorithm or software system), with even more possible combinations. An end user might not have the specialised expertise to make an optimal choice of these parameter settings, while the algorithm's configuration can have a substantial impact on its performance. Additionally, the optimal configuration may vary for different sets of problem instances.

This also applies to many existing state-of-the-art algorithms that naturally come with many parameters. Using suitable parameter settings is then critical for reaching state-of-the-art performance — especially for $\mathcal{NP}$-hard problems, such as ones studied in this work.

A solution to this problem lies in *automated algorithm configuration (AAC)* [79],

which is the process of automatically finding an optimised configuration of an algorithm's parameters for solving problem instances from a specific problem set, and critically enables PbO-based algorithm design.

After applying AAC to a *target algorithm* A with parameters $q_1, \ldots, q_n$ on a set of problem instances I, we obtain a configuration c^* that is expected to perform well, according to a given performance metric m, on new instances that are similar to those in I. In this work, since we are studying exact optimisation methods and therefore cannot optimise for, *e.g.*, quality of approximation, our performance metric is always running time.

There are two main types of configurators [86]: *model-free* configurators, such as (iterative) F-Race [12, 17] and paramILS [87, 88], and *model-based* configurators, such as SMAC [86] and GGA++ [3].

Model-free configurators are relatively simple. For example, in its most basic form, the well-known configurator F-Race operates by first choosing a set of configurations according to some kind of distribution, and then 'racing' them against each other to see which solves the problem instances the best, according to the performance metric [174]. Once it becomes clear that a configuration is too far behind the others to ever catch up, it is eliminated from the set of candidate configurations. At the end of its configuration run, F-Race returns a set of 'elite' configurations whose performances are statistically equivalent to each other, and statistically better than performances of the configuration outside the set of 'elite' configurations. paramILS [87], on the other hand, uses a process of iterated local search to find optimised parameters.

An advantage of this model-free approach is that it is well-suited for parallelisation. On the other hand, the different racers do not exchange information, thus missing the opportunity to learn about less successful configurations. Consequently, a configurator may lose efficiency by learning the same information more than once, or learning more slowly than it could have with information sharing.

Model-based configurators, on the other hand, sequentially build a model that captures the dependency of the performance of the target algorithm on its configuration. This model is used to predict the performance of configurations on multiple instances and to select promising candidate configurations, which is useful for identifying good configurations more quickly than model-free configurators, because of its ability to learn.

Another important property of configurators is the type of parameters that they support. For example, F-Race focuses on numerical parameters (integer- and real-valued) [174], while paramILS supports numerical and categorical parame-

ters, as well as conditional parameters, whose activations depend on the values assigned to other parameters. An advantage of model-based configurators is that they support many different types of parameters.

This is useful in the context of this work, because in many cases the alternative designs that we have implemented come with very specific sets of parameters. Consequently, the resulting configuration space has many nooks and crannies that are only relevant to explore under specific circumstances. By taking that into account, the search for optimised parameters can often be carried out.

In Chapter 7, we choose SMAC [86] as the configurator for our experiments, because it is one of the best-performing configurators that are freely available.

3.6 Conclusion

In this chapter we described tools for modelling and solving constraint (optimisation) problems, and motivated why we have chosen these specific tools to build on in this work.

Specifically, we described the probabilistic logic programming language DT-ProbLog, which is especially designed to program problems that involve optimal decision making under uncertainty. We described that we need to add functionality for constraints and other types of optimisation than just maximisation in order to use a DT-ProbLog-like language to program the kinds of SCPs studied in this work.

We then proposed to use CP solving as a well-established and powerful search mechanism. We briefly reflected on the two main processes that drive CP solvers: back-tracking search and propagation, and how those relate to the concept of consistency. Finally, we discussed the difference between local and global constraints. Then, we discussed MIP solving as another technique for optimisation. We focused specifically on branch-and-bound techniques and cutting planes techniques, used by MILPs solvers in particular to find integer solutions to mixed-integer linear programs. We also, very briefly, reflected on the existence of quadratic local constraints, and argued why we limit ourselves to linearisable local constraints in this dissertation.

Finally, we motivated why we apply the paradigm of PbO to create a SCP solving pipeline, and gave a brief introduction to this idea, and to AAC, which enables us to take a PbO-based approach. In short, our use of PbO is motivated by the observations that different subtasks in an algorithm can often be completed in different ways (without affecting correctness), and that different design choices that we make there can be better suited for some problems than for others. In

order to fully exploit the potential power of the solver, we therefore implement many of these alternatives, and use AAC to automatically determine optimised configurations for problems from different application domains. We believe that this approach is particularly useful when developing tools for solving hard problems, such as the $\mathcal{NP}$-hard SCPs that we study in this work.

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. Nijssen. '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 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* (AAIMS)[1] 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.

[1] Available at `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.

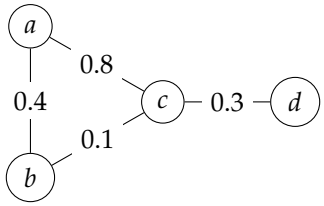
4.2 Modelling SCPs

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.

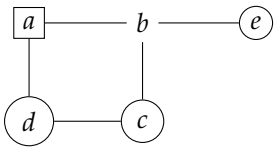
4.2.1 Modelling stochastic constraint (optimisation) problems

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 $D_i \in \{\top, \bot\}$, representing whether person i receives a free sample.*
- *We are interested in the events $\Phi = \{\phi_a, \phi_b, \phi_c, \phi_d\}$, where ϕ_i denotes the event of person i being our customer.*
- *Our objective is to find a strategy σ that maximises the expected utility $\sum_{i \in \{a,b,c,d\}} \rho_i \cdot P(\phi_i|_\sigma)$, 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 β 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 $V_{cons} = \{c, d, e\}$, power producers $V_{prod} = \{a\}$ and power transmitters $V_{trans} = \{b\}$, such that $V_{cons} \cap V_{prod} \cap V_{trans} = \varnothing$ and $V_{cons} \cup V_{prod} \cup V_{trans} = V$ is the set of nodes in the network.*
- *With each power line $(u, v) \in L$ we associate a decision variable $D_{uv} \in \{\top, \bot\}$ that indicates whether or not a power line is reinforced.*
- *We are interested in events $\Phi = \{\phi_i : i \in V_{cons}\}$, where ϕ_i represents that consumer i is still connected to a power producer after a natural disaster.*
- *Our objective is to find a strategy σ that maximises the expected utility $\sum_{i \in \Phi} \rho_i \cdot P(\phi_i|_\sigma)$, 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 ρ_ϕ is the reward associated with ϕ evaluating to *true*, θ is initialised to 0, c_i is the cost of setting decision variable $D_i \in \mathbf{D}$ to *true*, and β 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 θ 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 Φ_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:

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

where μ is a set of truth assignments to all stochastic variables in $\mathbf{T}$, such that μ 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 μ, and $W(T) := w_{\overline{T}}$ if $T = \bot$ in μ.

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 ϕ_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:

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

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 ϕ_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 $p\left(T_{ij}\right)$ with each Boolean variable T_{ij} that this variable is true*. We call T_{ij} a* stochastic variable*. The probability $P\left(\phi_d|_\sigma\right)$ is then defined as the sum of the probabilities of all the (logical) models of this formula, given the strategy. Given strategy $\sigma := \{D_a := \top, D_b := \bot, D_c := \bot, D_d := \bot\}$ (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 $\{T_{ac} = T_{cd} = \top, T_{ab} = T_{bc} = \bot\}$ (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 π_{uv} that takes the following values:*

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

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

$$\phi_d\left(\mathbf{D},\mathbf{T}\right) := (T_{ad} \vee (S_{ad} \wedge D_{ad})) \vee ((T_{ab} \vee (S_{ab} \wedge D_{ab})) \wedge (T_{bc} \vee (S_{bc} \wedge D_{bc})) \wedge (T_{cd} \vee (S_{cd} \wedge D_{cd}))) .$$

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**
$\{T_{uv} := \top, S_{uv} := \top\}$	$P(T_{uv} = \top) \cdot P(S_{uv} = \top)$
$\{T_{uv} := \top, S_{uv} := \bot\}$	$P(T_{uv} = \top) \cdot (1 - P(S_{uv} = \top))$
$\{T_{uv} := \bot, S_{uv} := \top\}$	$(1 - P(T_{uv} = \top)) \cdot P(S_{uv} = \top)$
	$P(T_{uv} = \top) + (1 - P(T_{uv} = \top)) \cdot P(S_{uv} = \top)$

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 ($D_{uv} := \bot$), the probability that ϕ_{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 ϕ_{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 $\sigma = \{D_{ad} = D_{bc} = \top, D_{ab} = D_{bf} = D_{cd} = \bot\}$, one example of a model for $\phi_d|_\sigma$ is: $S_{ad} = T_{ab} = S_{ab} = T_{cd} = S_{be} = \top, T_{ad} = T_{bc} = S_{bc} = S_{cd} = T_{be} = \bot$, 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(alexa).              person(claire).
2. person(behrouz).            person(daniel).

   % Probabilistic relation facts
3. 0.4::dir(alexa,behrouz).        0.8::dir(alexa,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 `=>` 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:

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

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].[2]

[2] 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:

$$\psi(\mathbf{X}) := \mathsf{Q}_1 X_1 \cdots \mathsf{Q}_{|\mathbf{X}|} X_{|\mathbf{X}|} \phi(\mathbf{X}), \tag{4.3}$$

where $\mathsf{Q}_i \in \{\exists, \text{ꓤ}\}$ represent the quantifiers and $X_i \in \mathbf{X}$ the corresponding variables. The quantifier ꓤ indicates that the corresponding variable is 'randomly' quantified, meaning that this variable takes the values $\top$ or $\bot$ with a certain probability, independently of the other variables. While in the general case, the order of the quantifiers in the prefix of ψ 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 σ to the existentially quantified variables $\mathbf{D}$, such that*

$$P(\psi|_\sigma = \top) > \theta? \tag{4.4}$$

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

Thus, the *exists-majority SAT (E-MAJSAT)* problem asks if there *exists* an assignment π such that the probability that ψ is satisfied exceeds a certain threshold value θ.

A well-known special case of E-MAJSAT in which $\mathbf{D} = \varnothing$ (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* ($\mathcal{PP}$) 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 ψ, 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 σ 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 ψ $(\mathbf{D}, \mathbf{T})$ as defined in Definition 4.4.1, which assignment σ of truth values to the existentially quantified (decision) variables in* $\mathbf{D}$ maximises $P\left(\psi|_\sigma = \top\right)$*?*

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

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 Φ^+ and Φ^- be our sets of interest, where events $\phi_{u \to v} \in \Phi^+$ 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 Φ^-. 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:

Variant 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:

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

Variant 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:

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

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

Variant 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:

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

or with the roles of Φ^+ and Φ^- reversed.

Variant 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:

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

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 θ 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:

Variant 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.

Variant 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 θ, 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 θ 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[3]. 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.

[3] Available at `github.com/bdw/GridKit`

In this work we refer to instances from this problem set as **powergrid** instances. We only consider **Variant 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 ϕ_v represents the event that a user v believes or adopts a piece of fake news, and σ_U represents the 'strategy' in which all users in U are considered to be the initial distributors of that news. In words: the *collective* influence of the users in U is at least θ. Second, the selected users all directly influence the same large group of other users: with each set of users U we can associate another set $W \subseteq V$ of users of size at least κ, such that there is an edge (u, w) in the network for each user $u \in U$ and for each user $w \in W$, meaning that u directly tries to influence w. 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 τ per user $v \in V$. Here, τ 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:

Variant 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 κ individual transactions $\tau \in \mathcal{D}$ (making them frequent), where each itemset U has a collective expected influence of at least θ.

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.

5

Decomposition methods for solving SCPs

Solving *stochastic constraint (optimisation) problems (SCPs)* requires efficient probabilistic inference and search. In this chapter, we propose a pipeline for effective SCP solving, which consists of two stages. In the first stage, we compile the underlying probability distribution of the input SCP into *decision diagrams (DDs)* (*ordered binary decision diagrams (OBDDs)* or *sentential decision diagrams (SDDs)*, specifically). These diagrams are converted into *arithmetic circuits (ACs)*, which are decomposed into models that are solved using *constraint programming (CP)* or *mixed integer programming (MIP)* solvers in the second stage. We show that, to yield linear constraints in those models, DDs need to be compiled in a specific form. We introduce a new method for compiling small SDDs in this form (OBDDs are naturally in this form). We evaluate the effectiveness of several variations of this pipeline on test cases in viral marketing and bioinformatics, and find that MIP-based methods outperform CP-based methods on all our test instances. This

chapter is based on the following publication:

A.L.D. Latour, B. Babaki, A. Dries, A. Kimmig, G. Van den Broeck, and S. Nijssen. 'Combining Stochastic Constraint Optimization and Probabilistic Programming — From Knowledge Compilation to Constraint Solving'. In: *CP*, Springer. pp. 495–511. 2017.

5.1 Introduction

Recall from Section 4.1 that this work is partially motivated by the observations that most SCP-related literature focuses on scheduling and planning problems, whereas SCPs formulated on probabilistic networks remain much less studied, and that there exists no generic language for programming stochastic constraint *optimisation* problems. We addressed both of these limitations in Chapter 4.

The aim of this chapter is to advance the state of the art in SCP solving on a third dimension, observing:

- There is no automatic pipeline for solving SCPs written in the new SC-ProbLog language.

We address this limitation by building a pipeline on technology that is taken both from the probabilistic reasoning literature and CP literature. For the probabilistic reasoning component, we leverage knowledge compilation technology, demonstrating how both OBDDs [26] and SDDs [46] can be used in this context, namely: by putting hard constraints on DD representations of probability distributions. In this chapter, our focus is primarily on SDDs, since they are known to lead to smaller representations of distributions than OBDDs [24] (see also Section 2.4).

We remark that, in this work, we study the use of compiling propositional formulae to OBDDs [26] and SDDs [46] to facilitate tractable *weighted model counting (WMC)*, in the context of SCP solving. Note that, in the *constraint programming (CP)* literature, both OBDDs and SDDs are often employed as compact representations for the satisfying assignments of a constraint [70, 77]. Here, we use these diagrams differently.

Specifically, we propose to convert the DDs into ACs, as described in Section 2.5, which we can use to compute conditional probabilities in a time that is linear in the size of the underlying DD. Part of the novelty of our approach lies in then formulating a hard constraint on the AC and *decomposing* that constraint (and thus the AC) into a set of local constraints.

We do this so we can translate a global constraint for which no propagation algorithm exists, into a set of constraints for which propagation algorithms have

been developed and optimised for decades, to see how much we can benefit from these in the context of solving SCPs. Finally, we solve these constraints using CP and MIP technology.

Note that this *modular* approach to building an SCP solver has the advantage of allowing us to use the best building blocks for the pipeline that are on offer, instead of having to integrate different elements into one single solver. By using knowledge compilation as part of this pipeline, part of the model counting problem can be solved in a preprocessing phase, by the knowledge compiler. The resulting DD can then be passed on to the next phase, where it is used to enable repeated querying, which is useful in finding an optimal strategy, or finding a strategy that respects a certain constraint.

Another key technical contribution of this work is that we show that SDDs need to satisfy strict criteria in order for them to yield linear representations of probabilistic constraints. We introduce a new algorithm for minimising SDDs within this normal form. This allows us to reduce the size of the resulting ACs, while keeping the resulting constraint optimisation model *linear*, rather than *quadratic*, and thus easier to solve for MIP solvers.

The remainder of this chapter is organised as follows. In Section 5.2, we demonstrate how stochastic constraints on OBDD- and SDD-representations of probability distributions can be decomposed for solving with CP or MIP technology. In that section we also introduce the aforementioned normal form and our new SDD minimisation algorithm, as well as a solving pipeline based on DD decomposition. We present an experimental evaluation in Section 5.3, and conclude this chapter in Section 5.4.

5.2 Decomposing and solving stochastic constraints

In this section we describe the pipeline that we propose in this chapter in more detail.

Recall from Section 2.5 that, once we have compiled a probability distribution into a DD, we can transform that DD into an ACs to compute conditional probabilities. Specifically, we can use ACs to compute the success probabilities of residual probability-weighted propositional formulae $\phi|_\sigma$. A key observation is that the constraint in Equation 1.1 essentially is a constraint on the strategy σ. Recall that we encode probability distributions using DDs. Taking OBDDs as an example, we encode a strategy by adding weights to the outgoing arcs of the decision nodes. Thus, we can see Equation 1.1 as a constraint on the outcome of the AC that encodes $P(\phi|_\sigma)$, given the weights on the outgoing arcs of the probabilis-

tic nodes of the underlying OBDD. We can also see Equation 1.1 as a constraint the weights we can put on the outgoing arcs of the decision nodes in that OBDD, to reflect a strategy. Because σ specifies value assignments to *Boolean* variables, we can cast solving Equation 1.1 as a *discrete* constraint satisfaction problem.

In this section, we demonstrate how we can decompose a constraint on an AC representation of a probability distribution into a (linear) program that can be solved by a CP or MIP solver. We first show how this can be done for ACs obtained from OBDDs, and then describe how we can do the same for ACs obtained from SDD representations. For the sake of brevity, we will often refer to "decomposing a constraint on an AC derived from a DD representation of a probability distribution" as "decomposing a DD".

We close this section with a proposal for a SCP solving pipeline that uses these DD decompositions.

5.2.1 Decomposing a stochastic constraint on an OBDD

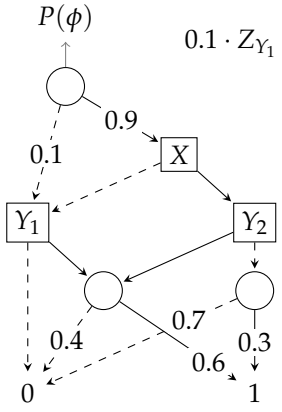
Recall from Section 2.5.1 that we represent a specific strategy by labelling the outgoing arcs of OBDD nodes labelled with decision variables with the values 0 and 1. Our aim is to solve Equation 1.1, which we interpret as a constraint on the values we can use to label those arcs. Therefore, we can interpret Equation 1.1 as a constraint on the AC induced by the OBDD that describes the probability distribution of an SCP.

Decomposition of a global constraint on an OBDD

We now show how we can decompose this global constraint on the OBDD into a multitude of smaller, local constraints.

Example 5.2.1 (Decomposition of a constraint on an OBDD representation of a probability distribution)**.** *Figure 5.1 shows an example of an OBDD representation of a formula ϕ. We impose the constraint $P(\phi|_\sigma) \geq 0.4$. Figure 5.1 also shows an example of a decomposition of $P(\phi|_\sigma) \geq 0.4$ on the whole OBDD, adding auxiliary variables Z_{Y_1}, Z_{Y_2} and Z_X, whose domains include real numbers. This decomposition represents a CP or MIP model of the constraint $P(\phi_\sigma) \geq 0.4$.*

The next step to solving the global constraint, is to simply feed this set of smaller, local constraints to a CP or MIP solver. However, the decomposition in Figure 5.1 contains quadratic constraints, as illustrated in the following example, which are hard to solve for MIP solvers and CP solvers.



$$
\begin{aligned}
P(\phi \mid \sigma) &\geq 0.4 \\
0.1 \cdot Z_{Y_1} + 0.9 \cdot Z_X &\geq 0.4 \\
Z_X &:= (1 - X) \cdot Z_{Y_1} + X \cdot Z_{Y_2} \\
Z_{Y_1} &:= 0.6 \cdot Y \\
Z_{Y_2} &:= 0.6 \cdot Y + 0.3 \cdot (1 - Y) \\
X, Y &\in \{0, 1\} \\
0 \leq P(\phi|_\sigma) &\leq 0.6 \\
0 \leq Z_X &\leq 0.6 \\
0 \leq Z_{Y_1} &\leq 0.6 \\
0.3 \leq Z_{Y_2} &\leq 0.6
\end{aligned}
$$

Figure 5.1: *A small OBDD (left) with three stochastic variables (circular nodes) and two decision variables X and Y (rectangular nodes). The two nodes corresponding to decision variable Y are indexed for clarity. The decomposition on the right is constructed using Equation* 2.11 *(page 39).*

Example 5.2.2 (Quadratic constraint)**.** *Figure 5.2 shows a graphical representation of the constraint in Example 5.2.1. It particularly shows a relaxation (recall the discussion of MIPs in Section 3.4) of the constraint. The coloured lines, labelled with values* 0.0 *to* 0.7 *represent contours on which the combination of (relaxed) X and Y values yield those particular values for $0.1 \cdot Z_{Y_1} + 0.9 \cdot Z_X$, and thus for $P(\phi \mid \sigma)$.*

We have added the extra constraint of $\sum_{D \in \{X,Y\}} D \leq 1$ to the figure, resulting in a SCP that corresponds to the constraint satisfaction problem (CSP) *in Example 3.3.2, and shaded the feasible region. It is easy to read from the figure that the only solution is $(X = 0, Y = 1)$, with value* 0.6*, as in Example 3.3.2*

Linearising quadratic constraints

Note that, while we can easily read the only solution to the SCP described above directly from Figure 5.2, the curved lines, due to the quadratic constraint, make it harder for the MIP solvers in particular to apply techniques such as branch-and-bound and cutting planes to narrow down the search towards integer solutions.

We therefore linearise this decomposition, for easier solving. A constraint of the form $A = B \cdot C$ can be linearised in the following cases:

1. At least one of the two variables in $\{B, C\}$ is a constant.
2. At least one of the two variables in $\{B, C\}$ is a Boolean variable.

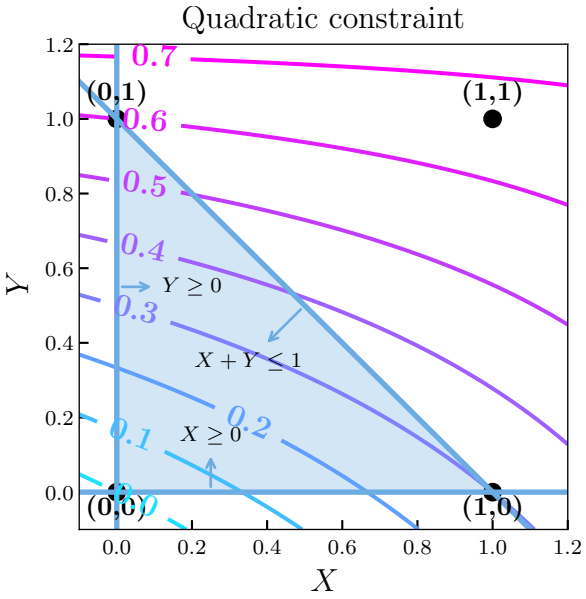


Figure 5.2: *Visualisation of the quadratic constraint* $0.1 \cdot Z_{Y_1} + 0.9 \cdot Z_X \geq 0.4$ *from Figure 5.1, as function of the values of* X *and* Y*. The added constraint* $X + Y \leq 1$ *makes this SCP correspond to the one described in Example 3.3.2.*

We obtain constraints that encode the OBDD by simply applying Equation 2.11, which we repeat here, for convenience:

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

to all (internal) nodes of the OBDD. An OBDD node r can be labelled either with a decision variable, or with a stochastic variable. If r is labelled with a decision variable, we can apply the big-M method [127] (with $M \leq 1$, because all real values are probabilities) to linearise the constraint expressed by Equation 2.11. If r is labelled with a stochastic variable, the arguments on either side of the '$+$'-sign each consist of a real number ($w(r)$ or $1 - w(r)$), multiplied by an expression ($v(r^{+})$ or $v(r^{-})$). If an expression is linear, multiplication with a real number preserves linearity. Summing two linear expressions yields another linear expression, making the constraint obtained by applying Equation 2.11 linear if r is labelled with a stochastic variable. Consequently, one of the above two cases always holds for all multiplications in the decomposition of a stochastic constraint on an OBDD representation of a probability distribution.

We illustrate this with the following example:

Example 5.2.3 (Linearising a quadratic stochastic constraint)**.** *The quadratic con-*

straint $Z_X := (1 - X) \cdot Z_{Y_1} + X \cdot Z_{Y_2}$ in Figure 5.1 (where X is a Boolean, as shown in Figure 5.1) can be linearised as follows:

$$\begin{array}{lll} Z_X := Z_{X_\top} + Z_{X_\bot} & Z_{X_\top} \leq X & Z_{X_\bot} \leq 1 - X \\ 0 \leq Z_{X_\top} \leq 1 & Z_{X_\top} \leq Z_{Y_2} + (1 - X) & Z_{X_\bot} \leq Z_{Y_1} + X \\ 0 \leq Z_{X_\bot} \leq 1 & Z_{X_\top} \geq Z_{Y_2} - (1 - X) & Z_{X_\bot} \geq Z_{Y_1} - X \end{array} \tag{5.1}$$

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

5.2.2 Decomposing a stochastic constraint on an SDD

The decomposition of a constraint on a probability distribution represented by an SDD is very similar to that of a constraint on a probability distribution that is represented by an OBDD. One important difference is that not every SDD can be decomposed into a *linear* program. In the general case, SDDs yield *quadratic* programs, which are typically harder or impossible to solve for MIP solvers than linear programs. As we expect these constraints to be nonpositive semidefinite in the general case, we expect that we cannot apply *quadratically constrained quadratic program (QCQP)* solvers, either. We delegate the finding of a proof for this hunch to future work.

We first show why SDDs cannot be decomposed into linear models in the general case . Then, we identify a specific property of SDDs that does allow SDDs with that property to be decomposed into linear programs. Finally, we show how we can create an SDD minimisation algorithm for SDDs with this property.

From SDD to CP or MIP model

To see why we cannot linearise any decomposed constraint on an SDDs, recall the method for creating a linear model out of a constraint on an OBDD representation of a probability distribution as described in Section 5.2.1, and observe the difference between Equation 2.11 and Equation 2.12, which we repeat here for convenience:

$$\text{Equation 2.11: } v(r) := w(r) \cdot v\left(r^+\right) + (1 - w(r)) \cdot v\left(r^-\right), \quad \text{(OBDD node), and}$$

$$\text{Equation 2.12: } v(r) := v\left(p^\ell\right) \cdot v\left(s^\ell\right) + v\left(p^r\right) \cdot v\left(s^r\right) \quad \text{(SDD node).}$$

While constraints generated with Equation 2.11 can always be linearised (because $w(r)$ and $(1 - w(r))$ are either constants or Booleans), this is not the case for constraints generated with Equation 2.12. In that equation, the two arguments on either side of the '$+$'-sign are each a product of two expressions, e.g., $v\left(p^\ell\right)$ and

$v\left(s^{\ell}\right)$ on the left-hand side of the '$+$'. Even if those two expressions are themselves linear, their product can only be linearised efficiently if at least one of the expressions is constructed using only constants (i.e., weights corresponding to stochastic variables, in which case the product is trivially linear), or using only decision variables (in which case the product can be linearised using the big-M method).

We now identify a class of SDDs whose decompositions *can* be linearised.

Single-mixed path vtrees

Recall the description of vtrees in Section 2.4.3, and recall that they generalise the concept of variable order. Recall also that, while SDDs do not require a total order, we can derive a total order $\mathcal{O}$ from a vtree by traversing it in a left to right manner, noting the variables that label the leaves in the order in which they are encountered in this traversal. Different vtrees can thus correspond to the same total order $\mathcal{O}$.

We now show that it suffices to constrain the vtrees to ensure that the decomposition of the SDDs that respect them can be linearised. Recall that for each SDD decomposition node, the respected vtree determines the scopes of sub formulae represented by the prime and the sub. We observe the following: if all left-hand (right-hand) descendants of an internal vtree node n are stochastic variables, then for each SDD decomposition node (p, s) whose parent respects n, it holds that all variables occurring in n's prime p (sub s) are stochastic as well. A similar property holds for decision variables.

Recall from Section 5.2.2 that if the sub or the prime of a decomposition node represents a constant or a Boolean variable, this means that the constraints associate with those decomposition nodes can be linearised. Note that, if $sc(p) \subseteq \mathbf{T}$, the only variables in the scope of prime p are stochastic ones. Since stochastic variables can be considered as constants for the MIP model, we can precompute the corresponding value for the prime, effectively eliminating the MIP model variable associated with that prime. Similarly, if $sc(p) \subseteq \mathbf{D}$, the sub formula represented by a prime p consists only of decision variables, which can only take Boolean values in the decomposition. Since we can linearise all operations on Boolean variables [127], any prime containing only decision variables can be expressed by a Boolean variable with linear relations to other variables. Thus, in each of these two cases, the expression represented by the prime can be linearised and hence the product represented by the SDD decomposition node as well. The same holds for subs.

This leads us to define the concept of *mixed* and *pure* nodes in a vtree. A *pure*

node is an internal node whose leaf descendants all are variables of the same type (either stochastic or decision), while a *mixed* node is an internal node that has leaf descendants of both types. We state that an SDD can be linearised into a MIP model if the vtree that it respects has the *single mixed path (SMP)* property.

Definition 5.2.1. *Given a vtree on variables of two distinct classes (e.g. decision and stochastic). This vtree has the* single mixed path (SMP) *property (and is called an* SMP vtree*) if, for each of its internal nodes n, the following holds: either both children of n are pure nodes, or one child of n is pure and the other child is mixed. As a consequence, if an SMP vtree has mixed nodes, all mixed nodes occur on the same path from the root of the vtree to the lowest mixed node.*

SMP-preserving SDD minimisation

Recall that SDDs that respect right-linear vtrees are equivalent to OBDDs. One can easily verify that a right-linear vtree has the SMP property: if it has an single mixed path, it is on the right spine of the vtree. From this follows that OBDDs can be linearised. However: right-linear vtrees generally do not yield the smallest SDDs. Since the size of the SDD determines the size of the resulting MIP model, and thus likely the solving time, small SDDs are preferable as input for the MIP model builder.

Choi and Darwiche have proposed a local search algorithm for SDD minimisation [36]. This algorithm considers three operations on the vtree: *right-rotate*, *left-rotate* (each well-known operations on binary trees) and *swap*. When a swap operation is applied to an internal node, the sub vtrees rooted at its children are swapped. Given a (sub) vtree, the greedy local search algorithm of Choi and Darwiche loops through its neighbourhood of different vtrees by applying consecutive rotate and swap operations, trying to find a vtree that yields a smaller SDD. Recall from Section 2.4.1 that we expect SDD minimisation to be $\mathcal{NP}$-hard.

Generally, this minimisation produces vtrees that do *not* have the SMP property, even if the initial vtree did, because rotation may remove this property.

A desirable property of Choi and Darwiche's algorithm is the following: the three local moves considered are sufficient to turn *any* vtree on a certain set of variables into *any* other vtree on the same set of variables. Consequently, the local moves in principle allow complete traversal of the search space of vtrees.

Here, we propose a simple modification of Choi and Darwiche's algorithm: we use the same local moves as their algorithm does, but any move that leads to a vtree that violates the SMP property is immediately rejected.

While this modification is conceptually easy, a relevant fundamental question

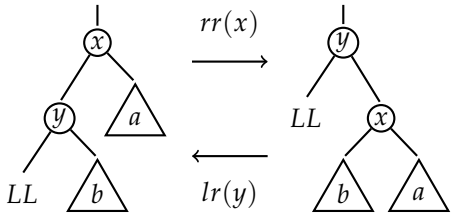


Figure 5.3: *Rotate operations on an SMP vtree. Node LL is the lowest variable in the variable ordering induced by these vtrees. Nodes x and y are internal; a and b are sub vtrees.*

is whether under this modification it is still possible to traverse the space of SMP vtrees on a fixed set of variables completely. We show that this is indeed the case.

In the following we refer to the leaf node that represents the variable that is lowest in the order associated with a vtree as *LL* (lowest leaf).

Lemma 5.2.1. *Let y be the parent and x the grandparent of the LL in an SMP vtree. Right rotate on x maintains the SMP property for the vtree rooted at y.*

Proof. Consider the left SMP vtree in Figure 5.3. Given that this vtree satisfies the SMP property by assumption, sub vtrees a and b cannot both be mixed, but one of them can be. Now consider the following cases:

Both a and b are pure and of the same class as LL: Lemma 5.2.1 holds trivially.

Both a and b are pure, not each of the same class as LL: Any class assignment to a and b will preserve the SMP property.

Node b is pure, node a is mixed: Since b is of the same class as LL (by assumption), node y is pure and node x is mixed. After applying right-rotate on node y, both y and x are mixed, and the SMP property is preserved.

Node b is mixed, node a is pure: Node a can belong to any class, since both node y and node x are mixed before as well as after applying right-rotate to y, preserving the SMP property under rotation.

These cases cover all possibilities for the classes of a and b. □

Note that the SMP vtree described above may be a sub vtree of a larger vtree. The fact that the right-rotate operation does not change the nature (mix or pure) of the root of this sub vtree, leads to the following corollary:

Corollary 5.2.1. *A right-rotate operation on the grandparent of the LL node does not change the SMP status of the full vtree.*

Lemma 5.2.2. *Given an SMP vtree, from which we derive total variable order $\mathcal{O}$ and a particular node LL. We can always obtain an SMP vtree from which we can derive the same total order $\mathcal{O}$, in which the LL is the left child of the root, through a series of right-rotate operations, without ever in the process transforming it into a vtree that violates the SMP property.*

Proof. A right-rotate operation on an internal vtree node decreases its left child's distance to the root of the vtree by one. Repeated applications of right-rotate on LL's grandparent ultimately makes LL's parent the vtree's root. By Lemma 5.2.1 and Corollary 5.2.1, the SMP status of the vtree never changes in this process. □

Lemma 5.2.3. *Given an SMP vtree on order $\mathcal{O}$, we can always obtain a right-linear vtree on the same order, through a series of right-rotate operations, without ever in the process transforming it into a vtree that violates the SMP property.*

Proof. By Lemma 5.2.2 we can turn any SMP vtree in one for which the LL is the left child of the root. This vtree can be made right-linear by recursively applying this method to the root's right child. □

Lemma 5.2.4. *A right-linear SMP vtree with variable order $\mathcal{O}$ can be transformed in any SMP vtree on the same variable order by a series of left-rotate operations without ever in the process transforming into a vtree without the SMP property.*

Proof. Since left-rotate is the dual operation of right-rotate, a sequence of right-rotate moves transforming any vtree to a right-linear one through right-rotate operations, can simply be reversed through left-rotate operations to turn a right-linear vtree in any other (on the same variable order). □

Note that rotate operations preserve the derived total order of the vtree, traversing the vtree from left to right, we still encounter the leaves in the same order. The only thing that changes, is the vtree's shape. However, the space of possible vtrees on a fixed set of variables is larger, since different total variable orders exist. The total order of variables is only changed by the application of swap operations.

Lemma 5.2.5. *Any right-linear vtree on variable order $\mathcal{O}$ can be transformed into a right-linear vtree on any other total variable order $\mathcal{O}'$ through a series of rotate and swap operations without ever in the process transforming into a vtree that violates the SMP property.*

Proof. Observe that any right-linear vtree satisfies the SMP property. Observe that if we can reverse the mutual total order of two adjacent variables (e.g. $A \prec B \prec$

$C \prec D$ becomes $A \prec C \prec B \prec D$), we can create any total variable order by repeatedly reversing the orders of adjacent variables. This reversal in the total order is simple to achieve. Suppose that node b in the right vtree of Figure 5.3 is a single variable, as is LL. We can make LL and b swap places by applying a left-rotate on y, resulting in the left vtree of Figure 5.3, and then applying a swap operation on y, followed by a right-rotate operation on x. □

Theorem 5.2.1. *Any SMP vtree can be transformed into any other SMP vtree on the same variable through a series of rotation and swap moves, without ever in the process transforming into a vtree that does not have the SMP property.*

We conclude that an SMP-preserving minimisation algorithm that applies only swap and rotate operations can in principle convert any SMP vtree into any other SMP vtree on the same variables. Note that, in principle, we could use an unrestricted minimisation algorithm. However, the search space of possible SMP vtrees on a given total order $\mathcal{O}$ is only a small part of the search space of all vtrees on $\mathcal{O}$. Therefore, it might not be easy or quick to transform a minimised SDD that violates the SMP property back into one that respects it, and we choose to adapt the minimisation algorithm in such a way that the vtree never loses the SMP property. Thanks to the above theorem, it is possible to traverse the entire search space of SMP vtree for a given total order, even though the path from one vtree to another might be very long.

Using the insights above, we implemented a greedy SMP-preserving minimisation algorithm as follows, building on the minimisation algorithm implemented in UCLA's `sdd 1.1.1` library[1]. First, we compile an SDD without any minimisation. Since in the default settings, the resulting SDD respect a right-linear vtree, by Definition 5.2.1 this SDD has the SMP property. We then minimise this SDD by iteratively selecting an internal vtree node and exploring the neighbourhood of possible vtrees by performing SMP-preserving left-rotate, right-rotate and swap operations on it. We greedily choose that operation that reduces the size of the SDD the most. We repeat this process until the SDD size converges.

5.2.3 A decomposition-based SCP solving pipeline

In Sections 4.2 and 4.3 we showed how to model and program SCPs, and in Section 1.3 we identified two components to SCP solving complexity: probabilistic inference and search space traversal. Then, in Section 2.5, we showed how we can use the compact truth table representations that decision diagrams offer to

[1] Available at `reasoning.cs.ucla.edu/sdd`

tractably perform online probabilistic inference. We briefly argued for the use of CP and MIP technology for efficient search space traversal in Sections 3.3 and 3.4.

In this chapter we described how we can combine these ingredients to create constraint programs and mixed-integer programs that encode SCPs that can be solved efficiently.

In order to solve SCPs, we propose the following decomposition-based pipeline, see also Figure 5.4:

Step 1: Model the problem using a probabilistic network and (stochastic) constraint(s) or optimisation criterion.

Step 2: Program the problem using SC-ProbLog.

Step 3: Model the program for the queries present in the optimisation criterion of the SCP into a set of propositional formulae Φ.

Step 4: Compile a multi-rooted OBDD or SDD Δ, such that each root encodes the conditional success probability $P(\phi \mid \sigma)$ of one of the queries $\phi \in \Phi$, using possibly SMP-preserving minimisation algorithms for the SDD compilation to guarantee linearised models.

Step 5: Convert Δ into a multi-rooted AC (see Section 2.5), and then decompose this AC into a set of constraints, using the big-M method to linearise constraints when appropriate.

Step 6: Add the (stochastic) constraints to the set of constraints.

Step 7: Add the (stochastic) optimisation criterion to the resulting CP or MIP model.

Step 8: Use an off-the-shelf CP or MIP solver to find the optimal solution.

Note that, while we include OBDDs in the pipeline for reasons of generality, in this chapter, the focus in primarily on SDDs. Recall from Section 2.5 that SDDs that respect a right-linear vtree are actually OBDDs and that SDDs can be more succinct than OBDDs, once minimised. This motivates our choice to focus on SDDs in this chapter.

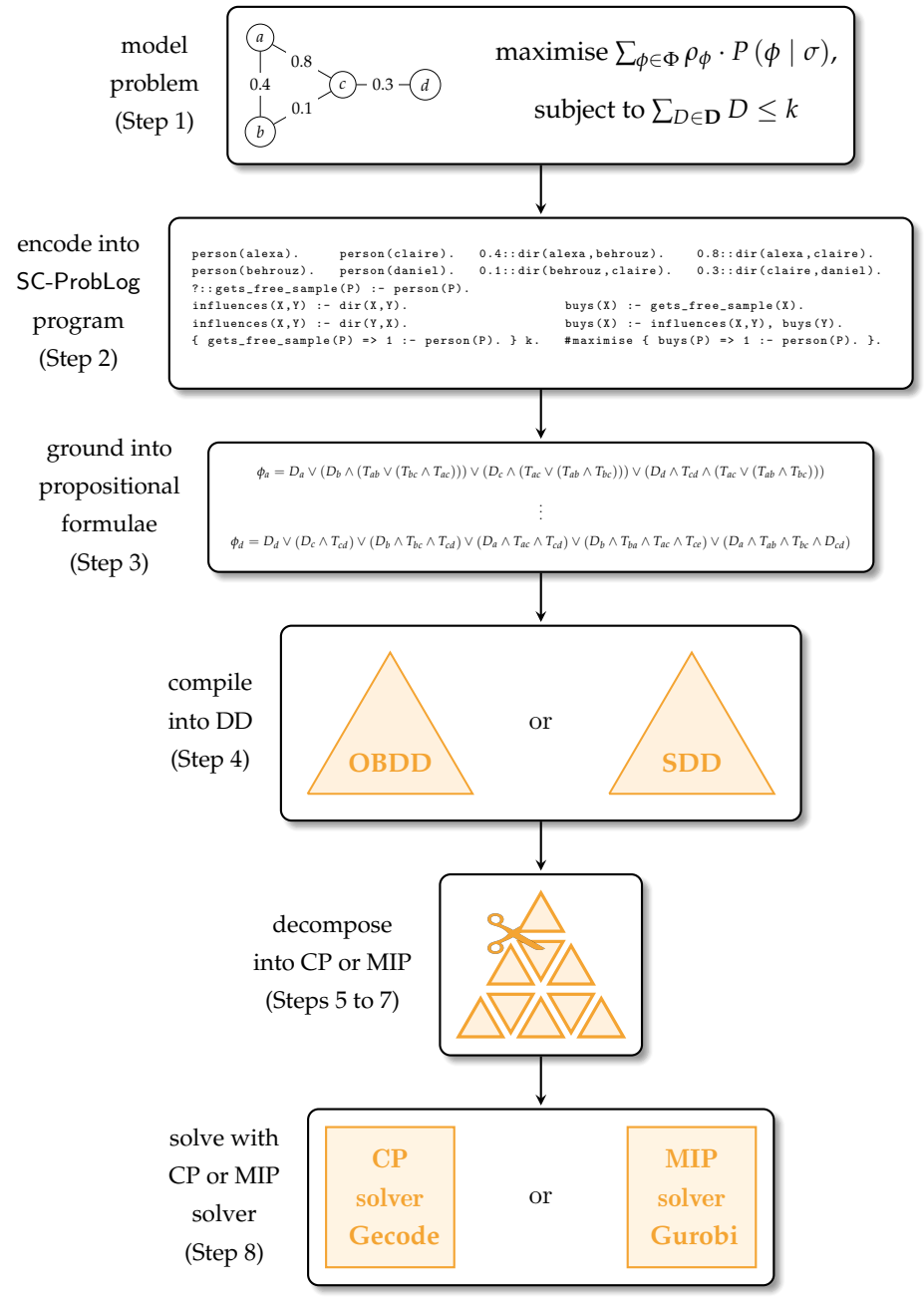


Figure 5.4: *Overview of the decomposition-based SCP solving pipeline we propose in this chapter.*

5.3 Experimental evaluation

We state some questions about the approach described in the previous section. Then we describe the experiments that we performed to answer these questions.

5.3.1 Research questions

Recall that the size of a MIP or CP model is linear in the size of the SDD representation of the probability distribution on which we impose a stochastic constraint. We expect smaller models to be faster to solve. However: minimising an SDD takes time. Furthermore, when quadratic constraints are allowed, we expect to obtain smaller SDDs; however, solving quadratic problems using CP may take longer than solving MIPs. We pose the following questions:

Q2 How do SDD sizes depend on the choice of minimisation algorithm?

Q3 How do the calculation times for the full toolchain compare for CP and MIP solvers, with and without appropriate minimisation?

Q4 How do the computation times for different phases of the algorithm compare to each other?

To answer these questions, and to demonstrate that SC-ProbLog programs can be solved in practice, we apply our algorithms to different SCPs. Of course, the constraints determine problem hardness, which begs the question:

Q1 Which threshold settings are useful for an evaluation of the solving times?

5.3.2 Experimental setup

We briefly describe our experimental setup and some details on the problem instances we used for our experiments.

Software and hardware

We implemented the stochastic constraint component of SC-ProbLog[2] in Python 3.4, building on the existing ProbLog 2.1 [59] implementation. ProbLog 2.1 uses UCLA's sdd 1.1.1 library [36], which is implemented in C, for SDD compilation.[3] We built on this code to implement our SMP-preserving SDD-minimisation algorithm. To convert constraints on SDDs representations of probability distributions into MIP models, we used Gurobi 6.52, which provides a convenient modelling

[2] Available at `github.com/ML-KULeuven/problog/tree/sc-problog`.

[3] Available at `reasoning.cs.ucla.edu/sdd`.

Table 5.1: *Some characteristics of the problem instances for the experiments in this section. We give the extracted community and variant of the problem we formulate on that network (see Section 4.5). We also provide the size of the set(s) of interest $|\Phi|$ and the number of decision variables $|\mathbf{D}|$ in the SC-ProbLog encoding of each problem. For each problem, we give the constraint threshold k or θ and objective value v_{obj} ('n/a' denotes a problem that has no solution for that threshold).*

instance	problem type	$\lvert\Phi\rvert$	$\lvert\mathbf{D}\rvert$	threshold	v_{obj}
spine16, variant 1	*sparsification*	23	33	$k = 15$	14.4
spine16, variant 2	*sparsification*	23	36	$\theta = 6.9$	8
spine27, variant 1	*sparsification*	13	76	$k = 25$	10.2
spine27, variant 2	*sparsification*	13	76	$\theta = 6.5$	8
spine27, variant 3	*sparsification*	26	86	$\theta = 1.3$	9.5
spine27, variant 4	*sparsification*	13	71	$\theta = 6.5$	52
hepth47, variant 1	*spread of influence*	20	20	$k = 10$	3.2
hepth47, variant 2	*spread of influence*	20	20	$\theta = 2$	6
hepth5, variant 1	*spread of influence*	10	33	$k = 20$	2.8
hepth5, variant 2	*spread of influence*	10	33	$\theta = 5$	n/a

interface through gurobipy.[4] We built our CP models using Gecode 5.0.0. We used Gurobi 6.52 as MIP solver and Gecode 5.0.0 as CP solver.[5]

We ran our experiments on a machine that we call JABBA. It has an Intel Xeon E5-2630 processor and 512GB RAM, running under Red Hat 4.8.3-9. For each individual computational step of the pipeline (Steps 3, 4 and 8) we used a timeout on our experiments of 3 600 s (1 hour).

Problem instances

For our experiments we use instances obtained from the **spine** [133] and **hepth** [131] datasets described in Section 4.5. We selected specific communities that we refer to as **spine16**, **spine27**, **hepth47** and **hepth5** in our results, and summarise some of the characteristics of the resulting problem instances in Table 5.1.

5.3.3 Results

To answer **Q1**, Figure 5.5 shows solving times for the **hepth47-v1** problem, for different thresholds. As expected, we find that thresholds that are not very strict

[4] Available at `www.gurobi.com`.

[5] Available `www.gecode.org`.

Table 5.2: *Performance in seconds of the different methods on the hardest instances (see Table 5.1) for the full pipeline. We show the solving times for SDDs obtained with compilation with no minimisation (t_{none}), with SMP minimisation (t_{smp}) and with default minimisation ($t_{default}$) for Gurobi and Gecode. We indicate a timeout with 't/o'.*

	Gurobi		Gecode	
instance	t_{none}	t_{smp}	t_{none}	$t_{default}$
spine16, variant 1	3.9	3.4	1 389.5	591.4
spine16, variant 2	4.1	3.9	70.9	31.4
spine27, variant 1	5.9	5.6	t/o	t/o
spine27, variant 2	4.7	5.7	t/o	1 878.2
spine27, variant 3	443.2	471.3	t/o	t/o
spine27, variant 4	23.3	21.9	222.9	8.6
hepth47, variant 1	545.8	412.7	t/o	130.9
hepth47, variant 2	188.6	163.8	2 859.9	6.9
hepth5, variant 1	2 076.8	1 185.7	t/o	t/o
hepth5, variant 2	364.6	346.4	t/o	t/o

or loose, require the longest solving times. We performed similar experiments for the other problem settings to systematically identify the threshold for which each problem was the hardest, which we then chose as test cases for the SCP solving method comparison.

To answer **Q2**, Figure 5.6 shows a comparison of the size reductions obtained by the SMP-minimisation algorithm and the default minimisation algorithm provided by the sdd library. We find that the SMP minimisation algorithm typically halves the size of the initial SDD. The default minimisation typically reduces the size of the SDD by one or two orders of magnitude.

To answer **Q3**, we summarise the performance of the four methods on our test cases in Table 5.2. For the **hepth5** problem we selected the ten highest-degree nodes for the queries, since the program could not be grounded within one hour if we selected all 33 nodes in the problem for querying. This reduced the grounding time to about 120 seconds. For the other test cases we have selected all queries in the problem, with grounding times in the range of 1–5 seconds.

We observe that without any minimisation of the SDD, Gurobi consistently outperforms Gecode. Furthermore, we observe that the difference made by SDD minimisation is larger for the Gecode methods than for the Gurobi methods. This can largely be explained by the results in Figure 5.6, and by those in Figure 5.7,

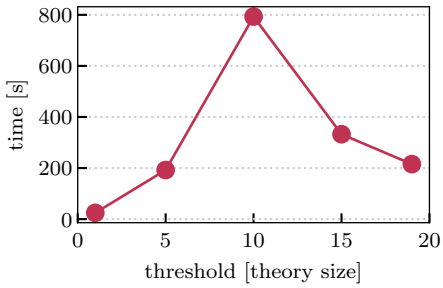


Figure 5.5: *Example of performance of Gurobi on a decomposed non-minimised SDD for different thresholds, for problem* **hepth47***, variant 1.*

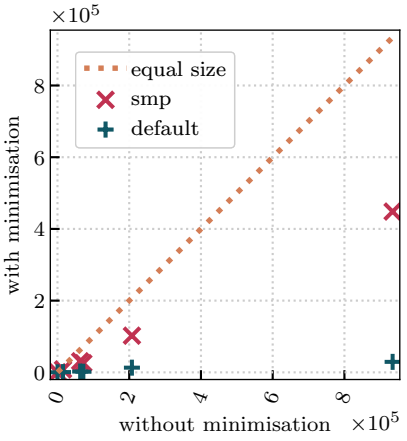


Figure 5.6: *Comparison of size reduction by SDD minimisation algorithms.*

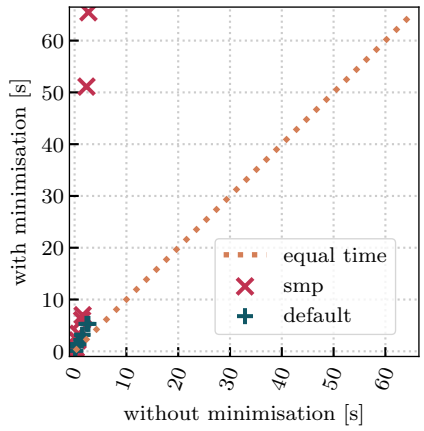


Figure 5.7: *Comparison of SDD compilation times.*

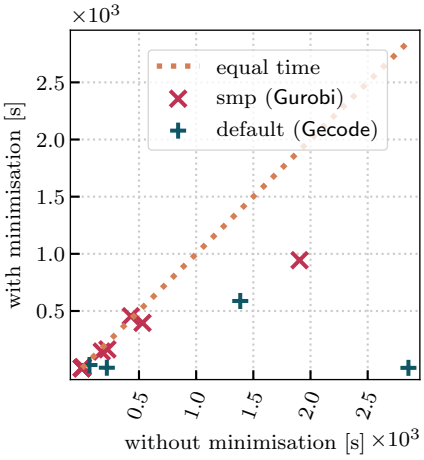


Figure 5.8: *Comparison of full pipeline solving times for the two solvers.*

which answers **Q4**. The latter results show that generally, compiling SDDs is a matter of seconds, whether they are being minimised or not. The exception is the **hepth5** problem, which takes tens of seconds to compile into an SDD when using SMP minimisation. Observe from the table that minimisation is still useful here, as it reduces solving time enough to make up for the extra minimisation time. We note that the minimisation algorithms are based on heuristics, and minimisation speed-up may lie in the improvement of these heuristics.

Finally, Figure 5.8 shows that the time that is gained during the optimisation part of the entire solving chain, can be orders of magnitude larger than the time lost by minimising the SDD. We do note that, since compiling the SDD can be done in seconds, this effect is less noticeable for the smaller problems.

5.4 Conclusion

In this chapter, and in Section 4.3, we showed how we can combine generic probabilistic programming technology (in the form of the SC-ProbLog programming language and knowledge compilation) and CP and MIP solvers (Gecode and Gurobi) to solve the type of SCPs that we described in Section 1.2. We combined these elements into a pipeline for solving these problems.

In constructing this pipeline, we presented two key contributions. The first is the decomposition of a hard constraint on an AC representation of a probability distribution (derived from its SDD representation), into a multitude of local constraints, such that they can be fed directly into an off-the-shelf CP or MIP solver.

The second key contribution in this chapter is the SDD minimisation algorithm that preserves properties that ensure that a constraint on an SDD representation of a probability distribution can be translated into a MIP model that is linearisable, while minimising the size of the SDD. This minimisation algorithm preserves a property of the vtree that defines the variable order of the SDD, which we call the SMP property.

In our experiments, we evaluated different variants of this pipeline on a range of problem instances from two different domains, exploring different combinations of stochastic constraints and optimisation criteria, and linear constraints and optimisation criteria. We showed that the pipeline that uses the MIP solver Gurobi consistently solved these instances faster than the pipeline that used the CP solver Gecode.

We also compared the running times of the pipelines when they use no SDD minimisation (which makes the compilation step fast, but results in larger models), or when they use SMP minimisation (in the case of the pipeline that uses Gurobi) or default minimisation (in the case of the pipeline that uses Gecode). Here we found that in both pipelines, minimisation consistently leads to shorter overall running times. In some cases minimisation makes the difference between a problem being solvable within the one hour time limit, or not. For the problems that are solvable within that time, minimisation can decrease the overall solving time with up to two orders of magnitude.

We note that a somewhat related study by Hemmi *et al.* also proposes a

decomposition-based approach to solving SCPs [78]. There are, however, some important differences in both the scope and approach between their work and ours. While Hemmi *et al.* solve multi-stage SCPs, our focus is on single-stage ones. In multi-stage SCPs, the solution consists of a *policy* that dictates which decisions should be made in each stage as a scenario unfolds. Hemmi *et al.*'s methods solve multi-stage SCPs by generating all possible scenarios for the next stage, solving the SCP for each scenario, and continuing recursively. This decouples the stages from each other (which they call "relaxation"), and hence may cause constraints on decisions that span multiple stages to become decoupled. They address this by detecting which constraints are violated, pruning those partial solutions from the search space, and iteratively refine the solution.

Hemmi *et al.*'s approach is suitable for multistage problems, while ours only supports single-stage problems. However, since their method requires all scenarios to be generated, it can only handle small problems. While their experiments show results for problems with up to a total of 343 possible scenarios and 150 decisions (all solved within 800 seconds), the largest problem in our problem set has 6.7 million scenarios, and 86 decisions (solved within 444 seconds by our fastest method, but not solved within an hour by our slowest). Note that the approach of Hemmi *et al.* is highly parallelisable, and their results were run using a parallelised implementation, using 32 hyper threaded cores. In our experiments, Gurobi is the only solver that is parallelised and attempts to use as many threads as possible, which was 8 in our case.

While the results presented in this chapter are clearly encouraging, the methods we presented do have some weaknesses. Chief among them is that, in decomposing the constraint on the AC, we lose information about the structure of the underlying SDD. We expect that a dedicated propagation algorithm for such a constraint that exploits the structure instead of erasing it, may well outperform the CP-based implementation of the decomposition method, if such a propagator can be devised and implemented. The resulting method may then become competitive with the MIP-based decomposition method.

6

Global SCMD propagation

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

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

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

6.1 Introduction

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

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

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

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

The main algorithmic contributions in this chapter are the following:

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

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

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

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

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

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

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

6.2 OBDDs and generalised arc consistency

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

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

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

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

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

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

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

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

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

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

6.3 Monotonicity

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

6.3.1 Monotonic probability distributions

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

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

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

the following holds:

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

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

6.3.2 Local monotonicity

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

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

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

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

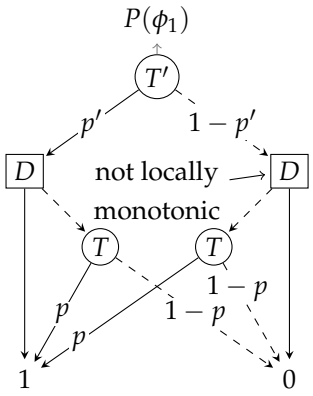
Theorem 6.3.1. *If a probability distribution $P(\phi|_\sigma)$ can be represented by a locally monotonic OBDD as defined in Definition 6.3.2, then it is a monotonic distribution, as per Definition 6.3.1.*

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

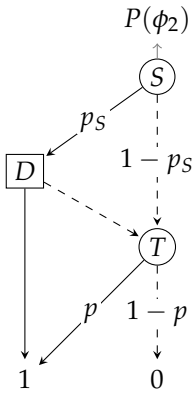
The inequality holds trivially if r is a leaf, and is in fact an equality in this case. We now assume that the inequality holds for all descendants of a node r, and distinguish the following cases:

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

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



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



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

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

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

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

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

Example 6.3.1 (An OBDD that is not locally monotonic)**.** *Recall the power grid reliability problem of Example 4.2.4, and let us focus on encoding the survival probability of a single power line. The decision to reinforce this power line or not, is denoted by Boolean decision variable D. Following the notation used in Example 4.2.4, we denote its survival probability when it is not reinforced p, and its survival probability when it is reinforced with p′, such that* $0 < p < p' < 1$. *The associated stochastic variables are T and T′, which take value* $\top$ *if the line survives and the value* $\bot$ *if it does not. We then encode the event* ϕ_1 *of survival of this power line, with the following formula:*

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

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

An example OBDD encoding of this formula is shown in Figure 6.1a. Consider the decision node on the right. If we set $D := \bot$, the score of that node is p. However, if we set $D := \top$, then the score in that node is 0. Consequently, flipping the value of D from $\bot$ to $\top$ is not locally monotonic (see Definition 6.3.2), even though the behaviour in the root of the OBDD remains monotonic. This local non-monotonic behaviour is directly due to the fact that we can falsify ϕ_1 by fixing $D := \top$, because of the second clause.

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

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

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

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

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

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

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

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

6.3.3 Stochastic constraint on monotonic distributions

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

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

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

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

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

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

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

6.4 Global SCMD propagation

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

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

6.4.1 Naïve SCMD propagation

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

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

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

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

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

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

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

6.4.2 A full-sweep SCMD propagator

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

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

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

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

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

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

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

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

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

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

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

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

Our global SCMD propagation algorithm is based on the following:

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

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

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

Proof. Observe that Equation 6.7 can be read as:

$$\begin{aligned} \frac{\partial f(D, \sigma' \setminus \{D\})}{\partial D} &= v|_{\sigma' \setminus \{D\}, D=\top}(r) - v|_{\sigma' \setminus \{D\}, D=\bot}(r) && (6.12) \\ &= \left(u_{r \to r^+} \cdot v|_{\sigma' \setminus \{D\}, D=\top}(r^+) + u_{r \to r^-} \cdot v|_{\sigma' \setminus \{D\}, D=\top}(r^-) \right) - \\ &\quad \left(u_{r \to r^+} \cdot v|_{\sigma' \setminus \{D\}, D=\bot}(r^+) + u_{r \to r^-} \cdot v|_{\sigma' \setminus \{D\}, D=\bot}(r^-) \right), && (6.13) \end{aligned}$$

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

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

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

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

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

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

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

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

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

1. A top-down pass over the complete OBDD for calculating all path weights.

2. A bottom-up pass for calculating the values for all nodes in the complete OBDD, calculating the derivatives for each variable in the process.

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

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

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

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

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

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

6.4.3 A partial-sweep SCMD propagator

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

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

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

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

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

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

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

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

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

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

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

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

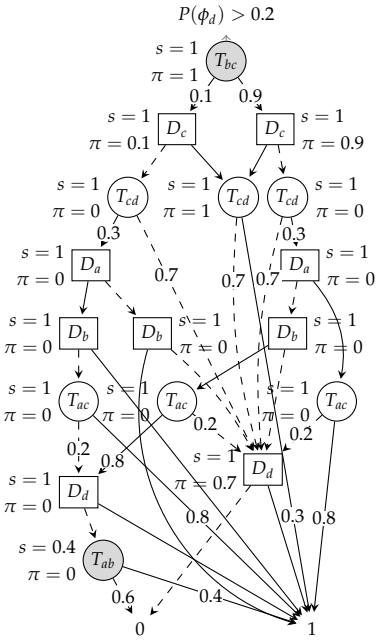
We call the first counter `FreeIn[r]`. It indicates the number of parents r' of r for which there is at least one valid path from an unbound decision node above r' to r'. If `FreeIn[r]`=0, we need not update scores of nodes above r if the score of r changes (**O4**).

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

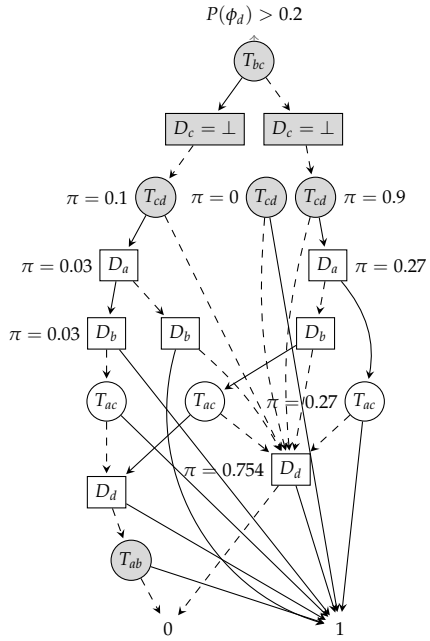
Because of **O6**, we use a third counter, which we call `Reachable[r]`.

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

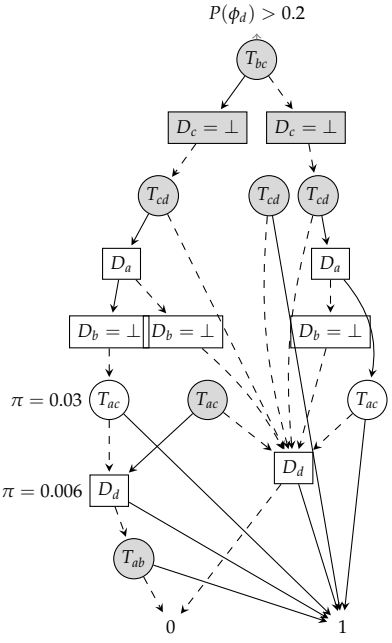
Note that, as observed in **O6**, some decision nodes labelled with free decision variables may become disconnected from the root. Consequently, we do not use their scores to compute the derivative in Equation 6.11. If the OBDD representation of the probability distribution is *not* locally monotonic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



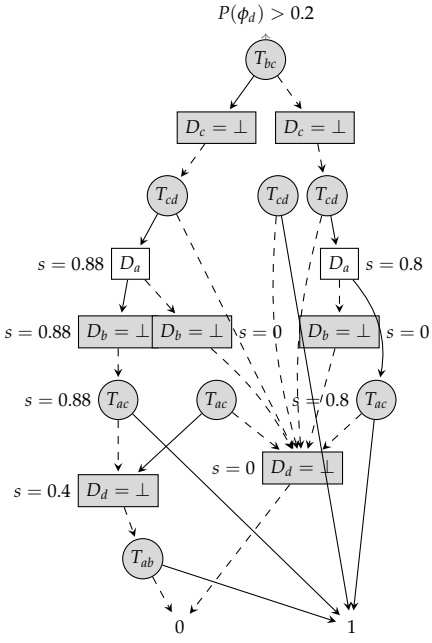
(a) *Iteration 0: just after initialisation.*



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



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



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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

6.4.4 A global constraint SCPMD solving pipeline

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

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

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

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

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

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

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

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

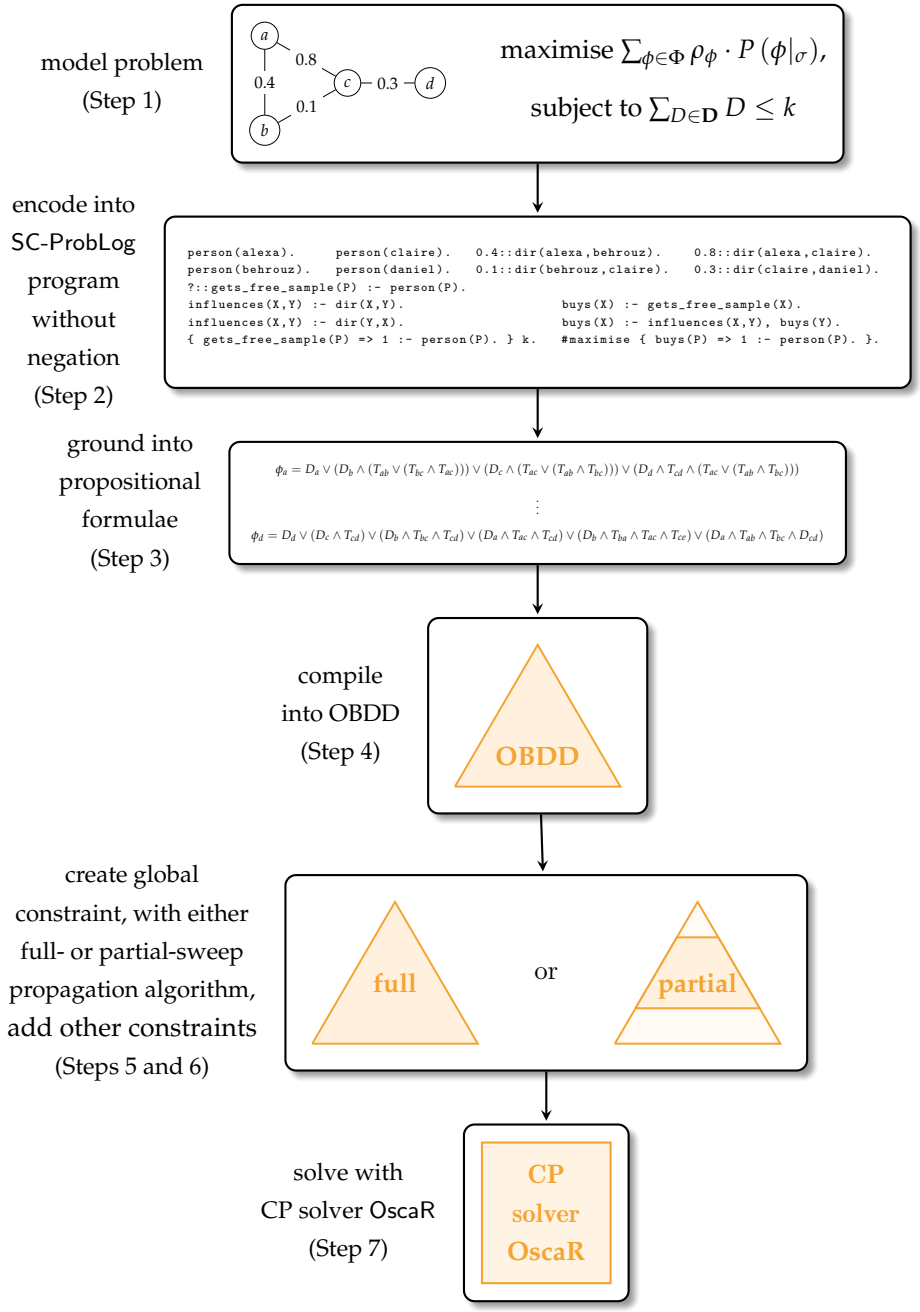


Figure 6.3: *Overview of the global constraint SCP solving pipeline we propose in this chapter.*

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

Step 7: Solve using a CP solver.

6.5 Experimental evaluation

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

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

6.5.1 Research questions

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

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

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

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

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

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

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

6.5.2 Experimental setup

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

Software and hardware

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

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

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

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

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

Overview of solving methods

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

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

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

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

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

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

Parameter settings

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

Problem instances

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

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

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

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

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

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

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

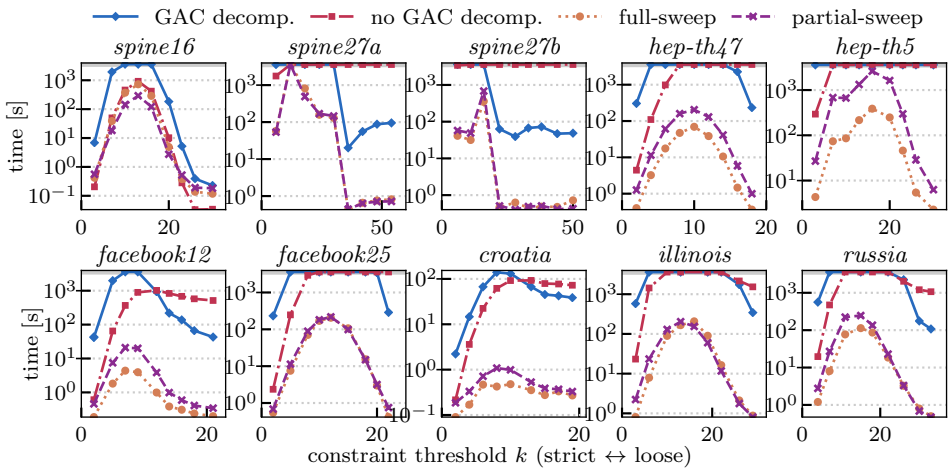


Figure 6.4: *Solving times of CP-decomposition methods and global SCMD methods. Cut-off time is 3 600 s (1 hour). Vertical axes are log scale.*

6.5.3 Results

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

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

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

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

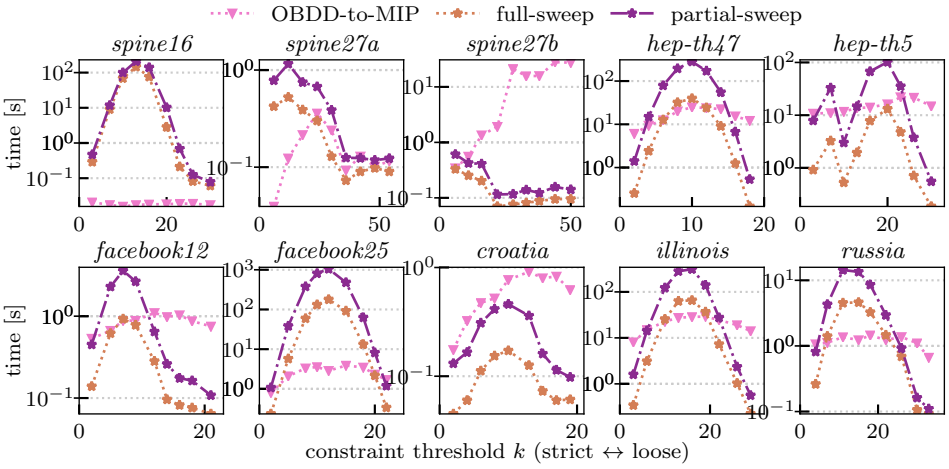


Figure 6.5: *Solving times of MIP-based OBDD decomposition method, compared to the full-sweep and partial-sweep methods. Cutoff time is* 3 600 *s (1 hour). Vertical axes are log scale.*

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

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

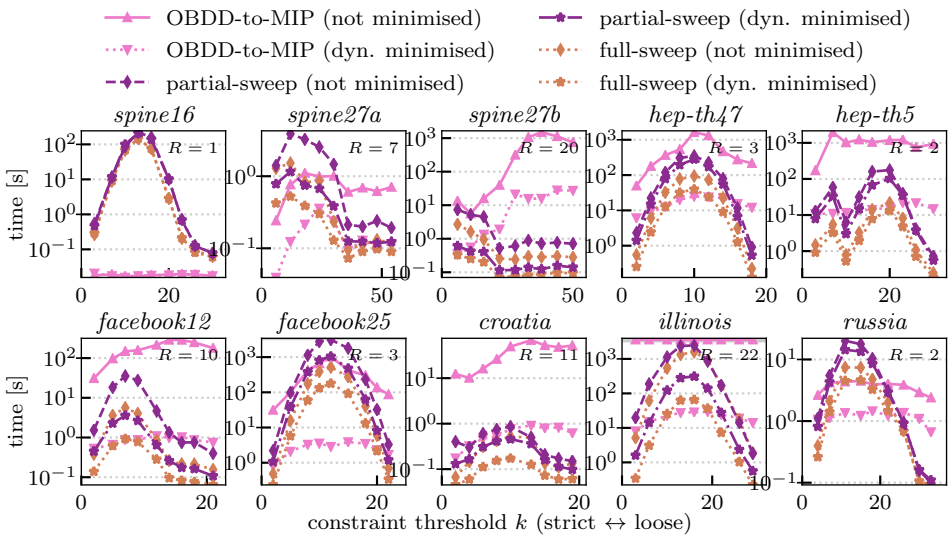


Figure 6.6: *Solving times of OBDD-to-MIP and global SCMD propagators on OBDDs of different sizes: either obtained without minimisation, or by using dynamic minimisation during compilation. Cutoff time is 3 600 s (1 hour). Vertical axes are log scale. R indicates the size ratio* $|OBDD_{nm}|/|OBDD_{dm}|$.

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

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

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

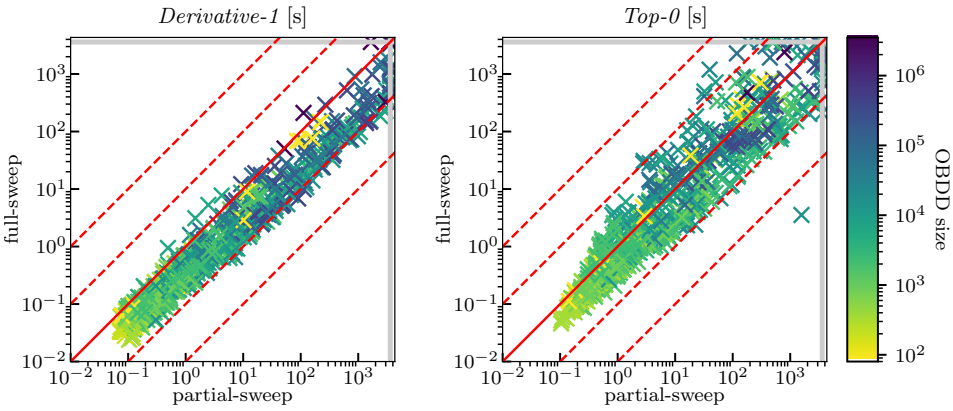


Figure 6.7: *Solving times of the full-sweep and partial-sweep SCMD propagators on dynamically minimised and non-minimised OBDDs, for 52 instances (Table 6.1). We compare two branching heuristics:* Derivative-1 *and* Top-0*. Cutoff time is 3 600 s (1 hour).*

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

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

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

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

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

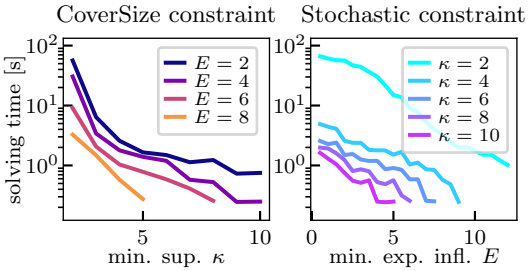
	All OBDDs (936)		Top 5% largest OBDDs (45)	
	Derivative-1	*Top-0*	*Derivative-1*	*Top-0*
full	847 s (21)	2 366 s (59)	13 817 s (17)	21 028 s (26)
partial	1 373 s (33)	2 470 s (61)	16 389 s (20)	19 585 s (24)
partial/full	1.6	1.0	1.2	0.9

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

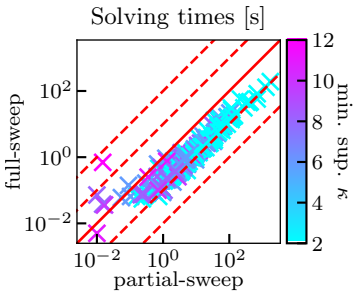
These results confirm that branching heuristics that aim to minimise the size of the active part of the OBDD can indeed give partial-sweep the edge over full-sweep for large OBDDs. *Derivative-1*, on the other hand, leads to smaller search trees. As the active part of OBDDs in this case does not get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

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

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



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



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

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

Lower values of θ and κ correspond to looser constraints. As expected, we see that solving times decrease as the constraints become stricter.

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

6.6 Conclusion

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

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

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

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

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

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

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

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

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

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

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

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

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

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

7

Applying PbO to exact SCPMD solving

In the previous two chapters we presented several *stochastic constraint (optimisation) problem (SCP)* solving pipelines, based on stochastic constraint decomposition (Chapter 5) and global stochastic constraint propagation (Chapter 6). However, we did not explore in much detail how parameter settings affect the performance of our proposed methods, nor did we explore many alternatives for the design choices we made in the process.

We address these two open ends in this chapter, by applying the paradigm of *programming by optimisation (PbO)* to the methods described in the previous two chapters. Specifically, we implement and expose myriad alternative design choices for different elements of the solving pipelines, and then use *automated algorithm configuration (AAC)* to find application-specific optimised configurations of these pipelines. After configuration, we find that the global *stochastic constraint on monotonic distributions (SCMD)* solving pipeline from Chapter 6 outperforms

its closest competitor (a *mixed integer programming (MIP)*-based decomposition pipeline from Chapter 5) on all test sets we considered by up to two orders of magnitude in terms of PAR10 scores. This chapter is based on the following peer-reviewed workshop paper and journal paper:

D. Fokkinga, A.L.D. Latour, M. Anastacio, S. Nijssen, and H. Hoos. 'Programming a Stochastic Constraint Optimisation Algorithm, by Optimisation'. In: *Data Science meets Optimization workshop 2019 (DSO 2019), colocated with IJCAI 2019, Macao, 2019*.

A.L.D. Latour, B. Babaki, D. Fokkinga, M. Anastacio, H.H. Hoos, and S. Nijssen. 'Exact Stochastic Constraint Optimisation with Applications in Network Analysis'. In: *Artificial Intelligence, vol 304, 2022.*

7.1 Introduction

As the results in Sections 5.3 and 6.5 show, different variants of solving methods behave differently on different problem instances. Based on this, we cannot decide what the optimal configuration is for each pipeline, or accurately predict how these or alternative configurations will behave on new problem types. Additionally, we largely relied on default parameter settings, with some minimal exploration of alternatives. Since generic solvers like Gurobi have many parameters, their defaults are unlikely to be optimal for a specific type of problem. While this might give us an indication of how well the decomposition method works 'out of the box', to assess its true potential, we need to tune its parameters. Finally, by learning which parameter settings yield shorter solving times for specific problems, we may also learn more about those problems and how to solve them more efficiently, potentially sparking interesting ideas for future research.

To address these observations, we leverage the PbO paradigm [80], which we briefly explained in Section 3.5. Specifically, in this chapter we make the following contributions:

1. We develop several design alternatives for different parts of decomposition-based and global constraint optimisation pipelines from Chapters 5 and 6 and expose them as configurable parameters (Section 7.2).
2. We apply AAC [79] to these configurable algorithms and demonstrate their effectiveness on benchmarks from two application domains (Section 7.3.3).
3. We then demonstrate how the optimised configurations of these methods generalise to harder problems and different problem settings (Section 7.3.4).

Automated optimisation techniques and tools such as SMAC [86] have been used to solve optimisation problems in approximate probabilistic inference [149], *constraint programming (CP)* solving [99] and MIP solving [85]. However, to the best of our knowledge, they have not yet been applied to the optimisation of the configuration of exact probabilistic inference methods.

In the remainder of this chapter, we first describe how we have applied the PbO paradigm to different elements of the *stochastic constraint (optimisation) problem on monotonic distributions (SCPMD)* solving pipelines described in Section 6.5.2. We then present our experiments in Section 7.3, and conclude this chapter in Section 7.4.

7.2 Design space of SCPMD solving pipelines

In Chapters 5 and 6 we described several approaches to solving SCPMDs. Figure 7.1 shows these different methods schematically, and visualises how they relate to each other. As we discussed above, in this chapter we apply the PbO paradigm on these methods. In this section we describe the different design choices that arise in the last three steps of the methods.

In this section we describe how we implemented alternative design choices where necessary, and how the addition of these design choices influences the size of the parameter space. As an illustration of the size of the parameter space in each step of the methods, Figure 7.1 shows the number of tuneable parameters, and their domain types, where applicable.

Note that we differ in our approach from earlier AAC approaches [85] by separating 'special values' of parameters from their regular domains. Parameters may have, for example, the domain of $\mathbb{N}^+$, but are turned off or tuned automatically when they take value 0. Contrary to earlier approaches, we split these parameters into a *switch parameter* and the *normal parameter*; the former turns the latter on or off, and the latter is only configured if the switch is on.

In the remainder of this section we discuss the different design choices available to us, or added by us, for the different solving methods in Figure 7.1. Note that Figure 7.1 shows the three pipelines as described in Section 6.5.2, with the one difference that the decomposition-based pipelines can now also compile the probability distributions to *sentential decision diagrams (SDDs)*, instead of only to *ordered binary decision diagrams (OBDDs)*. Details on the exact domains and the default values can be found at `github.com/latower/SCPMD-solving`.

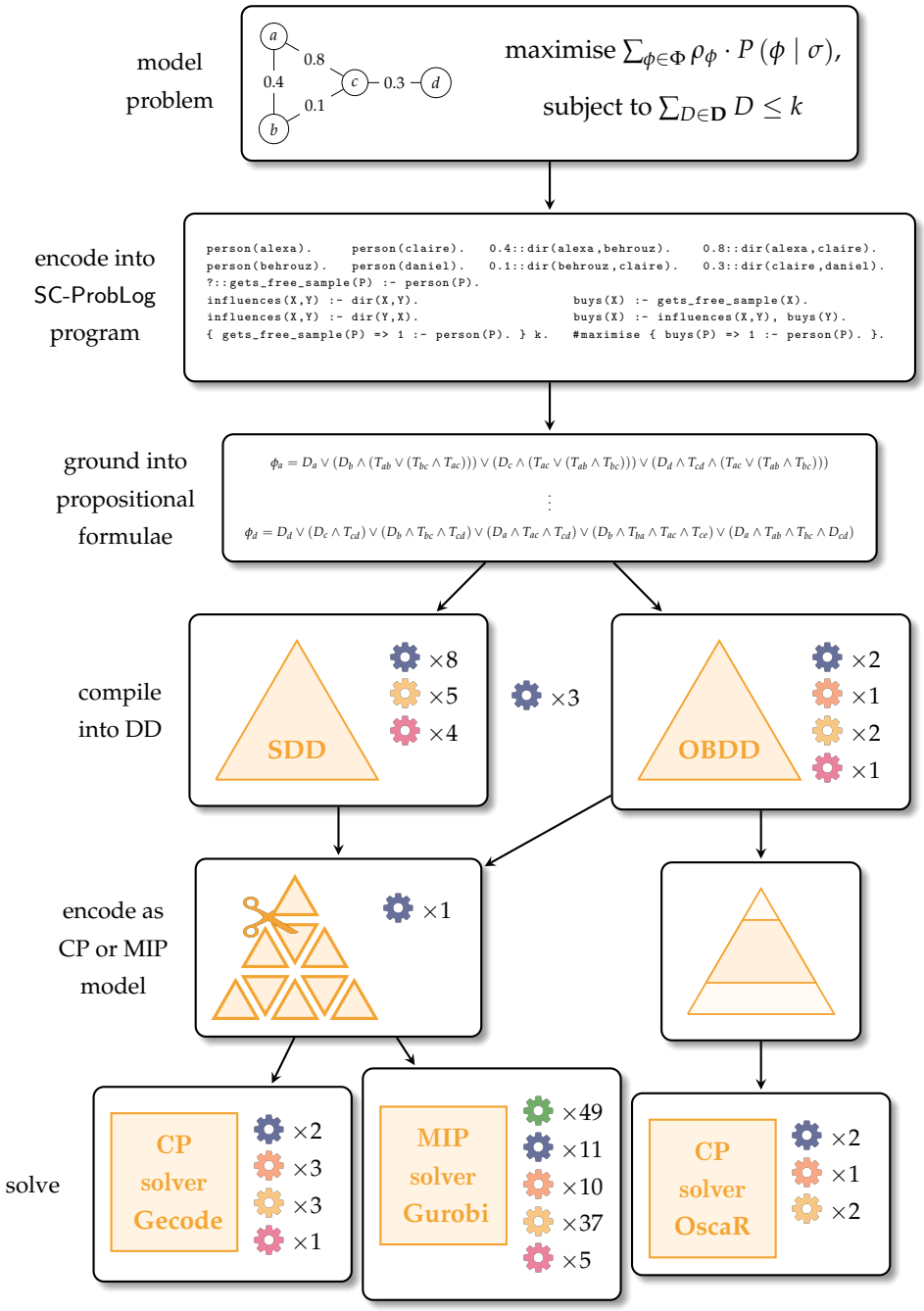


Figure 7.1: *Overview of the different SCPMD solving methods that we evaluate in this chapter, along with an indication of the number of parameters that we configure in each step, with switch, Boolean, categorical, integer, and real domains.*

7.2.1 Knowledge compilation step

The algorithms described in Chapter 5 take constraints on OBDD or SDD representations of probability distributions and decompose them into CP or MIP models. The algorithms proposed in Chapter 6 only operate on OBDDs representations. We summarise the parameters for the knowledge compilation step of our pipelines in Table 7.1.

Specifically, Table 7.1a shows the `Diagram` parameter, which we use to let the configurator choose between OBDD and SDD representations in the decomposition-based pipelines. For both types of *decision diagrams (DDs)*, we let the configurator tune if the knowledge compiler should attempt to minimise the diagram during compilation. Finally, the configurator can also tune whether the compilation algorithm should try to minimise the algorithm after compilation.

Table 7.1b shows the parameters related to minimisation in the CUDD 3.0.0 [168] compiler. In particular, we allow the configurator to exploit different minimisation algorithms: *Sifting (Sif)* [156], *Symmetric Sifting (SymSif)* [136], *Group Sifting (GSif)* [135], *Window Permutation (WP)* [89], a *Simulated Annealing (SA)* approach similar one from literature [21], a *Genetic Algorithm (GA)* inspired by one from the literature [58] and a randomised variable reordering algorithm based on the Sifting algorithm (*Rand*). They can be applied either to dynamic minimisation during the OBDD compilation, or we can make a call to a minimisation algorithm after compiling the OBDD.

The resulting parameter space for OBDD minimisation consists of two categorical parameters, one Boolean parameter, two integer-valued parameters and one real-valued parameter.

We use the SDD minimisation algorithm proposed in Section 5.2.2 as the only option for SDD compilation, as we must make sure that the resulting decomposed constraints can be linearised, for Gurobi to be able to find a solution in all cases. Otherwise, we expose all available parameters for configuration, resulting in eight Boolean parameters, five integer- and four real-valued parameters that can be tuned for the SDD minimisation.

One of these parameters, `ConvThreshold`, is used to define convergence: if the relative size reduction of the diagram is below this threshold, the algorithm has converged. All other parameters either limit the size of intermediate results, or the time that it takes to execute them, for the three vtree operations described in Section 5.2.2. Details are shown in Table 7.1c.

Combining these with the parameters in Table 7.1 results in a total of six categorical parameters, eight switch parameters, ten integer- and five real-valued parameters for the configuration of the compilation phase.

Table 7.1: *The configuration space of the knowledge compilation step of our pipelines, in which the probability distributions are compiled into DDs. Note that some of the parameters are conditional on the value of other parameters. For brevity, we denote the Boolean domain of* {True, False} *with* $\mathbb{B}$*, and use k, M and B to indicate thousands, millions and billions, respectively.*

(a) *Parameters for compilation in general.*

parameter	domain	description
`Diagram`	{*OBDD*, *SDD*}	Compile $\phi(\mathbf{D}, \mathbf{T})$ into an OBDD or SDD.
`DynMinimise`	$\mathbb{B}$	Use dynamic minimise during compilation (if `Minimise` $=$ *True*).
`Minimise`	$\mathbb{B}$	Minimise DD after compilation or not.

(b) *Parameters for OBDD compilation with* CUDD*, all conditioned on* `Diagram` $=$ OBDD *and (*`Minimise` $=$ True *or* `DynMinimise` $=$ True*).*

parameter	domain	description
`VarOrder`	{*Sif*, *SymSif*, *GSif*, *WP*, *SA*, *GA*, *Rand*}	Variable reordering algorithm used for OBDD minimisation.
`Converging`	$\mathbb{B}$	Repeat variable reordering algorithm until no improvement on OBDD size is found (if `VarOrder` $\in$ {*Sif*, *SymSif*, *GSif*, *WP*}).
`MaxSwap`	$[1, 3\text{M}]$	Upper bound on number of times two variables can be swapped in the variable order (if `VarOrder` $\in$ {*Sif*, *SymSif*, *GSif*}).
`MaxSift`	$[1, 3\text{k}]$	Upper bound on number of variables that are sifted, *i.e.*, moved up or down in the variable order (if `VarOrder` $\in$ {*Sif*, *SymSif*, *GSif*}).
`MaxGrowth`	$[0.0, 2.0]$	Upper bound on relative OBDD size increase during minimisation (if `VarOrder` $\in$ {*Sif*, *SymSif*, *GSif*}).
`WSizes`	$\{2, 3, 4\}$	Evaluate permutations of `WSizes` consecutive variables in the variable order at a time (if `VarOrder` $=$ *WP*).

(c) *Parameters for for SDD compilation with the* SDD *package, all conditioned on* `Diagram` = SDD *andand (*`Minimise` = True *or* `DynMinimise` = True*).*

parameter	domain	description
`ConvThreshold`	$[0.0, 50.0]$	Vtree convergence threshold.
`RRCartProdLimOn`	$\mathbb{B}$	Turn Cartesian product limit for right-rotate operations on.
`RRCartProdLim`	$[1, 65\,536]$	Maximum allowed size of a Cartesian product created by right-rotate operation (if `RRCartProdLimOn` = *True*).
`SWCartProdLimOn`	$\mathbb{B}$	Turn Cartesian product limit for swap operations on.
`SWCartProdLim`	$[1, 65\,536]$	Maximum allowed size of a Cartesian product created by right-rotate operation (if `SWCartProdLimOn` = *True*).
`RRTimeLimOn`	$\mathbb{B}$	Turn time limit for right-rotate operations on.
`RRTimeLim`	$[1, 25\mathrm{B}]$	Time limit on right-rotate operation (if `RRTimeLimOn` = *True*).
`SWTimeLimOn`	$\mathbb{B}$	Turn time limit for swap operations on.
`SWTimeLim`	$[1, 25\mathrm{B}]$	Time limit on swap operation (if `SWTimeLimOn` = *True*).
`LRTimeLimOn`	$\mathbb{B}$	Turn time limit for left-rotate operations on.
`LRTimeLim`	$[1, 25\mathrm{B}]$	Time limit on left-rotate operation (if `LRTimeLimOn` = *True*).
`RRSizeLimOn`	$\mathbb{B}$	Turn size growth limit for right-rotate operations on.
`RRSizeLim`	$[1.0, 2.0]$	Size growth limit on right-rotate operation (if `RRSizeLimOn` = *True*).
`SWSizeLimOn`	$\mathbb{B}$	Turn size growth limit for swap operations on.
`SWSizeLim`	$[1.0, 2.0]$	Size growth limit on swap operation (if `SWSizeLimOn` = *True*).
`LRSizeLimOn`	$\mathbb{B}$	Turn size growth limit for left-rotate operations on.
`LRSizeLim`	$[1.0, 2.0]$	Size growth limit on left-rotate operation (if `LRSizeLimOn` = *True*).

7.2.2 Encoding step

We consider two main ways of encoding stochastic constraints. One is the decomposition approach presented in Chapter 5, the other is the global approach presented in Chapter 6.

For the decomposition approach, we take a constraint on a SDD or OBDD representation of the probability distribution, and either encode it into a MIP-model or a CP-model. For the CP-encoding of OBDDs specifically, we consider two variants: one that guarantees *generalised arc consistency (GAC)* (Section 6.2) and one that does not (Sections 5.2.1 and 5.2.2). Since one of the goals in this work is to develop an efficient SCMD propagation algorithm that guarantees GAC and uses an OBDD for the probability distribution encoding, we consider developing a GAC-guaranteeing CP encoding of stochastic constraints on probability distributions represented by SDDs to be outside the scope of this work.

The other main encoding approach is to keep the stochastic constraint as a global constraint on the OBDD representation of the probability distribution. For now, we consider only one such encoding. The accompanying propagation algorithm (Section 6.4) guarantees GAC by design.

Consequently, the encoding step actually only has one parameter, and only if we choose to model the probability distributions with OBDDs and then use the CP-based decomposition method to solve the problem: it determines whether we use the GAC-preserving encoding in this case, or not.

7.2.3 Solving step

In the following, we will briefly discuss the parameter spaces of the three solvers that we use in this work: Gecode, Gurobi and OscaR.

Solving with the decomposition method

For the methods that make use of Gecode and Gurobi to solve a linear program[1] obtained by decomposing a *stochastic constraint on probability distributions (SCPD)*, we enable the configuration of all parameters that are relevant for the speed of solving the problem exactly. We base the choices for domains and default values on earlier work on the automated configuration of Gurobi [85] and Gecode [99]. Considering the fact that Gecode and Gurobi already offer a wide range of branching heuristics, we refrained from exploring additional heuristics for these solvers.

[1] Even though Gurobi can handle quadratic constraints, we limit ourselves to linearised decompositions, as described in Section 7.2.1.

Table 7.2: *The configuration space of OscaR, when using the global SCMD propagator from Section 6.4. Some of the parameters are conditional on the value of other parameters.*

parameter	domain	description
`Sweep`	{*Full*, *Partial*}	Full or partial-sweep propagator.
`VarSelHeur`	{*Top*, *Bottom*, *Derivative*, *Degree*, *Influence*, *Triangle*, *Similarity*, *Simmelian*, *ForestFire*, *Betweenness*, *Random*}	Heuristics to select which variable to branch on next.
`ValSelHeur`	$\mathbb{B}$	Heuristics to select which value to branch on first.
`TimeSteps`	$[1, 1k]$	If `VarSelHeur` $=$ *Influence*.
`NumSamples`	$[1, 100]$	If `VarSelHeur` $=$ *Betweenness*.
`FireProb`	$[0.0, 1.0]$	If `VarSelHeur` $=$ *ForestFire*.
`EdgesBurnt`	$[0.0, 1.0]$	If `VarSelHeur` $=$ *ForestFire*.

The resulting configuration space for solving linear program encodings of SCPs with Gecode consists of two Boolean parameters, three categorical parameters, three integer- and one real-valued parameter. The configuration space of solving linear program encodings of SCPs with Gurobi consists of 49 switch parameters, 11 Boolean parameters, 10 categorical parameters, 37 integer- and 5 real-valued parameters. For practical reasons, we do not list the specific parameters here, but refer the reader to the above-mentioned repository for more details.

Solving with the global SCMD propagation method

Our experiments in Section 6.5 showed that branching order has an important impact on search efficiency. Because we study a variety of problems with different properties (Section 4.5), we decided to add a range of problem-specific branching heuristics to explore this result in more detail.

Table 7.2 shows the parameters for the global SCMD solving algorithm. Aside from the parameter to choose between using the full or partial-sweep algorithm, all parameters are directly related to branching.

The *Top*, *Bottom* and *Derivative* variable branching heuristics (with corresponding value branching heuristics) are described in Section 6.4.3. These heuristics are

derived from the topology of the OBDD (in the case of *Top* and *Bottom*) or dynamically determined during the search (in the case of *Derivative*).

We propose seven new heuristics that take a different approach: they are derived directly from the probabilistic network on which the SCPMD is defined. An eighth new heuristic branches on variables that are selected uniformly at random.

In problems where decision variables are associated with nodes in a network (e.g., Example 4.2.1), the *Degree* heuristic branches based on the unweighted, undirected degree of the nodes. Similarly, *Influence* estimates the influence of nodes in order to quickly find a high-quality solution, inspired by work on social influence [23]. We translate the influence heuristic to problems with decision variables associated with the edges in the underlying network (e.g., Example 4.2.2), by taking for each edge the sum of the influence scores of its endpoints. We compute a degree-based score for edges using the *local-degree* measure from [109].

We observe that problems such as the theory compression problem of the **spine** problem instances or the power grid reliability problem of Example 4.2.2 are very similar to graph sparsification problems. We therefore derived the *Triangle*, *Similarity*, *Simmelian* and *ForestFire* heuristics from recent work on this problem [109, 163]. For the *Triangle* heuristic, we simply take the number of triangles that a node or edge is part of (not taking into account weights or directionality), to create versions of this heuristic that are suitable for problems with decision variables on nodes or edges, respectively. We translated the *Similarity*, *Simmelian* and *ForestFire* heuristics to problems with decision variables associated with the nodes in the underlying network (e.g., Example 4.2.1), by summing the scores of all incident edges on a node (not taking into account their weights or directionality). Finally, we use an estimate of either node or edge betweenness centrality as a proxy for the importance of a decision variable in the *Betweenness* heuristic.

Note that some branching heuristics incur preprocessing time, and that the computational complexity of this preprocessing as well as the quality of the resulting heuristic may depend on additional parameters. We mention these parameters in Table 7.2, but discussing them in detail is beyond the scope of this work. The resulting parameter space of the global SCMD propagator, consists of two Boolean parameters, one categorical parameter, two integer-valued parameters and two real-valued parameters.

7.3 Experimental evaluation

In this section, we report on experiments using AAC to determine which pipeline outperforms the others on two sets of problem instances, and to gauge how much

each pipeline benefits from being automatically configured for these specific sets of problem instances. We first discuss the specific research questions that we are trying to answer. Next, we provide some details on the experimental setup in Section 7.3.2. Finally, we analyse the results of our experiments in Section 7.3.3, to answer the questions.

7.3.1 Research questions

The experiments in this section were designed to answer the following questions:

Q1 How much can we improve the performance of the decomposition methods and the global SCMD method on different real-world problems by automatically configuring these methods for those specific instance sets?

Q2 Which automatically configured method solves these problems best?

Q3 What can we learn about these solvers from the configuration results?

Q4 How do our optimised configurations generalise to harder instances of the same problem type and to instances of a different problem type?

7.3.2 Experimental setup

We briefly review the software, hardware and problem instances that we used for our experiments. In addition to the results in this section, the reader can find more, and more detailed, results at `github.com/latower/SCPMD-solving`.

Software and hardware

For our configuration experiments, we mostly used the software as described in Section 6.5.2. We used the NetworkX 2.2 and NetworKit 5.0.1 Python toolkits for computing the scores used for variable branching heuristics, as described in Section 7.2.3.[2]

SDDs were compiled using a version of the sdd 1.1.1 package [36] we adapted to generate SDDs that can be decomposed into linear programs, as described in Section 5.2.2.[3]

Because of the nature of the parameters described in Section 7.2, we expect that a model-based search process for optimal configurations will yield the best

[2] Available at `networkx.github.io` and `networkit.github.io`.

[3] Available at `reasoning.cs.ucla.edu/sdd/` and `github.com/ML-KULeuven/problog/tree/sc-problog`.

Table 7.3: *Summary of characteristics of the benchmark sets we used in our experiments. We provide the range of sizes of the set of interest $|\Phi|$, numbers of stochastic variables $|\mathbf{T}|$, numbers of decision variables $|\mathbf{D}|$, OBDD sizes $|OBDD|$ and the sizes of the training and test sets.*

name	**problem type**	$\lvert\Phi\rvert$	$\lvert\mathbf{T}\rvert$	$\lvert\mathbf{D}\rvert$	$\lvert$**train**$\rvert$	$\lvert$**test**$\rvert$
facebook	*spread of infl.*	15–30	16–107	15–30	412	411
high-voltage	*power grid rel.*	6–39	30–300	15–150	51	50

results. For our configuration experiments, we chose the general-purpose configurator SMACv3 [86], because it is one of the best-performing configurators that are model-based and freely available.

All experiments in this section were performed on GRACE, a cluster with 32 nodes, each equipped with 94 GB of RAM and two Intel Xeon E5-2683 CPUs with 16 cores, a cache size of 40 MB, running at 2.10 GHz using CentOS Linux 7.7.1908. Running times were measured in CPU seconds. We report on aggregated results by using *penalised average runtime with penalty factor 10 (PAR10)* values as a measure for running time performance.

In our experiments, we chose default values for compilation, CUDD, Gecode and Gurobi based on the literature [85, 99], on the results from Section 5.3, and on their own default settings. The default settings for OscaR were chosen based on the experiments in Section 6.5.3.

Benchmark sets

For automated algorithm configuration, we require a large set of instances. This is because we need disjoint training and testing of sufficient size for the configurator to learn from different instances (training) and then validate its performance on a sufficiently varied set of instances (testing). We created these instances using the processes described in Section 4.5 and summarise them in Table 7.3. All of the SCPMD instances we formulate on these problems are of **Variant 1** (see Section 4.5).

For the **facebook** benchmark set, we select all nodes in a problem instance as our set of interest. We choose the upper bound on the cardinality of the solution to be constant in these examples. Specifically, we use $k = 10$, because it can be expected to yield challenging problems, as seen in our results in Section 6.5. Additionally, fixing this threshold to one value, even for problems with different sizes, is a realistic choice for real-life applications in this setting. After all, com-

Table 7.4: *PAR10 values in CPU seconds for the default (def.) and optimised (opt.) configurations of the three solving methods, for both the training set and the test set. We indicate in brackets the number of examples that hit our cutoff time (600 CPU s). We highlight the smallest PAR10 values on the test sets in* **bold**.

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

panies likely have a marketing budget that does not depend very directly on the size of the social network data they have access to.

We choose the threshold values for the **high-voltage** benchmark set differently. For these examples, we use $k = \lfloor |\mathbf{D}|/2 \rceil$, such that we can reinforce at most half of the total number of power lines in any given problem instance. We believe this to be realistic for real-life applications, since we can assume that the maintenance budgets for power grids might be roughly proportional to their size.

7.3.3 Configuration results

To address **Q1** to **Q3**, we performed fifteen independent 48-hour runs of SMAC on each solving pipeline (Section 7.2), on the two training sets in Table 7.3, minimising the PAR10 (penalised average running time with penalty factor 10) and using a cutoff time of 600 CPU seconds. Then, for each method and each dataset, we evaluated the final incumbent (the configuration with the smallest PAR10 value) on the appropriate test set.

The results in Table 7.4 show that the MIP-decomposition method makes the largest relative improvement after configuration, which answers **Q1**. We explain this by noting that Gurobi has a relatively large configuration space (which gives many options for improvement), compared to Gecode and OscaR, and by noting that we used default settings for Gurobi as our default configuration, while we chose our default configuration of OscaR based on our results in Section 6.5.

We also observe that, even with configuration, the CP-decomposition method is not competitive with the MIP-decomposition method and global SCMD

method, similar to what we see in Figure 6.4. Interestingly, for the CP-decomposition method, the automated configurator chooses the encoding that does not guarantee GAC for both the **facebook** and the **high-voltage** dataset. Once more, it appears that, for CP encodings of SCMDs, a global encoding is more favourable than a decomposed one. However, we see that the performance of the MIP-decomposition method and global SCMD method are comparable and complimentary after configuration, similar to what we see in Figure 6.5: on the **facebook** instances, MIP works better, while on the high-voltage datasets, the global constraint works better; this answers **Q2**.

We provide the optimised configurations obtained from these experiments online at `github.com/latower/SCPMD-solving`. To answer **Q3**, we first note that for the CP-decomposition and MIP-decomposition pipelines, SMAC always chooses to encode the probability distributions as OBDDs, rather than SDDs. Furthermore, here and in the global SCMD propagation pipeline, SMAC tends to favour the group sifting algorithm for OBDD minimisation, which is CUDD's default minimisation algorithm. Remarkably, the optimised configurations for the **facebook** and **high-voltage** sets agree on all parameter choices for OscaR: SMAC chooses to use the full-sweep version of the propagator, combined with the *Derivative-1* branching heuristic. We believe that further, detailed analysis of these and similar results of configuration experiments could provide useful directions for improvements to SCMD solving pipelines and see this as a promising direction for future work.

Finally, we note that the improvement in running time on the **high-voltage** benchmark set is less impressive (and even negative, in the case of CP-decomposition) than on the **facebook** benchmark set. We explain this by noting that the **high-voltage** example set is much smaller than the **facebook** set (and thus has fewer examples to learn from), while the problems tend to be larger (see Table 7.3), causing relatively many examples to hit the cutoff time.

7.3.4 Generalisation of automated configuration results

We addressed **Q4** by running the default and optimised configurations obtained from Section 7.3.3 on the examples in Table 7.3 that were not solved by any solver during the configuration experiment in Section 7.3.3 and therefore represent the hardest instances in the training and test sets. In this new experiment, however, we used a cutoff time of 3 600 CPU seconds instead of 600. Rather than using just one threshold k per problem instance, we ran each configuration with nine different thresholds per example, like we did for the experiments in Section 6.5.

In the configuration experiments, 19 of the **high-voltage** examples in the train-

ing set were never solved. Since we now use these examples again to evaluate the generalisation results, there is some leakage of information. However, since this is only $\frac{19}{351}$ and thus roughly 5% of the instances, we do not expect this to affect the results much. For the **facebook** set there are 5 such instances, which is less than 1%.

Similarly, we ran the optimised configuration obtained on the **facebook** dataset on the **hepth** and **facebook** examples described in Table 6.1, since these are spread-of-influence problems. Finally, we ran the optimised configurations obtained on the **high-voltage** dataset on the **spine** and **high-voltage** examples, because of their similarity. Note that, for practical reasons, there is a small overlap (at most 5%) in the instances used for training in Section 7.3.3 and the **facebook** and **high-voltage** test sets we are using in this experiment. We present the results in Table 7.5 and observe patterns similar to those in Table 7.4.

From Table 7.5a, we see that the results for the harder examples with the larger cutoff time are very similar to the ones shown in Table 7.4 and conclude that our configuration results translate predictably to harder instances of the same problem type, answering part of **Q4**.

Table 7.5b shows very similar results for the **facebook** and **high-voltage** problem instances. This is unsurprising, since they are taken from the same datasets and represent the same problem types as the ones used for the configuration experiments. For the **spine** and **hepth** examples, we notice dramatic improvement in the performance of the MIP-decomposition pipeline, but not so much for the global SCMD pipeline, with negative results for **hepth**. Still, the global SCMD pipeline outperforms the MIP-decomposition pipeline on these examples, with both the default and the optimised configurations. We conclude that the results obtained in Section 7.3.3 translate reasonably to problem instances of different types, with a small advantage for the global SCMD approach, answering the remainder of **Q4**.

7.4 Conclusion

In order to make a fair comparison between the SCMD solving pipelines proposed in Chapters 5 and 6, we applied the paradigm of PbO to these pipelines. This resulted in three highly configurable pipelines, with alternative design choices for the knowledge compilation, encoding and solving components. We used AAC to automatically configure these pipelines for instances from two different real-world application domains.

Our findings indicate that after configuration, the pipeline that encodes proba-

Table 7.5: *PAR10 values in CPU seconds for the default and optimised configurations on two test sets. We indicate in brackets the total number of (problem, k) combinations in the first column. In the other columns, we indicate in brackets how many of those combinations reached the cutoff time of 3 600 CPU seconds. For each problem set, we highlight the lowest PAR10 value for the optimised configurations in bold.*

(a) *Results for the problems in the benchmark sets in Table 7.3, that were not solved by any of the solvers in the configuration experiment of Section 7.3.3.*

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

(b) *Results on the full set of 52 examples in Table 6.1, with 9 threshold values for each example.*

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

bility distributions as OBDDs and then solves the SCPMD using the global SCMD propagator proposed in Chapter 6 tended to outperform the other pipelines. This effect was particularly noticeable in the experiments in which we tested how well the optimised configurations generalise to larger instances, and to instances from different application domains. Note that this is also the pipeline that can only be applied to solving SCPMDs, and not to SCPs in general.

We also found that pipelines tended to favour OBDD representations of probability distributions over SDD representations and that a regret-based branching heuristic is always favoured for the SCMD propagation algorithm.

We applied AAC in the current study with a focus on running time minimisation. Other criteria can be of interest as well. For example, the memory use of knowledge compilers can be prohibitively large, so optimising solving methods

to use less memory could increase the applicability of those methods. Furthermore, to the best of our knowledge, this work represents the first use of AAC in exact probabilistic inference. The configuration results we presented in this work encourage us to expect automated solver configuration to also be beneficial for optimisation solvers for other exact probabilistic inference tasks than the ones discussed in this work.

8

Conclusion and outlook

> Only a question has the capacity to be flexible enough to be wisdom.
>
> Dr. Hannah Gadsby

In this dissertation, we set out to develop solving methods for *stochastic constraint (optimisation) problems (SCPs)* that strike a good balance between convenience, generality and speed. We wanted our methods to solve real-world problems fast enough to be useful, to not be dedicated to a particular problem but support problem settings from a range of application domains, and to be easy to use and accessible. We focused on solving single-stage SCPs that are formulated on probabilistic networks. Our work was motivated by three key limitations of existing methods for solving SCPs:

1. Most existing methods focus on scheduling and planning problems, and are less suited for solving problems formulated on probabilistic networks.
2. There was no language for conveniently modelling stochastic constraint opti-

misation problems on probabilistic networks in particular.

3. There was no automatic pipeline for solving SCPs, once they are modelled.

Below, we discuss how we addressed these limitations and how we met our convenience, generality and speed goals, by answering the research questions stated in Section 1.4. We also briefly discuss interesting remaining challenges that can be addressed in future research, and end this chapter, and indeed this dissertation, with some final closing remarks.

8.1 Research questions, revisited

We now revisit the main research questions that we formulated in Section 1.4, answer them, and reflect on how our contributions addressed the above limitations of existing SCP solving methods.

MRQ1 How can we conveniently model SCPs and specify them to a computer?

Our answer to this question is: SC-ProbLog. As described in Section 4.3, our newly proposed SCP programming language SC-ProbLog provides a convenient way to model the complex probability distributions that are generated by formulating SCPs on probabilistic networks, because it is built on the probabilistic logic programming language ProbLog [52, 64]. Additionally, it provides support for modelling maximisation problems on both stochastic variables and decision variables, because it is also built on ProbLog's successor, DT-ProbLog. We extended DT-ProbLog by adding functionality for both minimisation and maximisation problems, and by adding support for stochastic and linear constraints. It does not (yet) support more complex constraints, such as the CoverSize constraint necessary for modelling *frequent itemset mining (FIM)* problems.

We argued that, because of its declarative nature, we expect SC-ProbLog to provide a very easy-to-learn and quick way for a user to specify highly complex probability distributions, particularly those that arise from the probabilistic networks on which the SCPs that we study in this work, are formulated.

MRQ2 How can we leverage *constraint programming (CP)*, *mixed integer programming (MIP)* and knowledge compilation technology to solve SCPs?

In Chapters 5 and 6 we presented a number of SCP solving pipelines. Each takes as input an SC-ProbLog model of an SCP, compiles the underlying probability distribution into either *ordered binary decision diagrams (OBDDs)* or *sentential*

decision diagrams (SDDs), formulates a stochastic constraint on those *decision diagrams (DDs)* representations of the probability distributions, and feeds an encoding of that constraint on the DDs into a CP or MIP solver.

Combining the benefits of knowledge compilation for tractable probabilistic inference and CP and MIP solvers for efficient search is not a trivial task. It requires a translation of a constraint on a DD representation of a probability distribution into the appropriate solver in a manner that is not only efficient, but also easy to use.

Our first approach, the decomposition method presented in Chapter 5, leverages existing CP and MIP solving technology by simply decomposing the constraint on the DD into a multitude of local constraints, which are then fed to the solver. It is straightforward to implement and can in principle be used in any CP or MIP solver that supports variables with real domains, including Gurobi, Gecode, CPLEX, and CPOptimizer. We can use both OBDDs and SDDs to encode the probability distributions.

A downside of the decomposition method is that the search can be inefficient for CP implementations, since it does not guarantee *generalised arc consistency (GAC)*. We addressed this in Chapter 6 by developing a dedicated stochastic constraint propagation algorithm, which does guarantee GAC, but can only be applied to constraints on probability distributions that exhibit a certain monotonic property. We call the corresponding constraint the *stochastic constraint on monotonic distributions (SCMD)*.

We reflect some more on the role that OBDDs and SDDs play in these methods, and on monotonic probability distributions, in our answer to **MRQ3**.

Note that, in our contributions that answer **MRQ2**, we have presented an idea that we expect to be useful in other contexts as well: solving SCPs by means of a modular approach that decouples knowledge compilation from search. A key benefit of this approach is that we need not reinvent (or rather: reimplement) the wheel, and can use whichever tools are the current state of the art for each element of the pipeline. This makes the resulting pipeline easy to keep up-to-date with the latest developments, and allows for flexible tool-building, where the user can choose between different tools that can fulfil the same role in pipeline. We reflect some more on this in answering **MRQ4**.

MRQ3 How can we leverage the properties of SDDs and OBDDs for faster SCP solving?

We provided two different answers to this question. In Chapter 5, we identified a special property of SDDs that allows global constraints on probability distribu-

tions that are represented by SDDs with that property, to be decomposed into a multitude of linear, instead of quadratic, constraints, making them faster to solve by both CP and MIP solvers. We also implemented an SDD minimisation algorithm that preserves this property.

All OBDDs already have that property, and thus always yield constraint decompositions that can be linearised. Studying OBDDs, however, we presented a way to exploit the specific property of OBDDs that their internal nodes are labelled with variables, to develop a *global* constraint propagation algorithm, in Chapter 6. A limitation of this algorithm is that it is only suitable for solving constraints on monotonic probability distributions (*stochastic constraints on monotonic distributions (SCMDs)*).

However, as we have argued in Chapter 6, this limitation on the types of probability distributions that can be handled by the propagator is not very limiting in practice, since real-life applications that exhibit these monotonic distributions are plentiful. That being said, an interesting line of future work would be to investigate either more general GAC-guaranteeing propagators, or other specialised propagators that guarantee GAC. We also see potential for future work in studying SDDs more carefully to identify properties that may result in SCMD propagation algorithms that are even more efficient in practice than the ones presented in Chapter 6, due to the fact that SDDs can be made to more succinct than OBDDs.

MRQ4 How can we fairly and informatively evaluate the running time performance of complex solving pipelines on problems from different application domains, and ultimately employ these pipelines for solving real-world SCPs?

In Chapter 7, we applied the paradigm of *programming by optimisation (PbO)* [80] to all our solving methods. In doing so, we attempted to take away any bias in our analysis from design choices that favour certain problem types, as well as find optimised parameter settings for problem instances from different application domains, thus taking full advantage of the solving power of our SCP solving pipelines. Instead, we implemented many alternative design choices for the different parts of our SCP solving pipelines. We then used *automated algorithm configuration (AAC)* to automatically configure the resulting, highly configurable, pipeline for different problem settings.

In our experiments, we found that the global constraint propagation approach presented in Chapter 6 outperformed the decomposition methods from Chapter 5 in terms of running time. The experiments presented in Chapter 7 are also encouraging, because they indicate that configuration for specific applications is ef-

fective. Moreover, we find that the application-specific optimised configurations tend to generalise well to larger (harder) instances from the same application domain. In addition, we found that, for the SCP solving pipeline that uses a MIP solver in particular, the optimised configurations also generalise well between application domains, at least for application domains whose SC-ProbLog models have a similar structure.

A direction that we touched upon, yet remained relatively unexplored in the work presented in Chapter 7 is that of parameter importance. Some AAC tools provide functionality for analysing which parameters have a large influence on the performance of an algorithm. While we identified some parameter settings that seem to work universally well in our experiments (mostly related to search heuristics and OBDD minimisation), we leave it to future research to analyse these results more carefully. We believe that such an analysis can be useful to extract insights into specific properties of the solving methods as well as the specific problems studied, which can serve as inspiration for the development of future SCP solving methods.

8.2 Future work

Having answered our main research questions, some technical challenges remain. We now discuss those, as well as other interesting directions for future research.

Having demonstrated the power of SC-ProbLog in modelling SCPs, we see a number of ways in which this modelling language can be further developed to extend its expressiveness. First and foremost, we note that SC-ProbLog's support for constraints and optimisation criteria is still limited to linear and stochastic ones. It does not yet provide support for other constraints and optimisation functions. Chief among them is a lack of support for the CoverSize constraint necessary for modelling FIM problems, such as the top fake news distributors problem described in Section 4.5.4. Another interesting direction of the further development of ProbLog would be to add functionality for multi-objective optimisation problems.

In addition, we see opportunities for extending SC-ProbLog's support to also include constraints relevant to scheduling and vehicle routing problems, since we expect both those kinds of problems to have variants in which probability and relations play a key role. This would include, *e.g.*, interval constraints for specifying the time window in which a task must be completed, or maybe constraints for specifying ranked preferences of users for drivers. The work presented in this dissertation shows that SC-ProbLog is an effective and convenient tool for mod-

elling SCPs, and has the potential to become a flexible and powerful modelling tool for a wide range of applications. Extending SC-ProbLog's syntax and semantics to add support for the kinds of constraints as described above is a first, but vital, step towards fulfilling that full potential.

Alternatively, future research may focus on the development of new ProbLog-based languages. Much like SC-ProbLog was built on DT-ProbLog, we see potential for new languages to be developed based on, or inspired by, SC-ProbLog.

While the pipeline model used for our SCP solving methods has the advantages of being flexible and easy to keep up-to-date with the latest technological advances, it does require a significant effort from the user to make sure that all components are installed correctly. An obvious direction of future work is to take the lessons learnt in this work and to use them to create a dedicated solver for SCPs, naturally still taking a PbO-based approach.

Another challenge is scalability. The experiments presented in Chapters 5 to 7 are performed on problem instances with at most a few hundred stochastic variables and decision variables. While problem instances of this size are not uncommon in real-world domains like the power grid reliability problem described in Section 4.5.3 or the signalling regulatory pathway problem described in Section 4.5.1, modern social networks typically contain millions, rather than hundreds, of users.

For those applications we would need a different approach. Here, we see different possibilities. Firstly, we could simply 'shrink' the problem by sampling the network or aggregating nodes, if possible. The network analysis community has produced many effective network sampling techniques that could be applied to the probabilistic networks that are integral to the problems studied in this work, see, *e.g.*, [72, 105]. While these methods are promising, their use requires some, and perhaps even prohibitively much, expertise from the user, since different sampling techniques should be used for different types of networks and different sampling goals.

A perhaps more obvious approach would be to simply not solve the SCP *exactly*, but rather *approximately*. Throughout this dissertation, we have assumed that the probabilities are given, but we have never questioned the accuracy of those probabilities (and rightly so, because that is out of the scope of the work here presented). However, it is fair to ask what it really means that someone "has a 30% chance of influencing their friend"? And how do we know that it is 30% and not 25% or even 50%? How much does it matter? How much precision in the exact probability makes sense here? Given these questions, it may seem strange that we are solving SCPs *exactly*, and with arbitrary precision. Additionally, in

practice we likely do not need arbitrary precision, although exact methods can be very useful when evaluating the precision of approximate methods.

We now briefly list some ideas on how to incorporate approximations in SCP solving techniques. A first class of methods would maintain the steps in the pipelines as presented in this work.

One first alternative to the exact solving pipelines is an *anytime* solving pipeline. When we use the SCMD propagators presented in Chapter 6 to solve problems with a stochastic optimisation criterion, we can simply treat the constraint optimiser as an anytime solver. The longer it runs, the better the solution, but the user can stop the process at any time if they are satisfied with the best-found solution thus far. In this context, an interesting line of future research would be to develop branching heuristics aimed at finding a very good solution very quickly, even if finding the optimal solution takes longer.

In addition, we could investigate the use of local search and sampling in the solving of the SCP, finding local optima in the search space and losing the guarantees of exact optimisation. Stochastic local search techniques and decomposition techniques for approximation have been used widely in *constraint satisfaction problem (CSP)* solving and probabilistic inference alike [78, 154]. We expect that all methods described in this work are, or can easily be made, compatible with these techniques. Similarly, we could further explore, *e.g.*, local search and sampling algorithms for the minimisation of DD representations of probability distributions, or continue our search for minimisation algorithms that yield DDs with specific properties that can be effectively exploited for faster SCP solving.

In Sections 2.4 and 2.5, we argued that a knowledge compilation approach to probabilistic inference has many advantages, especially in a context in which we may want to re-evaluate certain probabilities. A downside of this approach, however, is that knowledge compilation may require amounts of memory that are exponential in the input size of the problem, which can become prohibitive when problems are too large. Additionally, compiled diagrams contain all the information needed for *exact* inference, but, as we argued above, this might not be needed or necessary. An interesting line of future research would therefore ask how to compile 'approximate' DDs, that only contain all information needed to compute probabilities within a certain, pre-specified precision. Alternatively, maybe these diagrams could be compiled as much as the solver expects is needed to verify if a stochastic constraint is satisfied, or even iteratively refined if need be during the solving process. All these forms of approximate compilation could be part of an SCP solving pipeline like the ones presented in this dissertation.

We also have some ideas on approximate SCP solvers that move away from

the pipelines presented in this work. For example, we consider an interesting line of research to be one in which we do *not* use knowledge compilation, but instead go back to the model counters on which most knowledge compilers are based. Contrary to knowledge compilers, most (weighted) model counters give the user the option to limit their memory use, thus guaranteeing that probabilities can be computed without exceeding the memory of the machine. There is a wide range of exact and approximate (weighted) model counters that can be employed as part of a SCP solving system, *e.g.*, [29, 63, 74, 75, 128, 134, 166, 169, 173]. Note that, in this approach, we would let go of the pipeline model and instead focus on developing a dedicated solver.

Perhaps more interestingly, we could modify modern DPLL-based (weighted) model counters to obtain a solver similar to Littman *et al.*'s algorithm for solving *extended SSAT (XSSAT)* [111]. This system would solve SCPs by encoding them in *conjunctive normal form (CNF)* and combining DPLL with branch-and-bound, in order to not have to traverse the entire search tree, but still be able to find an optimal value for a stochastic objective function or check if a stochastic constraint is satisfied. Naturally, this could also come in the form of an anytime algorithm, or the form of another kind of approximation algorithm.

Alternatively, it could be used to, *e.g.*, find bounds on the value of the objective function. Note that this approach would require all constraints to be encoded into a CNF, which may not be possible for all constraints, or make the CNF blow up too much to be feasible. In addition, algorithms of this kind put strong constraints on the order in which the DPLL algorithm branches on the variables, which hinders the solver's ability to keep the search tree small. Much like bucket elimination-style algorithms, these constraints on the branching order can be relaxed to find approximations rather than exact solutions.

Finally, we believe that the methods we presented can also be applied in other contexts than those studied here. Many possibilities particularly remain for the further integration of CP and probabilistic programming, given the limitations on the types of constraints and probabilistic models studied in this work. As mentioned in the discussion of **MRQ1**, extending SC-ProbLog's syntax and semantics to include support for a wider variety of constraints would go a long way towards achieving this, but we also imagine that other types of stochastic constraints, and the implementation of their propagators, would be a valuable contribution towards making the techniques presented in this work more widely applicable. We hope that future researchers will specifically take a GAC-by-design approach to propagator development when crafting new, or improved, propagation algorithms for constraints on probability distributions, like we did in Chapter 6.

Additionally, it is our hope that Chapter 7 serves as inspiration for future researchers. We not only believe that a PbO-based approach to algorithm development can aid in unlocking the full potential of solving methods, we also believe that an AAC-based approach to evaluating the resulting methods is a good antidote against the cherry picking of results. We believe that a thorough, AAC-based evaluation of solving methods for any problem, but $\mathcal{NP}$-hard ones in particular, provides the reader with an honest and nuanced insight into the strengths and weaknesses of these methods. It is our hope that this work contributes to nurturing a scientific work ethic that includes PbO and AAC as standard elements in algorithm design and software development.

8.3 Conclusion

Scientists [133], policy makers [185] and companies [7, 150, 181] have to make decisions under constraints and uncertainty on a daily basis. When the stakes are high, *e.g.*, because they must allocate large sums of money or take decisions that influence the lives of people, we want those decisions to be optimal with respect to some kind of objective. Even if the stakes are lower, we want our decisions to be as good as possible.

In this work we focused on developing exact methods for solving single-stage *stochastic constraint (optimisation) problems (SCPs)* that are formulated on probabilistic networks. We chose to focus on this particular subset of SCPs because we found that the literature lacked tools for solving such problems, despite their ubiquity. We believe that we presented the reader with encouraging results, and thus motivation for further research into this topic.

We also believe that the way in which we performed our research is exemplary of what we believe should be the standard in computer science research in general: by taking a PbO-based approach to algorithm development, and an AAC-based approach to evaluating our methods and exploiting their full potential. While AAC has already shown its power in the realm of MIP and CP solving to some extent already [85, 99], to the best of our knowledge, this work presents the first attempt at applying PbO and AAC to the development of exact probabilistic inference methods. It is our hope that this work establishes the use of PbO and AAC as a new best practice in the probabilistic inference community.

We conclude this chapter by observing that, in answering the four main research questions listed in Section 1.4, we have addressed the three key limitations of the existing work on SCP solving listed at the beginning of this chapter. Our goal was to develop SCP solving methods that strike a good balance between con-

venience, generality and speed. As discussed above, we indeed sometimes had to make choices that sacrificed generality over speed, speed over convenience, or convenience over generality, demonstrating once again that there is no such thing as a free lunch. We do, however, believe that we explored the trade-offs between these three goals, and struck a reasonable balance, resulting in practical tools. The above demonstrates that, even though we made significant progress in SCP solving in the work presented in this dissertation, our work also opens many avenues to future research in both exact and approximate methods.

We thus believe that we have made a significant and promising contribution to helping humans in science and society make better choices, even when faced with limitations and an uncertain universe.

Bibliography

[1] A. M. Abdelbar and S. M. Hedetniemi. Approximating MAPs for belief networks is NP-hard and other theorems. *Artificial Intelligence*, 102(1):21–38, 1998.

[2] C. C. Aggarwal and J. Han, editors. *Frequent Pattern Mining*. Springer, 2014.

[3] C. Ansótegui, Y. Malitsky, H. Samulowitz, M. Sellmann, and K. Tierney. Model-based genetic algorithms for algorithm configuration. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI-15)*, pages 733–739. AAAI Press, 2015.

[4] A. Antonucci and T. Tiotto. Approximate MMAP by marginal search. In R. Barták and E. Bell, editors, *Proceedings of the 33rd International Florida Artificial Intelligence Research Society Conference (FLAIRS-20)*, pages 181–184. AAAI Press, 2020.

[5] U. Apsel and R. I. Brafman. Lifted MEU by weighted model counting. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence (AAAI-12)*. AAAI Press, 2012.

[6] S. Arora and B. Barak. *Computational Complexity — A Modern Approach*. Cambridge University Press, 2009.

[7] B. Babaki, T. Guns, and L. De Raedt. Stochastic constraint programming with AND-OR branch-and-bound. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI-17)*, pages 539–545. IJCAI, 2017.

[8] F. Bacchus, S. Dalmao, and T. Pitassi. Algorithms and complexity results for #SAT and Bayesian inference. In *44th Symposium on Foundations of Computer Science (FOCS-03)*, pages 340–351. IEEE Computer Society, 2003.

[9] F. Bacchus, S. Dalmao, and T. Pitassi. DPLL with caching: A new algorithm for #SAT and Bayesian inference. *Electronic Colloquium on Computational Complexity (ECCC)*, 10(003), 2003.

[10] F. Bacchus and T. Walsh. Propagating logical combinations of constraints. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI-05)*, pages 35–40. Professional Book Center, 2005.

[11] R. I. Bahar, E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebraic decision diagrams and their applications. *Formal Methods in System Design*, 10(2/3):171–206, 1997.

[12] P. Balaprakash, M. Birattari, and T. Stützle. Improvement strategies for the F-Race algorithm: Sampling design and iterative refinement. In *Proceedings of Hybrid Metaheuristics, 4th International Workshop (HM-07)*, pages 108–122. Springer, 2007.

[13] A. Bart, F. Koriche, J. Lagniez, and P. Marquis. An improved CNF encoding scheme for probabilistic inference. In *22nd European Conference on Artificial Intelligence (ECAI-16), including Prestigious Applications of Artificial Intelligence (PAIS-16)*, volume 285 of *Frontiers in Artificial Intelligence and Applications*, pages 613–621. IOS Press, 2016.

[14] M. Ben-Ari. *Mathematical Logic for Computer Science, 3rd Edition*. Springer, 2012.

[15] C. Bessière and P. V. Hentenryck. To be or not to be . . . a global constraint. In *Proceedings of the 9th International Conference on Principles and Practice of Constraint Programming (CP-03)*, volume 2833 of *Lecture Notes in Computer Science*, pages 789–794. Springer, 2003.

[16] A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.

[17] M. Birattari, Z. Yuan, P. Balaprakash, and T. Stützle. F-race and iterated f-race: An overview. In *Experimental Methods for the Analysis of Optimization Algorithms*, pages 311–336. Springer, 2010.

[18] S. Bistarelli and F. Rossi. Semiring-based soft constraints. In *Concurrency, Graphs and Models*, volume 5065 of *Lecture Notes in Computer Science*, pages 155–173. Springer, 2008.

[19] V. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics Theory and Experiment*, 2008, 04 2008.

[20] H. L. Bodlaender, F. van den Eijkhof, and L. C. van der Gaag. On the complexity of the MPA problem in probabilistic networks. In *Proceedings of the 15th European Conference on Artificial Intelligence (ECAI-02)*, pages 675–679. IOS Press, 2002.

[21] B. Bollig, M. Löbbing, and I. Wegener. Simulated annealing to improve variable orderings for OBDDs. In *International Workshop on Logic and Synthesis*, page 5, 1995.

[22] B. Bollig and I. Wegener. Improving the variable ordering of OBDDs is NP-complete. *IEEE Transactions on Computers*, 45(9):993–1002, 1996.

[23] C. Borgs, M. Brautbar, J. T. Chayes, and B. Lucier. Maximizing social influence in nearly optimal time. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA-14)*, pages 946–957. SIAM, 2014.

[24] S. Bova. SDDs are exponentially more succinct than OBDDs. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI-16)*, pages 929–935. AAAI Press, 2016.

[25] S. P. Bradley, A. C. Hax, and T. L. Magnanti. *Applied Mathematical Programming*. Addison-Wesley, 1977.

[26] R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691, 1986.

[27] Y. Caseau and F. Laburthe. Solving various weighted matching problems with constraints. *Constraints*, 5(1/2):141–160, 2000.

[28] S. Chakraborty, D. J. Fremont, K. S. Meel, S. A. Seshia, and M. Y. Vardi. Distribution-aware sampling and weighted model counting for SAT. In

Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI-14), pages 1722–1730. AAAI Press, 2014.

[29] S. Chakraborty, D. Fried, K. S. Meel, and M. Y. Vardi. From weighted to unweighted model counting. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI-15)*, pages 689–695. AAAI Press, 2015.

[30] S. Chakraborty, K. S. Meel, and M. Y. Vardi. Algorithmic improvements in approximate counting for probabilistic inference: From linear to logarithmic SAT calls. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI-16)*, pages 3569–3576. IJCAI/AAAI Press, 2016.

[31] A. Charnes and W. W. Cooper. Chance-constrained programming. *Management Science*, 6(1):73–79, 1959.

[32] M. Chavira and A. Darwiche. Compiling Bayesian networks with local structure. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-07)*, pages 1306–1312. Professional Book Center, 2005.

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

[34] M. Chavira, A. Darwiche, and M. Jaeger. Compiling relational Bayesian networks for exact inference. *International Journal of Approximate Reasoning*, 42(1-2):4–20, 2006.

[35] Q. Cheng, F. Chen, J. Dong, W. Xu, and A. T. Ihler. Approximating the sum operation for marginal-MAP inference. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence (AAAI-12)*. AAAI Press, 2012.

[36] A. Choi and A. Darwiche. Dynamic minimization of sentential decision diagrams. In *Proceedings of the 27th AAAI Conference on Artificial Intelligence (AAAI-13)*. AAAI Press, 2013.

[37] A. Choi, D. Kisa, and A. Darwiche. Compiling probabilistic graphical models using sentential decision diagrams. In *Proceedings of the 12th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU-13)*, volume 7958 of *Lecture Notes in Computer Science*, pages 121–132. Springer, 2013.

[38] S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing*, STOC 1971, pages 151–158, New York, NY, USA, 1971. ACM.

[39] H. Cui, R. Marinescu, and R. Khardon. From stochastic planning to marginal MAP. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems (NeurIPS-18)*, pages 3085–3095, 2018.

[40] G. H. Dal, A. W. Laarman, and P. J. F. Lucas. Parallel probabilistic inference by weighted model counting. In *International Conference on Probabilistic Graphical Models (PGM-18)*, volume 72 of *Proceedings of Machine Learning Research*, pages 97–108. PMLR, 2018.

[41] G. H. Dal and P. J. F. Lucas. Weighted positive binary decision diagrams for exact probabilistic inference. *International Journal of Approximate Reasoning*, 90:411–432, 2017.

[42] A. Darwiche. On the tractable counting of theory models and its application to truth maintenance and belief revision. *Journal of Applied Non-Classical Logics*, 11(1-2):11–34, 2001.

[43] A. Darwiche. A logical approach to factoring belief networks. In *Proceedings of the 8th International Conference on Principles and Knowledge Representation and Reasoning (KR-02)*, pages 409–420. Morgan Kaufmann, 2002.

[44] A. Darwiche. A differential approach to inference in Bayesian networks. *Journal of the ACM*, 50(3):280–305, 2003.

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

[46] A. Darwiche. SDD: A new canonical representation of propositional knowledge bases. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI-11)*, pages 819–826. IJCAI/AAAI, 2011.

[47] A. Darwiche. Three modern roles for logic in AI. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS-20)*, pages 229–243. ACM, 2020.

[48] A. Darwiche and P. Marquis. A knowledge compilation map. *Journal of Artificial Intelligence Research*, 17:229–264, 2002.

[49] M. Davis, G. Logemann, and D. W. Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962.

[50] L. De Raedt, K. Kersting, A. Kimmig, K. Revoredo, and H. Toivonen. Compressing probabilistic Prolog programs. *Machine Learning*, 70(2-3):151–168, 2008.

[51] L. De Raedt and A. Kimmig. Probabilistic (logic) programming concepts. *Machine Learning*, 100(1):5–47, 2015.

[52] L. De Raedt, A. Kimmig, and H. Toivonen. Problog: A probabilistic Prolog and its application in link discovery. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-07)*, pages 2462–2467, 2007.

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

[54] R. Dechter and I. Rish. Mini-buckets: A general scheme for bounded inference. *Journal of the ACM*, 50(2):107–153, 2003.

[55] S. Demassey, G. Pesant, and L. Rousseau. A cost-regular based hybrid column generation approach. *Constraints An Int. J.*, 11(4):315–333, 2006.

[56] P. M. Domingos and M. Richardson. Mining the network value of customers. In *Proceedings of the 7th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD-01)*, pages 57–66. ACM, 2001.

[57] C. Domshlak and J. Hoffmann. Probabilistic planning via heuristic forward search and weighted model counting. *Journal of Artificial Intelligence Research*, 30:565–620, 2007.

[58] R. Drechsler, B. Becker, and N. Gockel. Genetic algorithm for variable ordering of OBDDs. *IEEE Computers and Digital Techniques*, 143(6):364–368, 1996.

[59] DTAI Research Group, KU Leuven. ProbLog Python library. https://github.com/ML-KULeuven/problog, 2015–2019.

[60] J. M. Dudek, V. Phan, and M. Y. Vardi. ADDMC: weighted model counting with algebraic decision diagrams. In *The 34th AAAI Conference on Artificial Intelligence (AAAI-20)*, pages 1468–1476. AAAI Press, 2020.

[61] L. Dueñas-Osorio, K. S. Meel, R. Paredes, and M. Y. Vardi. Counting-based reliability estimation for power-transmission grids. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI-17)*, pages 4488–4494. AAAI Press, 2017.

[62] J. K. Fichte, M. Hecher, S. Woltran, and M. Zisser. Weighted model counting on the GPU by exploiting small treewidth. In *Proceedings of the 26th Annual European Symposium on Algorithms (ESA-18)*, volume 112 of *LIPIcs*, pages 28:1–28:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[63] J. K. Fichte, M. Hecher, and M. Zisser. An improved GPU-based SAT model counter. In *Proceedings of the 25th International Conference on Principles and Practice of Constraint Programming (CP-19)*, volume 11802 of *Lecture Notes in Computer Science*, pages 491–509. Springer, 2019.

[64] D. Fierens, G. Van den Broeck, J. Renkens, D. S. Shterionov, B. Gutmann, I. Thon, G. Janssens, and L. De Raedt. Inference and learning in probabilistic logic programs using weighted Boolean formulas. *Theory and Practice of Logic Programming*, 15(3):358–401, 2015.

[65] D. Fierens, G. Van den Broeck, I. Thon, B. Gutmann, and L. De Raedt. Inference in probabilistic logic programs using weighted CNF's. In *Proceedings of the 27th Conference on Uncertainty in Artificial Intelligence (UAI-11)*, pages 211–220. AUAI Press, 2011.

[66] P. A. Flach. *Simply logical - intelligent reasoning by example*. Wiley professional computing. Wiley, 1994.

[67] N. Flerova, R. Marinescu, and R. Dechter. Weighted heuristic anytime search: new schemes for optimization over graphical models. *Annals of Mathematics and Artificial Intelligence*, 79(1-3):77–128, 2017.

[68] N. Fuhr. Probabilistic datalog: Implementing logical information retrieval for advanced applications. *Journal of the American Society for Information Science*, 51(2):95–110, 2000.

[69] G. Gange, P. J. Stuckey, and P. V. Hentenryck. Explaining propagators for edge-valued decision diagrams. In C. Schulte, editor, *Proceedings of the 19th International Conference on Principles and Practice of Constraint Programming (CP-13)*, volume 8124 of *Lecture Notes in Computer Science*, pages 340–355. Springer, 2013.

[70] G. Gange, P. J. Stuckey, and V. Lagoon. Fast set bounds propagation using a BDD-SAT hybrid. *Journal of Artificial Intelligence Research*, 38:307–338, 2010.

[71] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[72] M. Gjoka, M. Kurant, C. T. Butts, and A. Markopoulou. Walking in facebook: A case study of unbiased sampling of OSNs. In *Proceedings of the 29th IEEE International Conference on Computer Communications (INFOCOM-10)*, pages 2498–2506. IEEE, 2010.

[73] S. W. Golomb and L. D. Baumert. Backtrack programming. *Journal of the ACM*, 12(4):516–524, 1965.

[74] C. P. Gomes, J. Hoffmann, A. Sabharwal, and B. Selman. From sampling to model counting. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-07)*, pages 2293–2299, 2007.

[75] C. P. Gomes, A. Sabharwal, and B. Selman. Model counting: A new strategy for obtaining good bounds. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI-06)*, pages 54–61. AAAI Press, 2006.

[76] R. Gomory. Outline of an Algorithm for Integer Solutions to Linear Programs. *Bulletin of the American Mathematical Society*, 64:275–278, 1958.

[77] P. Hawkins and P. J. Stuckey. A hybrid BDD and SAT finite domain constraint solver. In *Proceedings of the 8th International Symposium on Practical Aspects of Declarative Languages (PADL-06)*, volume 3819 of *Lecture Notes in Computer Science*, pages 103–117. Springer, 2006.

[78] D. Hemmi, G. Tack, and M. Wallace. A recursive scenario decomposition algorithm for combinatorial multistage stochastic optimisation problems. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI-18)*, pages 1322–1329, 2018.

[79] H. H. Hoos. Automated algorithm configuration and parameter tuning. In *Autonomous Search*, pages 37–71. Springer, 2012.

[80] H. H. Hoos. Programming by optimization. *Communications of the ACM*, 55(2):70–80, 2012.

[81] A. Horn. On sentences which are true of direct unions of algebras. *Journal of Symbolic Logic*, 16(1):14–21, 1951.

[82] J. Huang. Combining knowledge compilation and search for conformant probabilistic planning. In *Proceedings of the 16th International Conference on Automated Planning and Scheduling, (ICAPS-06)*, pages 253–262. AAAI, 2006.

[83] J. Huang, M. Chavira, and A. Darwiche. Solving MAP exactly by searching on compiled arithmetic circuits. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI-06)*, pages 1143–1148. AAAI Press, 2006.

[84] M. Huth and M. Ryan. *Logic in computer science : modelling and reasoning about systems*. Cambridge University Press, Cambridge [U.K.]; New York, 2004.

[85] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Automated configuration of mixed integer programming solvers. In *Proceedings of the 7th International Conference Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 6140 of *Lecture Notes in Computer Science*, pages 186–202. Springer, 2010.

[86] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Proceedings of the 5th International Conference on Learning and Intelligent Optimization (LION-11)*, pages 507–523. Springer, 2011.

[87] F. Hutter, H. H. Hoos, K. Leyton-Brown, and T. Stützle. Paramils: An automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36:267–306, 2009.

[88] F. Hutter, H. H. Hoos, and T. Stützle. Automatic algorithm configuration based on local search. In *Proceedings of the 22nd AAAI Conference on Artificial Intelligence (AAAI-07)*, pages 1152–1157. AAAI Press, 2007.

[89] N. Ishiura, H. Sawada, and S. Yajima. Minimization of binary decision diagrams based on exchanges of variables. In *Proceedings of IEEE/ACM International Conference on Computer-Aided Design (ICCAD-91)*, pages 472–475. IEEE, 1991.

[90] F. V. Jensen, K. G. Olesen, and S. K. Andersen. An algebra of Bayesian belief universes for knowledge-based systems. *Networks*, 20(5):637–659, 1990.

[91] J. Jiang, P. Rai, and H. Daumé III. Message-passing for approximate MAP inference with latent variables. In *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems (NIPS-11)*, pages 1197–1205, 2011.

[92] D. Kempe, J. M. Kleinberg, and É. Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the Ninth ACM SIGKDD*

International Conference on Knowledge Discovery and Data Mining (KDD-03), pages 137–146. ACM, 2003.

[93] K. Kersting and L. De Raedt. Bayesian logic programs. *CoRR*, cs.AI/0111058, 2001.

[94] K. Kersting and L. De Raedt. Basic principles of learning Bayesian logic programs. In *Probabilistic Inductive Logic Programming*, volume 4911 of *Lecture Notes in Computer Science*, pages 189–221. Springer, 2008.

[95] A. Kimmig, G. V. den Broeck, and L. D. Raedt. Algebraic model counting. *Journal of Applied Logic*, 22:46–62, 2017.

[96] A. Kimmig, G. Van den Broeck, and L. De Raedt. An algebraic Prolog for reasoning about possible worlds. In *Proceedings of the 25th AAAI Conference on Artificial Intelligence (AAAI-11)*. AAAI Press, 2011.

[97] D. Koller and N. Friedman. *Probabilistic Graphical Models — Principles and Techniques*. MIT Press, 2009.

[98] F. Koriche, J. Lagniez, P. Marquis, and S. Thomas. Knowledge compilation for model counting: Affine decision trees. In F. Rossi, editor, *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI-13)*, pages 947–953. IJCAI/AAAI, 2013.

[99] L. Kotthoff. Constraint solvers: An empirical evaluation of design decisions. *CoRR*, abs/1002.0134, 2010.

[100] J. Kwisthout. Complexity results for enumerating MPE and Partial MAP. In *European Workshop on Probabilistic Graphical Models*, 2008.

[101] S. L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society. Series B (Methodological)*, 50(2):157–224, 1988.

[102] J. Lee, A. T. Ihler, and R. Dechter. Generalized dual decomposition for bounding maximum expected utility of influence diagrams with perfect recall. In *The Workshops of the The 32nd AAAI Conference on Artificial Intelligence AAAI Workshops (AAAI-18 Workshops)*, volume WS-18 of *AAAI Workshops*, pages 674–681. AAAI Press, 2018.

[103] J. Lee, A. T. Ihler, and R. Dechter. Join graph decomposition bounds for influence diagrams. In *Proceedings of the 34th Conference on Uncertainty in Artificial Intelligence (UAI-18)*, pages 1053–1062. AUAI Press, 2018.

[104] J. Lee, R. Marinescu, R. Dechter, and A. T. Ihler. From exact to anytime solutions for marginal MAP. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI-16)*, pages 3255–3262. AAAI Press, 2016.

[105] J. Leskovec and C. Faloutsos. Sampling from large graphs. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-06)*, pages 631–636. ACM, 2006.

[106] L. A. Levin. Universal sequential search problems. *Problems of Information Transmission*, 9(3), 1973.

[107] P. Liberatore. On the complexity of choosing the branching literal in DPLL. *Artificial Intelligence*, 116(1-2):315–326, 2000.

[108] V. Lifschitz. *Answer Set Programming*. Springer, 2019.

[109] G. Lindner, C. L. Staudt, M. Hamann, H. Meyerhenke, and D. Wagner. Structure-preserving sparsification of social networks. In *Social Network Analysis and Mining*, pages 448–454. ACM, 2015.

[110] M. L. Littman, J. Goldsmith, and M. Mundhenk. The computational complexity of probabilistic planning. *Journal of Artificial Intelligence Research*, 9:1–36, 1998.

[111] M. L. Littman, S. M. Majercik, and T. Pitassi. Stochastic boolean satisfiability. *Journal of Automated Reasoning*, 27(3):251–296, 2001.

[112] Q. Liu and A. T. Ihler. Variational algorithms for marginal MAP. *Journal of Machine Learning Research*, 14(1):3165–3200, 2013.

[113] M. Lombardi and M. Milano. Allocation and scheduling of conditional task graphs. *Artificial Intelligence*, 174(7-8):500–529, 2010.

[114] A. K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8(1):99–118, 1977.

[115] R. Marinescu and R. Dechter. AND/OR branch-and-bound search for combinatorial optimization in graphical models. *Artificial Intelligence*, 173(16-17):1457–1491, 2009.

[116] R. Marinescu and R. Dechter. Memory intensive AND/OR search for combinatorial optimization in graphical models. *Artificial Intelligence*, 173(16-17):1492–1524, 2009.

[117] R. Marinescu, R. Dechter, and A. T. Ihler. AND/OR search for marginal MAP. In *Proceedings of the 30th Conference on Uncertainty in Artificial Intelligence (UAI-14)*, pages 563–572. AUAI Press, 2014.

[118] R. Marinescu, R. Dechter, and A. T. Ihler. Pushing forward marginal MAP with best-first search. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI-15)*, pages 696–702. AAAI Press, 2015.

[119] R. Marinescu, R. Dechter, and A. T. Ihler. Stochastic anytime search for bounding marginal MAP. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI-18)*, pages 5074–5081. ijcai.org, 2018.

[120] R. Marinescu, A. Kishimoto, A. Botea, R. Dechter, and A. T. Ihler. Anytime recursive best-first search for bounding marginal MAP. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI-19)*, pages 7924–7932. AAAI Press, 2019.

[121] R. Marinescu, J. Lee, R. Dechter, and A. T. Ihler. AND/OR search for marginal MAP. *Journal of Artificial Intelligence Research*, 63:875–921, 2018.

[122] R. Marinescu, J. Lee, A. T. Ihler, and R. Dechter. Anytime best+depth-first search for bounding marginal MAP. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI-17)*, pages 3775–3782. AAAI Press, 2017.

[123] P. Marquis. Knowledge compilation using theory prime implicates. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 837–845. Morgan Kaufmann, 1995.

[124] J. C. Martin. *Introduction to Languages and the Theory of Computation*. McGraw-Hill, fourth edition, 2011.

[125] R. Mateescu and R. Dechter. Mixed deterministic and probabilistic networks. *Annals of Mathematics and Artificial Intelligence*, 54(1-3):3–51, 2008.

[126] D. D. Mauá and C. P. de Campos. Anytime marginal MAP inference. In *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*. icml.cc / Omnipress, 2012.

[127] K. I. M. McKinnon and H. P. Williams. Constructing integer programming models by the predicate calculus. *Annals of Operations Research*, 21(1):227–245, Dec 1989.

[128] K. S. Meel and S. Akshay. Sparse hashing for scalable approximate model counting: Theory and practice. In H. Hermanns, L. Zhang, N. Kobayashi,

and D. Miller, editors, *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS-20)*, pages 728–741. ACM, 2020.

[129] S. Muggleton. Learning stochastic logic programs. *Electronic Transactions on Artificial Intelligence*, 4(B):141–153, 2000.

[130] N. Nethercote, P. J. Stuckey, R. Becket, S. Brand, G. J. Duck, and G. Tack. Minizinc: Towards a standard CP modelling language. In C. Bessiere, editor, *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming (CP-07)*, volume 4741 of *Lecture Notes in Computer Science*, pages 529–543. Springer, 2007.

[131] M. E. Newman. The Structure of Scientific Collaboration Networks. *Proceedings of the National Academy of Sciences*, 98(2):404–409, 2001.

[132] OscaR Team. OscaR: Scala in OR, 2012. Available from `bitbucket.org/oscarlib/oscar`.

[133] O. Ourfali, T. Shlomi, T. Ideker, E. Ruppin, and R. Sharan. SPINE: A framework for signaling-regulatory pathway inference from cause-effect experiments. In *Proceedings of the 15th International Conference on Intelligent Systems for Molecular Biology (ISMB) & 6th European Conference on Computational Biology (ISMB/ECCB-07, Supplement of Bioinformatics)*, pages 359–366, 2007.

[134] U. Oztok and A. Darwiche. An exhaustive DPLL algorithm for model counting. *Journal of Artificial Intelligence Research*, 62:1–32, 2018.

[135] S. Panda and F. Somenzi. Who are the variables in your neighborhood. In *Proceedings of the 1995 IEEE/ACM International Conference on Computer-Aided Design (ICCAD-95)*, pages 74–77. IEEE Computer Society / ACM, 1995.

[136] S. Panda, F. Somenzi, and B. Plessier. Symmetry detection and dynamic variable ordering of decision diagrams. In *Proceedings of the 1994 IEEE/ACM International Conference on Computer-Aided Design (ICCAD-94)*, pages 628–631. IEEE Computer Society / ACM, 1994.

[137] C. H. Papadimitriou. Games against nature. *Journal of Computer and System Sciences*, 31(2):288–301, 1985.

[138] J. D. Park and A. Darwiche. Solving MAP exactly using systematic search. In *Proceedings of the 19th Conference in Uncertainty in Artificial Intelligence (UAI-03)*, pages 459–468. Morgan Kaufmann, 2003.

[139] J. D. Park and A. Darwiche. Complexity results and approximation strategies for MAP explanations. *Journal of Artificial Intelligence Research*, 21:101–133, 2004.

[140] J. Pearl. *Probabilistic reasoning in intelligent systems — Networks of plausible inference*. Morgan Kaufmann series in representation and reasoning. Morgan Kaufmann, 1989.

[141] G. Perez and J. Régin. MDDs: Sampling and probability constraints. In *Proceedings of the 23rd International Conference on Principles and Practice of Constraint Programming (CP-17)*, volume 10416 of *Lecture Notes in Computer Science*, pages 226–242. Springer, 2017.

[142] G. Perez and J. Régin. Soft and cost MDD propagators. In S. Singh and S. Markovitch, editors, *Proceedings of the 31st Conference on Artificial Intelligence (AAAI-17)*, pages 3922–3928. AAAI Press, 2017.

[143] W. Ping, Q. Liu, and A. T. Ihler. Decomposition bounds for marginal MAP. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems (NIPS-15)*, pages 3267–3275, 2015.

[144] K. Pipatsrisawat and A. Darwiche. New compilation languages based on structured decomposability. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI-08)*, pages 517–522. AAAI Press, 2008.

[145] K. Pipatsrisawat and A. Darwiche. A new d-DNNF-based bound computation algorithm for functional E-MAJSAT. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI-09)*, pages 590–595, 2009.

[146] D. Poole. Probabilistic Horn abduction and Bayesian networks. *Artificial Intelligence*, 64(1):81–129, 1993.

[147] D. Poole. Exploiting the rule structure for decision making within the independent choice logic. In *Proceedings of the 11th Annual Conference on Uncertainty in Artificial Intelligence (UAI-95)*, pages 454–463. Morgan Kaufmann, 1995.

[148] D. Poole. The independent choice logic and beyond. In *Probabilistic Inductive Logic Programming*, volume 4911 of *Lecture Notes in Computer Science*, pages 222–243. Springer, 2008.

[149] T. Rainforth, T. A. Le, J. van de Meent, M. A. Osborne, and F. D. Wood. Bayesian optimization for probabilistic programs. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems (NIPS-16)*, pages 280–288, 2016.

[150] A. Rendl, G. Tack, and P. J. Stuckey. Stochastic MiniZinc. In *Proceedings of the 20th International Conference on Principles and Practice of Constraint Programming (CP-14)*, volume 8656 of *Lecture Notes in Computer Science*, pages 636–645. Springer, 2014.

[151] J. Renkens, A. Kimmig, and L. De Raedt. Lazy explanation-based approximation for probabilistic logic programming. *CoRR*, abs/1507.02873, 2015.

[152] J. Renkens, A. Kimmig, G. Van den Broeck, and L. De Raedt. Explanation-based approximate weighted model counting for probabilistic logics. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI-14)*, pages 2490–2496. AAAI Press, 2014.

[153] S. Riedel. Improving the accuracy and efficiency of MAP inference for Markov logic. In *Proceedings of the 24th Conference in Uncertainty in Artificial Intelligence (UAI-08)*, pages 468–475. AUAI Press, 2008.

[154] F. Rossi, P. van Beek, and T. Walsh, editors. *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*. Elsevier, 2006.

[155] D. Roth. On the hardness of approximate reasoning. *Artificial Intelligence*, 82(1-2):273–302, 1996.

[156] R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *Proceedings of the 1993 IEEE/ACM International Conference on Computer-Aided Design (ICCAD-93)*, pages 42–47. IEEE, 1993.

[157] T. Sang, F. Bacchus, P. Beame, H. A. Kautz, and T. Pitassi. Combining component caching and clause learning for effective model counting. In *Online Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing (SAT-04)*, 2004.

[158] T. Sang, P. Beame, and H. A. Kautz. Performing Bayesian inference by weighted model counting. In *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI-05)*, pages 475–482. AAAI Press / The MIT Press, 2005.

[159] T. Sang, P. Beame, and H. A. Kautz. A dynamic approach for MPE and weighted MAX-SAT. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-07)*, pages 173–179, 2007.

[160] T. Sato. A statistical learning method for logic programs with distribution semantics. In *Proceedings of the 12th International Conference on Logic Programming (ICLP-95)*, pages 715–729. MIT Press, 1995.

[161] T. Sato and Y. Kameya. PRISM: A language for symbolic-statistical modeling. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 1330–1339. Morgan Kaufmann, 1997.

[162] T. Sato and Y. Kameya. Parameter learning of logic programs for symbolic-statistical modeling. *Journal of Artificial Intelligence Research*, 15:391–454, 2001.

[163] V. Satuluri, S. Parthasarathy, and Y. Ruan. Local graph sparsification for scalable clustering. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD-11)*, pages 721–732. ACM, 2011.

[164] P. Schaus, J. O. R. Aoga, and T. Guns. CoverSize: A global constraint for frequency-based itemset mining. In *Proceedings of the 23rd International Conference on Principles and Practice of Constraint Programming (CP-17)*, volume 10416 of *Lecture Notes in Computer Science*, pages 529–546. Springer, 2017.

[165] B. Selman and H. A. Kautz. Knowledge compilation and theory approximation. *Journal of the ACM*, 43(2):193–224, 1996.

[166] S. Sharma, S. Roy, M. Soos, and K. S. Meel. GANAK: A scalable probabilistic exact model counter. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI-19)*, pages 1169–1176. ijcai.org, 2019.

[167] S. E. Shimony. Finding MAPs for belief networks is NP-hard. *Artificial Intelligence*, 68(2):399–410, 1994.

[168] F. Somenzi. CUDD: CU Decision Diagram package-release 2.4.0, 2004. University of Colorado at Boulder.

[169] M. Soos and K. S. Meel. BIRD: engineering an efficient CNF-XOR SAT solver and its applications to approximate model counting. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI-19)*, pages 1592–1599. AAAI Press, 2019.

[170] D. Suciu, D. Olteanu, C. Ré, and C. Koch. *Probabilistic Databases*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011.

[171] A. Tarim, S. Manandhar, and T. Walsh. Stochastic constraint programming: A scenario-based approach. *Constraints*, 11(1):53–80, 2006.

[172] S. A. Tarim, B. Hnich, S. D. Prestwich, and R. Rossi. Finding reliable solutions: event-driven probabilistic constraint programming. *Annals of Operations Research*, 171(1):77–99, 2009.

[173] M. Thurley. sharpSAT — counting models with advanced component caching and implicit BCP. In *Proceedings of the 9th International Conference on Theory and Applications of Satisfiability Testing (SAT-06)*, volume 4121 of *Lecture Notes in Computer Science*, pages 424–429. Springer, 2006.

[174] L. Trujillo, E. Álvarez González, E. Galván, J. J. Tapia, and A. Ponsich. On the analysis of hyper-parameter space for a genetic programming system with iterated F-Race. *Soft computing (Berlin, Germany)*, 24(19):14757–14770, 2020.

[175] E. Tsamoura, V. Gutiérrez-Basulto, and A. Kimmig. Beyond the grounding bottleneck: Datalog techniques for inference in probabilistic logic programs. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI-20)*, pages 10284–10291. AAAI Press, 2020.

[176] L. G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8:189–201, 1979.

[177] L. G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979.

[178] G. Van den Broeck, I. Thon, M. van Otterlo, and L. De Raedt. DTProbLog: A decision-theoretic probabilistic Prolog. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI-10)*. AAAI Press, 2010.

[179] H. Verhaeghe, C. Lecoutre, and P. Schaus. Compact-MDD: Efficiently filtering (s)MDD constraints with reversible sparse bit-sets. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI-18)*, pages 1383–1389, 2018.

[180] B. Viswanath, A. Mislove, M. Cha, and K. Gummadi. On the evolution of user interaction in Facebook. In *Proceedings of the 2nd ACM Workshop on Online Social Networks (WOSN-09)*, pages 37–42, 2009.

[181] T. Walsh. Stochastic constraint programming. In *Proceedings of the 15th European Conference on Artificial Intelligence (ECAI-02)*, pages 111–115. IOS Press, 2002.

[182] W. Wei and B. Selman. A new approach to model counting. In *Proceedings of the 8th International Conference on Theory and Applications of Satisfiability Testing (SAT-05)*, volume 3569 of *Lecture Notes in Computer Science*, pages 324–339. Springer, 2005.

[183] B. Wiegmans. Gridkit: European and North-American extracts, 2016.

[184] Y. Xue, Z. Li, S. Ermon, C. P. Gomes, and B. Selman. Solving marginal MAP problems with NP oracles and parity constraints. In D. D. Lee, M. Sugiyama, U. von Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems (NIPS-16)*, pages 1127–1135, 2016.

[185] Y. Xue, X. Wu, D. Morin, B. Dilkina, A. Fuller, J. A. Royle, and C. P. Gomes. Dynamic optimization of landscape connectivity embedding spatial-capture-recapture information. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI-17)*, pages 4552–4558. AAAI Press, 2017.

[186] C. Yuan and E. A. Hansen. Efficient computation of jointree bounds for systematic MAP search. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI-09)*, pages 1982–1989, 2009.

[187] N. L. Zhang and D. Poole. A simple approach to Bayesian network computations. In *Proceedings of the 10th Canadian Conference on Artificial Intelligence*, 1994.

Appendices

A

Pseudocode of partial-sweep algorithm

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

Note that OscaR [132] uses *reversible data structures* that provide very convenient support for backtracking. We do not include any 'undo' operations for backtracking in our algorithm, as those mechanisms are already provided by the reversible data structures implemented in OscaR.

A.1 Notation and terminology

We use r to refer to a node in the *ordered binary decision diagram (OBDD)*, and r^- and r^+ to its lo and hi child, respectively. We use $var(r)$ to indicate *variable* that labels a node r, and we use $w(r)$ to indicate its weight in case $var(r)$ is a stochas-

tic variable. The path weight of r is denoted by $\pi(r)$, and its score according to Equation 2.11 by $s(r)$.

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

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

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

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

We think of propagation as the act of *removing* outgoing arcs of decision nodes when we fix the corresponding decision variable (recall Figure 6.2 in Section 6.4.3). Specifically, we remove an OBDD arc (p, c) from a parent p to child c if we fix $var(p)$ to *true* and c is p's lo child, or if we fix $var(p)$ to *false* and c is p's hi child (see Algorithm 9).

Through this process of removing arcs, we effectively remove *valid paths* (recall the definition of a valid path from Section 6.4.2) from the OBDD. Valid paths from OBDD roots to internal OBDD nodes, or to or from active decision nodes can determine whether or not we consider OBDD nodes to still be *relevant*, given the current partial strategy and corresponding removed arcs.

There are two ways in which a node r can be relevant. In the first case, it is a free decision node *and* it is reachable through a valid path from an OBDD root. In the second case, it is itself not a free decision node, but there is at least one valid path from a free decision node above r in the diagram down to r *and* there is a valid path from r itself down to a free decision node below it (see Algorithm 9).

In order to determine if a node is relevant, and to keep track of the part of the OBDD that is active (see Section 6.4.3), we associate three counters with each node r:

`Reachable[r]` Indicates the number of valid paths from the *artificial root* (see be-

low) of the OBDD down to r. The counter for this artificial root itself (and for the actual OBDD roots) is always 1.

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

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

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

The intuition behind the `Reachable` counter is the following: during search and propagation, assignments to decision variables may disconnect part of the OBDD from the root, because we remove arcs accordingly (**O6** in Section 6.4.3). This happens for example in Figure 6.2b.

The `FreeIn` counter of a node r has a value in the same domain as the `Reachable` counter, but represents a different concept. As addressed in **O4** in Section 6.4.3, only score changes in nodes that are descendants of free decision nodes, can influence the scores of those decision nodes. Therefore, during the bottom-up traversal of the OBDD to update the scores (Algorithm 4), we do not always need to propagate all the way to the roots. Once we encounter a node r whose score has changed due to recent value assignment to decision variables, but from which there is no valid path back to the artificial OBDD root that passes through a free decision node, we do not need to enqueue the parents of r for score updates. We keep track of this by counting how many of the incoming arcs of node r are on such a path.

The logic behind the `FreeOut` counter is similar to that of the `FreeIn` counter. However: instead of stopping an upward sweep, it serves to stop the downward sweep for path weight computation, to address **O5** in Section 6.4.3. The value of the `FreeOut` counter for a node r is either 0, 1 or 2, as it represent the number of children of the `FreeOut` counter that are on valid paths down to free decision nodes. Observe that if a node r is a fixed decision node, the value of its `FreeOut` counter can never exceed 1, as one of the outgoing arcs of r is removed by fixing the corresponding decision variable.

A.2 An SCPMD solving algorithm

Algorithm 1 shows the basic steps needed for solving an SCPMD in the *maximise expectation* setting (to which both problems described in Examples 4.2.1 and 4.2.2 belong).

Recall that these problems seek to maximise an expected score. We use the *stochastic constraint on monotonic distributions (SCMD)* for solving these problems by solving the constraint

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

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

A.3 Initialisation

Before the search for a solution to the stochastic constraint of Equation 1.1 begins, we initialise the data structures needed for enforcing the SCMD with the function INITIALISESCMD(OBDD, **D**), as given in Algorithm 2.

A.4 Partial-sweep propagation algorithm

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

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

Algorithm 1 Solving an *stochastic constraint (optimisation) problem on monotonic distributions (SCPMD)*, in the *maximise expectation* setting.

Input: an OBDD, a set of decision variables $\mathbf{D}$, a maximum cardinality k. These are all considered to be global variables.

Output: the optimal strategy σ^* and its corresponding score $s(\sigma^*)$.

1: **procedure** BRANCH(σ', D, a)
2: $\mathbf{D}_{\text{free}} \leftarrow \mathbf{D}_{\text{free}} \setminus \{D\}$
3: $F \leftarrow \{D\}$ ▷ The set of decision variables that are fixed in this call to the BRANCH function.
4: $\sigma' \leftarrow \sigma' \cup \{D = a\}$ ▷ Update partial strategy.
5: $(\text{conflict}, \sigma', F) \leftarrow$ PROPAGATESCMD(σ', F) ▷ See Algorithm 3.
6: **if** conflict **then return** and BACKTRACK **end if**
7: $(\text{conflict}, \sigma', F) \leftarrow$ PROPAGATECARDINALITYCONSTRAINT(σ', F) ▷ Assumed given, outside the scope of this work.
8: **if** conflict **then return** and BACKTRACK **end if**
9: SOLVE(σ')

10: **procedure** SOLVE(σ')
11: **if** $\mathbf{D}_{\text{free}} = \varnothing$ **and** $s(\sigma') > s^*$ **then**
12: $\sigma^* \leftarrow \sigma'$
13: $s^* \leftarrow s(\sigma^*)$ ▷ Score is computed incrementally (see Algorithm 3).
14: UPDATESCMDTHRESHOLD(s^*)
15: **return** and BACKTRACK
16: **for** $D \in \mathbf{D}_{\text{free}}$ **do** ▷ There are different selection strategies for determining which D to branch on next.
17: $a \leftarrow$ SELECTVALUE($dom(D)$) ▷ And different strategies for determining on which value to branch.
18: BRANCH(σ', D, a)
19: BRANCH(σ', D, $\overline{a}$)

20: INITIALISESCMD ▷ See Algorithm 2.
21: INITIALISECARDINALITYCONSTRAINT($\mathbf{D}$, k) ▷ Assumed given, outside the scope of this work.
22: $\mathbf{D}_{\text{free}} \leftarrow \mathbf{D}$ ▷ Set of free decision variables, global variable.
23: $\sigma^* \leftarrow \{D = \bot \mid D \in \mathbf{D}\}$, $s^* \leftarrow 0$ ▷ Optimal strategy and corresponding score, global variables.
24: $\sigma' \leftarrow$ ENFORCEDOMAINCONSISTENCY($\mathbf{D}_{free}$) ▷ Fix those variables that must be *true* to obtain partial strategy (Algorithm 6).
25: SOLVE(σ')
26: **return** σ^*, $s(\sigma^*)$

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

```
1:  procedure InitialiseFreeIn
2:      for r ∈ OBDD do FreeIn[r] ← 0 end for
3:      for r ∈ sorted(OBDD) do                          ▷ Downward sweep.
4:          if var(r) is decision OR FreeIn[r] > 0 then
5:              FreeIn[r⁻] ← FreeIn[r⁻] + 1
6:              FreeIn[r⁺] ← FreeIn[r⁺] + 1

7:  procedure InitialiseFreeOut
8:      for r ∈ OBDD do FreeOut[r] ← 0 end for
9:      for r ∈ reversed(sorted(OBDD)) do                ▷ Upward sweep.
10:         if var(r) is decision OR FreeOut[r] > 0 then
11:             for p ∈ parents(r) do FreeOut[p] ← FreeOut[p] + 1 end for

12: procedure InitialiseReachable
13:     for r ∈ OBDD do Reachable[r] ← 0 end for
14:     Reachable[root] ← 1
15:     for r ∈ sorted(OBDD) do                          ▷ Downward sweep.
16:         Reachable[r⁻] ← FreeIn[r⁻] + 1
17:         Reachable[r⁺] ← FreeIn[r⁺] + 1

18: procedure InitialiseScores
19:     for r ∈ reversed(sorted(OBDD)) do                ▷ Upward sweep.
20:         if var(r) is decision then
21:             s(r) ← s(r⁺)
22:         else
23:             s(r) ← w(r) · s(r⁺) + (1 − w(r)) · s(r⁻)
```

```
24: procedure INITIALISEPATHWEIGHTS
25:     for r ∈ OBDD do π(r) ← 0 end for
26:     for r ∈ SORTED(OBDD) do                          ▷ Downward sweep.
27:         if r is an original root of the OBDD then
28:             π(r) ← π(r) + ρ_r
29:         else
30:             for p ∈ PARENTS(r) do
31:                 if var(p) is decision then
32:                     if r is hi child of p then w ← 1 else w ← 0
33:                 else
34:                     if r is hi child of p then w ← w(p) else w ← (1 − w(p))
35:                 π(r) ← π(r) + π(p) · w

36: procedure INITIALISESCMD
37:     INITIALISEFREEIN
38:     INITIALISEFREEOUT
39:     INITIALISEREACHABLE
40:     INITIALISESCORES
41:     INITIALISEPATHWEIGHTS
42:     θ ← 0                                  ▷ The current best score to beat.
```

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

```
1: procedure PROPAGATESCMD(σ′, s_old, F)
2:     s ← s_old                          ▷ Score of previous partial strategy.
3:     δ ← UPDATESCORES(F)   ▷ δ is sum of derivatives of decision variables
   that were recently fixed to false, see also Algorithm 4.
4:     s ← s − δ                          ▷ score of current partial strategy σ′
5:     if s ≤ θ then return (true, σ′, F) end if   ▷ If we cannot satisfy the
   constraint, we must return and backtrack.
6:     UPDATEPATHWEIGHTS(F)                                  ▷ See Algorithm 5.
7:     (σ′, F) ← ENFORCEDOMAINCONSISTENCY(σ′, F, s)          ▷ See Algorithm 6.
8:     UPDATEREACHABLEFREEIN(F)                              ▷ See Algorithm 7.
9:     UPDATEFREEOUT(F)                                      ▷ See Algorithm 8.
10:    return (false, σ′, F)
```

Algorithm 4 Given a set F of decision variables that were recently fixed (either by branching or by propagation), update the node scores (using Equation 2.11 on page 39) that may have changed due to these new truth assignments. See Algorithm 9 for helper functions.

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

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

```
1:  procedure UPDATEPATHWEIGHTS(F)                          ▷ Downward sweep.
2:      Q ← ∅                                               ▷ Min heap (treat as set).
3:      for r ∈ {r | var(r) ∈ F ∧ Reachable[r] > 0} do
4:          if var(r) is false then
5:              Q.ENQUEUE(r⁻)
6:              Q.ENQUEUE(r⁺)
7:      while Q ≠ ∅ do
8:          r ← Q.DEQUEUE
9:          π_old ← π(r)
10:         if r is an original root of the OBDD then π_new ← ρ_r else π_new ← 0
    ▷ Roots have a path weight of ρ_r, which is the utility of the corresponding query.
11:         for p ∈ PARENTS(r) do
12:             if var(p) is decision variable then                ▷ r is a decision node
13:                 if ACTIVECHILD(p) = r then w ← 1 else w ← 0
14:             else                                               ▷ r is a stochastic node
15:                 if r is hi child of p then w ← w(p) else w ← (1 − w(p))
16:             π_new ← π_new + π(p) · w
17:         if π_new ≠ π_old then    ▷ We do not need to continue the propagation if the path weight has not changed.
18:             π(r) ← π_new
19:             if var(r) is stochastic variable then              ▷ r is a stochastic node.
20:                 ENQUEUERELEVANT(Q, r⁻)
21:                 ENQUEUERELEVANT(Q, r⁺)
22:             else                                               ▷ r is a decision node.
23:                 ENQUEUERELEVANT(Q, ACTIVECHILD(r))
```

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

1: **procedure** ENFORCEDOMAINCONSISTENCY(σ', F, s)
2: **for** $D \in \mathbf{D}_{free}$ **do**
3: $\Delta \leftarrow 0$ ▷ Partial derivative for free decision variable D.
4: **for** $r \in \{r \in \text{OBDD}_D \mid \texttt{Reachable[}r\texttt{]}\}$ **do**
5: $\Delta \leftarrow \Delta + \pi(r) \cdot (s(r^-) - s(r^+))$ ▷ Update the partial derivative for D.
6: **if** $s - \Delta \leq \theta$ **then** ▷ The current partial strategy cannot be extended to a valid solution if we fix D to *false*.
7: $\sigma' \leftarrow \sigma' \cup \{d = true\}$ ▷ Infer that D must be *true*.
8: $\mathbf{D}_{free} \leftarrow \mathbf{D}_{free} \setminus \{D\}$
9: $F \leftarrow F \cup \{D\}$
10: **return** (σ', F)

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

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

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

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

procedure UPDATEFREEOUT(V) ▷ Upward sweep.
  $\mathcal{U} \leftarrow \{r \mid var(r) \in V \wedge \texttt{Reachable}[r] > 0 \wedge \texttt{FreeIn}[r] > 0\}$ ▷ Max heap (treat as set).
  while $\mathcal{U} \neq \varnothing$ **do**
    $r \leftarrow \mathcal{U}$.DEQUEUE
    if $var(r) \in V$ **then**
      if $\texttt{FreeOut}[\text{ACTIVECHILD}(r)] > 0$ **then** $\texttt{FreeOut}[r] \leftarrow 1$ **else** $\texttt{FreeOut}[r] \leftarrow 0$
    if $\texttt{FreeOut}[r] = 0$ **and** ($var(r)$ is stochastic variable OR $var(r)$ is bound) **then**
      for $p \in$ PARENTS(r) **do**
        if not REMOVED(p, r) **and** RELEVANT(p) **then**
          $\texttt{FreeOut}[p] \leftarrow \texttt{FreeOut}[p] - 1$
          $\mathcal{U}$.ENQUEUE(p)

Algorithm 9 Helper functions for the update algorithms.

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

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

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

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

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

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

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

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

22: procedure ENQUEUERELEVANT(Q, r)
23:     if RELEVANT(r) then Q.ENQUEUE(r) end if
```

B

SIKS Dissertation Series

2011 01 Botond Cseke (RUN), Variational Algorithms for Bayesian Inference in Latent Gaussian Models

02 Nick Tinnemeier (UU), Organizing Agent Organizations. Syntax and Operational Semantics of an Organization-Oriented Programming Language

03 Jan Martijn van der Werf (TUE), Compositional Design and Verification of Component-Based Information Systems

04 Hado van Hasselt (UU), Insights in Reinforcement Learning; Formal analysis and empirical evaluation of temporal-difference

05 Bas van der Raadt (VU), Enterprise Architecture Coming of Age - Increasing the Performance of an Emerging Discipline.

06 Yiwen Wang (TUE), Semantically-Enhanced Recommendations in Cultural Heritage

07 Yujia Cao (UT), Multimodal Information Presentation for High Load Human Computer Interaction

08 Nieske Vergunst (UU), BDI-based Generation of Robust Task-Oriented Dialogues
09 Tim de Jong (OU), Contextualised Mobile Media for Learning
10 Bart Bogaert (UvT), Cloud Content Contention
11 Dhaval Vyas (UT), Designing for Awareness: An Experience-focused HCI Perspective
12 Carmen Bratosin (TUE), Grid Architecture for Distributed Process Mining
13 Xiaoyu Mao (UvT), Airport under Control. Multiagent Scheduling for Airport Ground Handling
14 Milan Lovric (EUR), Behavioral Finance and Agent-Based Artificial Markets
15 Marijn Koolen (UvA), The Meaning of Structure: the Value of Link Evidence for Information Retrieval
16 Maarten Schadd (UM), Selective Search in Games of Different Complexity
17 Jiyin He (UVA), Exploring Topic Structure: Coherence, Diversity and Relatedness
18 Mark Ponsen (UM), Strategic Decision-Making in complex games
19 Ellen Rusman (OU), The Mind's Eye on Personal Profiles
20 Qing Gu (VU), Guiding service-oriented software engineering - A view-based approach
21 Linda Terlouw (TUD), Modularization and Specification of Service-Oriented Systems
22 Junte Zhang (UVA), System Evaluation of Archival Description and Access
23 Wouter Weerkamp (UVA), Finding People and their Utterances in Social Media
24 Herwin van Welbergen (UT), Behavior Generation for Interpersonal Coordination with Virtual Humans On Specifying, Scheduling and Realizing Multimodal Virtual Human Behavior
25 Syed Waqar ul Qounain Jaffry (VU), Analysis and Validation of Models for Trust Dynamics
26 Matthijs Aart Pontier (VU), Virtual Agents for Human Communication - Emotion Regulation and Involvement-Distance Trade-Offs in Embodied Conversational Agents and Robots
27 Aniel Bhulai (VU), Dynamic website optimization through autonomous management of design patterns
28 Rianne Kaptein (UVA), Effective Focused Retrieval by Exploiting Query Context and Document Structure
29 Faisal Kamiran (TUE), Discrimination-aware Classification

30 Egon van den Broek (UT), Affective Signal Processing (ASP): Unraveling the mystery of emotions
31 Ludo Waltman (EUR), Computational and Game-Theoretic Approaches for Modeling Bounded Rationality
32 Nees-Jan van Eck (EUR), Methodological Advances in Bibliometric Mapping of Science
33 Tom van der Weide (UU), Arguing to Motivate Decisions
34 Paolo Turrini (UU), Strategic Reasoning in Interdependence: Logical and Game-theoretical Investigations
35 Maaike Harbers (UU), Explaining Agent Behavior in Virtual Training
36 Erik van der Spek (UU), Experiments in serious game design: a cognitive approach
37 Adriana Burlutiu (RUN), Machine Learning for Pairwise Data, Applications for Preference Learning and Supervised Network Inference
38 Nyree Lemmens (UM), Bee-inspired Distributed Optimization
39 Joost Westra (UU), Organizing Adaptation using Agents in Serious Games
40 Viktor Clerc (VU), Architectural Knowledge Management in Global Software Development
41 Luan Ibraimi (UT), Cryptographically Enforced Distributed Data Access Control
42 Michal Sindlar (UU), Explaining Behavior through Mental State Attribution
43 Henk van der Schuur (UU), Process Improvement through Software Operation Knowledge
44 Boris Reuderink (UT), Robust Brain-Computer Interfaces
45 Herman Stehouwer (UvT), Statistical Language Models for Alternative Sequence Selection
46 Beibei Hu (TUD), Towards Contextualized Information Delivery: A Rule-based Architecture for the Domain of Mobile Police Work
47 Azizi Bin Ab Aziz (VU), Exploring Computational Models for Intelligent Support of Persons with Depression
48 Mark Ter Maat (UT), Response Selection and Turn-taking for a Sensitive Artificial Listening Agent
49 Andreea Niculescu (UT), Conversational interfaces for task-oriented spoken dialogues: design aspects influencing interaction quality

2012 01 Terry Kakeeto (UvT), Relationship Marketing for SMEs in Uganda
02 Muhammad Umair (VU), Adaptivity, emotion, and Rationality in Human and Ambient Agent Models
03 Adam Vanya (VU), Supporting Architecture Evolution by Mining Software Repositories

04 Jurriaan Souer (UU), Development of Content Management System-based Web Applications
05 Marijn Plomp (UU), Maturing Interorganisational Information Systems
06 Wolfgang Reinhardt (OU), Awareness Support for Knowledge Workers in Research Networks
07 Rianne van Lambalgen (VU), When the Going Gets Tough: Exploring Agent-based Models of Human Performance under Demanding Conditions
08 Gerben de Vries (UVA), Kernel Methods for Vessel Trajectories
09 Ricardo Neisse (UT), Trust and Privacy Management Support for Context-Aware Service Platforms
10 David Smits (TUE), Towards a Generic Distributed Adaptive Hypermedia Environment
11 J.C.B. Rantham Prabhakara (TUE), Process Mining in the Large: Preprocessing, Discovery, and Diagnostics
12 Kees van der Sluijs (TUE), Model Driven Design and Data Integration in Semantic Web Information Systems
13 Suleman Shahid (UvT), Fun and Face: Exploring non-verbal expressions of emotion during playful interactions
14 Evgeny Knutov (TUE), Generic Adaptation Framework for Unifying Adaptive Web-based Systems
15 Natalie van der Wal (VU), Social Agents. Agent-Based Modelling of Integrated Internal and Social Dynamics of Cognitive and Affective Processes.
16 Fiemke Both (VU), Helping people by understanding them - Ambient Agents supporting task execution and depression treatment
17 Amal Elgammal (UvT), Towards a Comprehensive Framework for Business Process Compliance
18 Eltjo Poort (VU), Improving Solution Architecting Practices
19 Helen Schonenberg (TUE), What's Next? Operational Support for Business Process Execution
20 Ali Bahramisharif (RUN), Covert Visual Spatial Attention, a Robust Paradigm for Brain-Computer Interfacing
21 Roberto Cornacchia (TUD), Querying Sparse Matrices for Information Retrieval
22 Thijs Vis (UvT), Intelligence, politie en veiligheidsdienst: verenigbare grootheden?
23 Christian Muehl (UT), Toward Affective Brain-Computer Interfaces: Exploring the Neurophysiology of Affect during Human Media Interaction

24 Laurens van der Werff (UT), Evaluation of Noisy Transcripts for Spoken Document Retrieval
25 Silja Eckartz (UT), Managing the Business Case Development in Inter-Organizational IT Projects: A Methodology and its Application
26 Emile de Maat (UVA), Making Sense of Legal Text
27 Hayrettin Gurkok (UT), Mind the Sheep! User Experience Evaluation & Brain-Computer Interface Games
28 Nancy Pascall (UvT), Engendering Technology Empowering Women
29 Almer Tigelaar (UT), Peer-to-Peer Information Retrieval
30 Alina Pommeranz (TUD), Designing Human-Centered Systems for Reflective Decision Making
31 Emily Bagarukayo (RUN), A Learning by Construction Approach for Higher Order Cognitive Skills Improvement, Building Capacity and Infrastructure
32 Wietske Visser (TUD), Qualitative multi-criteria preference representation and reasoning
33 Rory Sie (OUN), Coalitions in Cooperation Networks (COCOON)
34 Pavol Jancura (RUN), Evolutionary analysis in PPI networks and applications
35 Evert Haasdijk (VU), Never Too Old To Learn – On-line Evolution of Controllers in Swarm- and Modular Robotics
36 Denis Ssebugwawo (RUN), Analysis and Evaluation of Collaborative Modeling Processes
37 Agnes Nakakawa (RUN), A Collaboration Process for Enterprise Architecture Creation
38 Selmar Smit (VU), Parameter Tuning and Scientific Testing in Evolutionary Algorithms
39 Hassan Fatemi (UT), Risk-aware design of value and coordination networks
40 Agus Gunawan (UvT), Information Access for SMEs in Indonesia
41 Sebastian Kelle (OU), Game Design Patterns for Learning
42 Dominique Verpoorten (OU), Reflection Amplifiers in self-regulated Learning
43 Withdrawn
44 Anna Tordai (VU), On Combining Alignment Techniques
45 Benedikt Kratz (UvT), A Model and Language for Business-aware Transactions
46 Simon Carter (UVA), Exploration and Exploitation of Multilingual Data for Statistical Machine Translation
47 Manos Tsagkias (UVA), Mining Social Media: Tracking Content and Predicting Behavior

48 Jorn Bakker (TUE), Handling Abrupt Changes in Evolving Time-series Data
49 Michael Kaisers (UM), Learning against Learning - Evolutionary dynamics of reinforcement learning algorithms in strategic interactions
50 Steven van Kervel (TUD), Ontologogy driven Enterprise Information Systems Engineering
51 Jeroen de Jong (TUD), Heuristics in Dynamic Sceduling; a practical framework with a case study in elevator dispatching

2013 01 Viorel Milea (EUR), News Analytics for Financial Decision Support
02 Erietta Liarou (CWI), MonetDB/DataCell: Leveraging the Column-store Database Technology for Efficient and Scalable Stream Processing
03 Szymon Klarman (VU), Reasoning with Contexts in Description Logics
04 Chetan Yadati (TUD), Coordinating autonomous planning and scheduling
05 Dulce Pumareja (UT), Groupware Requirements Evolutions Patterns
06 Romulo Goncalves (CWI), The Data Cyclotron: Juggling Data and Queries for a Data Warehouse Audience
07 Giel van Lankveld (UvT), Quantifying Individual Player Differences
08 Robbert-Jan Merk (VU), Making enemies: cognitive modeling for opponent agents in fighter pilot simulators
09 Fabio Gori (RUN), Metagenomic Data Analysis: Computational Methods and Applications
10 Jeewanie Jayasinghe Arachchige (UvT), A Unified Modeling Framework for Service Design.
11 Evangelos Pournaras (TUD), Multi-level Reconfigurable Self-organization in Overlay Services
12 Marian Razavian (VU), Knowledge-driven Migration to Services
13 Mohammad Safiri (UT), Service Tailoring: User-centric creation of integrated IT-based homecare services to support independent living of elderly
14 Jafar Tanha (UVA), Ensemble Approaches to Semi-Supervised Learning Learning
15 Daniel Hennes (UM), Multiagent Learning - Dynamic Games and Applications
16 Eric Kok (UU), Exploring the practical benefits of argumentation in multi-agent deliberation
17 Koen Kok (VU), The PowerMatcher: Smart Coordination for the Smart Electricity Grid
18 Jeroen Janssens (UvT), Outlier Selection and One-Class Classification

19 Renze Steenhuizen (TUD), Coordinated Multi-Agent Planning and Scheduling
20 Katja Hofmann (UvA), Fast and Reliable Online Learning to Rank for Information Retrieval
21 Sander Wubben (UvT), Text-to-text generation by monolingual machine translation
22 Tom Claassen (RUN), Causal Discovery and Logic
23 Patricio de Alencar Silva (UvT), Value Activity Monitoring
24 Haitham Bou Ammar (UM), Automated Transfer in Reinforcement Learning
25 Agnieszka Anna Latoszek-Berendsen (UM), Intention-based Decision Support. A new way of representing and implementing clinical guidelines in a Decision Support System
26 Alireza Zarghami (UT), Architectural Support for Dynamic Homecare Service Provisioning
27 Mohammad Huq (UT), Inference-based Framework Managing Data Provenance
28 Frans van der Sluis (UT), When Complexity becomes Interesting: An Inquiry into the Information eXperience
29 Iwan de Kok (UT), Listening Heads
30 Joyce Nakatumba (TUE), Resource-Aware Business Process Management: Analysis and Support
31 Dinh Khoa Nguyen (UvT), Blueprint Model and Language for Engineering Cloud Applications
32 Kamakshi Rajagopal (OUN), Networking For Learning; The role of Networking in a Lifelong Learner's Professional Development
33 Qi Gao (TUD), User Modeling and Personalization in the Microblogging Sphere
34 Kien Tjin-Kam-Jet (UT), Distributed Deep Web Search
35 Abdallah El Ali (UvA), Minimal Mobile Human Computer Interaction
36 Than Lam Hoang (TUe), Pattern Mining in Data Streams
37 Dirk Börner (OUN), Ambient Learning Displays
38 Eelco den Heijer (VU), Autonomous Evolutionary Art
39 Joop de Jong (TUD), A Method for Enterprise Ontology based Design of Enterprise Information Systems
40 Pim Nijssen (UM), Monte-Carlo Tree Search for Multi-Player Games
41 Jochem Liem (UVA), Supporting the Conceptual Modelling of Dynamic Systems: A Knowledge Engineering Perspective on Qualitative Reasoning
42 Léon Planken (TUD), Algorithms for Simple Temporal Reasoning

43 Marc Bron (UVA), Exploration and Contextualization through Interaction and Concepts

2014 01 Nicola Barile (UU), Studies in Learning Monotone Models from Data
02 Fiona Tuliyano (RUN), Combining System Dynamics with a Domain Modeling Method
03 Sergio Raul Duarte Torres (UT), Information Retrieval for Children: Search Behavior and Solutions
04 Hanna Jochmann-Mannak (UT), Websites for children: search strategies and interface design - Three studies on children's search performance and evaluation
05 Jurriaan van Reijsen (UU), Knowledge Perspectives on Advancing Dynamic Capability
06 Damian Tamburri (VU), Supporting Networked Software Development
07 Arya Adriansyah (TUE), Aligning Observed and Modeled Behavior
08 Samur Araujo (TUD), Data Integration over Distributed and Heterogeneous Data Endpoints
09 Philip Jackson (UvT), Toward Human-Level Artificial Intelligence: Representation and Computation of Meaning in Natural Language
10 Ivan Salvador Razo Zapata (VU), Service Value Networks
11 Janneke van der Zwaan (TUD), An Empathic Virtual Buddy for Social Support
12 Willem van Willigen (VU), Look Ma, No Hands: Aspects of Autonomous Vehicle Control
13 Arlette van Wissen (VU), Agent-Based Support for Behavior Change: Models and Applications in Health and Safety Domains
14 Yangyang Shi (TUD), Language Models With Meta-information
15 Natalya Mogles (VU), Agent-Based Analysis and Support of Human Functioning in Complex Socio-Technical Systems: Applications in Safety and Healthcare
16 Krystyna Milian (VU), Supporting trial recruitment and design by automatically interpreting eligibility criteria
17 Kathrin Dentler (VU), Computing healthcare quality indicators automatically: Secondary Use of Patient Data and Semantic Interoperability
18 Mattijs Ghijsen (UVA), Methods and Models for the Design and Study of Dynamic Agent Organizations
19 Vinicius Ramos (TUE), Adaptive Hypermedia Courses: Qualitative and Quantitative Evaluation and Tool Support
20 Mena Habib (UT), Named Entity Extraction and Disambiguation for Informal Text: The Missing Link

21 Kassidy Clark (TUD), Negotiation and Monitoring in Open Environments
22 Marieke Peeters (UU), Personalized Educational Games - Developing agent-supported scenario-based training
23 Eleftherios Sidirourgos (UvA/CWI), Space Efficient Indexes for the Big Data Era
24 Davide Ceolin (VU), Trusting Semi-structured Web Data
25 Martijn Lappenschaar (RUN), New network models for the analysis of disease interaction
26 Tim Baarslag (TUD), What to Bid and When to Stop
27 Rui Jorge Almeida (EUR), Conditional Density Models Integrating Fuzzy and Probabilistic Representations of Uncertainty
28 Anna Chmielowiec (VU), Decentralized k-Clique Matching
29 Jaap Kabbedijk (UU), Variability in Multi-Tenant Enterprise Software
30 Peter de Cock (UvT), Anticipating Criminal Behaviour
31 Leo van Moergestel (UU), Agent Technology in Agile Multiparallel Manufacturing and Product Support
32 Naser Ayat (UvA), On Entity Resolution in Probabilistic Data
33 Tesfa Tegegne (RUN), Service Discovery in eHealth
34 Christina Manteli (VU), The Effect of Governance in Global Software Development: Analyzing Transactive Memory Systems.
35 Joost van Ooijen (UU), Cognitive Agents in Virtual Worlds: A Middleware Design Approach
36 Joos Buijs (TUE), Flexible Evolutionary Algorithms for Mining Structured Process Models
37 Maral Dadvar (UT), Experts and Machines United Against Cyberbullying
38 Danny Plass-Oude Bos (UT), Making brain-computer interfaces better: improving usability through post-processing.
39 Jasmina Maric (UvT), Web Communities, Immigration, and Social Capital
40 Walter Omona (RUN), A Framework for Knowledge Management Using ICT in Higher Education
41 Frederic Hogenboom (EUR), Automated Detection of Financial Events in News Text
42 Carsten Eijckhof (CWI/TUD), Contextual Multidimensional Relevance Models
43 Kevin Vlaanderen (UU), Supporting Process Improvement using Method Increments
44 Paulien Meesters (UvT), Intelligent Blauw. Met als ondertitel: Intelligence-gestuurde politiezorg in gebiedsgebonden eenheden.

45 Birgit Schmitz (OUN), Mobile Games for Learning: A Pattern-Based Approach
46 Ke Tao (TUD), Social Web Data Analytics: Relevance, Redundancy, Diversity
47 Shangsong Liang (UVA), Fusion and Diversification in Information Retrieval

2015 01 Niels Netten (UvA), Machine Learning for Relevance of Information in Crisis Response
02 Faiza Bukhsh (UvT), Smart auditing: Innovative Compliance Checking in Customs Controls
03 Twan van Laarhoven (RUN), Machine learning for network data
04 Howard Spoelstra (OUN), Collaborations in Open Learning Environments
05 Christoph Bösch (UT), Cryptographically Enforced Search Pattern Hiding
06 Farideh Heidari (TUD), Business Process Quality Computation - Computing Non-Functional Requirements to Improve Business Processes
07 Maria-Hendrike Peetz (UvA), Time-Aware Online Reputation Analysis
08 Jie Jiang (TUD), Organizational Compliance: An agent-based model for designing and evaluating organizational interactions
09 Randy Klaassen (UT), HCI Perspectives on Behavior Change Support Systems
10 Henry Hermans (OUN), OpenU: design of an integrated system to support lifelong learning
11 Yongming Luo (TUE), Designing algorithms for big graph datasets: A study of computing bisimulation and joins
12 Julie M. Birkholz (VU), Modi Operandi of Social Network Dynamics: The Effect of Context on Scientific Collaboration Networks
13 Giuseppe Procaccianti (VU), Energy-Efficient Software
14 Bart van Straalen (UT), A cognitive approach to modeling bad news conversations
15 Klaas Andries de Graaf (VU), Ontology-based Software Architecture Documentation
16 Changyun Wei (UT), Cognitive Coordination for Cooperative Multi-Robot Teamwork
17 André van Cleeff (UT), Physical and Digital Security Mechanisms: Properties, Combinations and Trade-offs
18 Holger Pirk (CWI), Waste Not, Want Not! - Managing Relational Data in Asymmetric Memories
19 Bernardo Tabuenca (OUN), Ubiquitous Technology for Lifelong Learners

20 Lois Vanhée (UU), Using Culture and Values to Support Flexible Coordination
21 Sibren Fetter (OUN), Using Peer-Support to Expand and Stabilize Online Learning
22 Zhemin Zhu (UT), Co-occurrence Rate Networks
23 Luit Gazendam (VU), Cataloguer Support in Cultural Heritage
24 Richard Berendsen (UVA), Finding People, Papers, and Posts: Vertical Search Algorithms and Evaluation
25 Steven Woudenberg (UU), Bayesian Tools for Early Disease Detection
26 Alexander Hogenboom (EUR), Sentiment Analysis of Text Guided by Semantics and Structure
27 Sándor Héman (CWI), Updating compressed colomn stores
28 Janet Bagorogoza (TiU), Knowledge Management and High Performance; The Uganda Financial Institutions Model for HPO
29 Hendrik Baier (UM), Monte-Carlo Tree Search Enhancements for One-Player and Two-Player Domains
30 Kiavash Bahreini (OU), Real-time Multimodal Emotion Recognition in E-Learning
31 Yakup Koç (TUD), On the robustness of Power Grids
32 Jerome Gard (UL), Corporate Venture Management in SMEs
33 Frederik Schadd (TUD), Ontology Mapping with Auxiliary Resources
34 Victor de Graaf (UT), Gesocial Recommender Systems
35 Jungxao Xu (TUD), Affective Body Language of Humanoid Robots: Perception and Effects in Human Robot Interaction

2016 01 Syed Saiden Abbas (RUN), Recognition of Shapes by Humans and Machines
02 Michiel Christiaan Meulendijk (UU), Optimizing medication reviews through decision support: prescribing a better pill to swallow
03 Maya Sappelli (RUN), Knowledge Work in Context: User Centered Knowledge Worker Support
04 Laurens Rietveld (VU), Publishing and Consuming Linked Data
05 Evgeny Sherkhonov (UVA), Expanded Acyclic Queries: Containment and an Application in Explaining Missing Answers
06 Michel Wilson (TUD), Robust scheduling in an uncertain environment
07 Jeroen de Man (VU), Measuring and modeling negative emotions for virtual training
08 Matje van de Camp (TiU), A Link to the Past: Constructing Historical Social Networks from Unstructured Data
09 Archana Nottamkandath (VU), Trusting Crowdsourced Information on Cultural Artefacts

10 George Karafotias (VUA), Parameter Control for Evolutionary Algorithms
11 Anne Schuth (UVA), Search Engines that Learn from Their Users
12 Max Knobbout (UU), Logics for Modelling and Verifying Normative Multi-Agent Systems
13 Nana Baah Gyan (VU), The Web, Speech Technologies and Rural Development in West Africa - An ICT4D Approach
14 Ravi Khadka (UU), Revisiting Legacy Software System Modernization
15 Steffen Michels (RUN), Hybrid Probabilistic Logics - Theoretical Aspects, Algorithms and Experiments
16 Guangliang Li (UVA), Socially Intelligent Autonomous Agents that Learn from Human Reward
17 Berend Weel (VU), Towards Embodied Evolution of Robot Organisms
18 Albert Meroño Peñuela (VU), Refining Statistical Data on the Web
19 Julia Efremova (Tu/e), Mining Social Structures from Genealogical Data
20 Daan Odijk (UVA), Context & Semantics in News & Web Search
21 Alejandro Moreno Célleri (UT), From Traditional to Interactive Playspaces: Automatic Analysis of Player Behavior in the Interactive Tag Playground
22 Grace Lewis (VU), Software Architecture Strategies for Cyber-Foraging Systems
23 Fei Cai (UVA), Query Auto Completion in Information Retrieval
24 Brend Wanders (UT), Repurposing and Probabilistic Integration of Data; An Iterative and data model independent approach
25 Julia Kiseleva (TU/e), Using Contextual Information to Understand Searching and Browsing Behavior
26 Dilhan Thilakarathne (VU), In or Out of Control: Exploring Computational Models to Study the Role of Human Awareness and Control in Behavioural Choices, with Applications in Aviation and Energy Management Domains
27 Wen Li (TUD), Understanding Geo-spatial Information on Social Media
28 Mingxin Zhang (TUD), Large-scale Agent-based Social Simulation - A study on epidemic prediction and control
29 Nicolas Höning (TUD), Peak reduction in decentralised electricity systems - Markets and prices for flexible planning
30 Ruud Mattheij (UvT), The Eyes Have It
31 Mohammad Khelghati (UT), Deep web content monitoring
32 Eelco Vriezekolk (UT), Assessing Telecommunication Service Availability Risks for Crisis Organisations

33 Peter Bloem (UVA), Single Sample Statistics, exercises in learning from just one example
34 Dennis Schunselaar (TUE), Configurable Process Trees: Elicitation, Analysis, and Enactment
35 Zhaochun Ren (UVA), Monitoring Social Media: Summarization, Classification and Recommendation
36 Daphne Karreman (UT), Beyond R2D2: The design of nonverbal interaction behavior optimized for robot-specific morphologies
37 Giovanni Sileno (UvA), Aligning Law and Action - a conceptual and computational inquiry
38 Andrea Minuto (UT), Materials that Matter - Smart Materials meet Art & Interaction Design
39 Merijn Bruijnes (UT), Believable Suspect Agents; Response and Interpersonal Style Selection for an Artificial Suspect
40 Christian Detweiler (TUD), Accounting for Values in Design
41 Thomas King (TUD), Governing Governance: A Formal Framework for Analysing Institutional Design and Enactment Governance
42 Spyros Martzoukos (UVA), Combinatorial and Compositional Aspects of Bilingual Aligned Corpora
43 Saskia Koldijk (RUN), Context-Aware Support for Stress Self-Management: From Theory to Practice
44 Thibault Sellam (UVA), Automatic Assistants for Database Exploration
45 Bram van de Laar (UT), Experiencing Brain-Computer Interface Control
46 Jorge Gallego Perez (UT), Robots to Make you Happy
47 Christina Weber (UL), Real-time foresight - Preparedness for dynamic innovation networks
48 Tanja Buttler (TUD), Collecting Lessons Learned
49 Gleb Polevoy (TUD), Participation and Interaction in Projects. A Game-Theoretic Analysis
50 Yan Wang (UVT), The Bridge of Dreams: Towards a Method for Operational Performance Alignment in IT-enabled Service Supply Chains

2017 01 Jan-Jaap Oerlemans (UL), Investigating Cybercrime
02 Sjoerd Timmer (UU), Designing and Understanding Forensic Bayesian Networks using Argumentation
03 Daniël Harold Telgen (UU), Grid Manufacturing; A Cyber-Physical Approach with Autonomous Products and Reconfigurable Manufacturing Machines
04 Mrunal Gawade (CWI), Multi-core Parallelism in a Column-store
05 Mahdieh Shadi (UVA), Collaboration Behavior

06 Damir Vandic (EUR), Intelligent Information Systems for Web Product Search
07 Roel Bertens (UU), Insight in Information: from Abstract to Anomaly
08 Rob Konijn (VU) , Detecting Interesting Differences:Data Mining in Health Insurance Data using Outlier Detection and Subgroup Discovery
09 Dong Nguyen (UT), Text as Social and Cultural Data: A Computational Perspective on Variation in Text
10 Robby van Delden (UT), (Steering) Interactive Play Behavior
11 Florian Kunneman (RUN), Modelling patterns of time and emotion in Twitter #anticipointment
12 Sander Leemans (TUE), Robust Process Mining with Guarantees
13 Gijs Huisman (UT), Social Touch Technology - Extending the reach of social touch through haptic technology
14 Shoshannah Tekofsky (UvT), You Are Who You Play You Are: Modelling Player Traits from Video Game Behavior
15 Peter Berck (RUN), Memory-Based Text Correction
16 Aleksandr Chuklin (UVA), Understanding and Modeling Users of Modern Search Engines
17 Daniel Dimov (UL), Crowdsourced Online Dispute Resolution
18 Ridho Reinanda (UVA), Entity Associations for Search
19 Jeroen Vuurens (UT), Proximity of Terms, Texts and Semantic Vectors in Information Retrieval
20 Mohammadbashir Sedighi (TUD), Fostering Engagement in Knowledge Sharing: The Role of Perceived Benefits, Costs and Visibility
21 Jeroen Linssen (UT), Meta Matters in Interactive Storytelling and Serious Gaming (A Play on Worlds)
22 Sara Magliacane (VU), Logics for causal inference under uncertainty
23 David Graus (UVA), Entities of Interest — Discovery in Digital Traces
24 Chang Wang (TUD), Use of Affordances for Efficient Robot Learning
25 Veruska Zamborlini (VU), Knowledge Representation for Clinical Guidelines, with applications to Multimorbidity Analysis and Literature Search
26 Merel Jung (UT), Socially intelligent robots that understand and respond to human touch
27 Michiel Joosse (UT), Investigating Positioning and Gaze Behaviors of Social Robots: People's Preferences, Perceptions and Behaviors
28 John Klein (VU), Architecture Practices for Complex Contexts

29 Adel Alhuraibi (UvT), From IT-BusinessStrategic Alignment to Performance: A Moderated Mediation Model of Social Innovation, and Enterprise Governance of IT"
30 Wilma Latuny (UvT), The Power of Facial Expressions
31 Ben Ruijl (UL), Advances in computational methods for QFT calculations
32 Thaer Samar (RUN), Access to and Retrievability of Content in Web Archives
33 Brigit van Loggem (OU), Towards a Design Rationale for Software Documentation: A Model of Computer-Mediated Activity
34 Maren Scheffel (OU), The Evaluation Framework for Learning Analytics
35 Martine de Vos (VU), Interpreting natural science spreadsheets
36 Yuanhao Guo (UL), Shape Analysis for Phenotype Characterisation from High-throughput Imaging
37 Alejandro Montes Garcia (TUE), WiBAF: A Within Browser Adaptation Framework that Enables Control over Privacy
38 Alex Kayal (TUD), Normative Social Applications
39 Sara Ahmadi (RUN), Exploiting properties of the human auditory system and compressive sensing methods to increase noise robustness in ASR
40 Altaf Hussain Abro (VUA), Steer your Mind: Computational Exploration of Human Control in Relation to Emotions, Desires and Social Support For applications in human-aware support systems
41 Adnan Manzoor (VUA), Minding a Healthy Lifestyle: An Exploration of Mental Processes and a Smart Environment to Provide Support for a Healthy Lifestyle
42 Elena Sokolova (RUN), Causal discovery from mixed and missing data with applications on ADHD datasets
43 Maaike de Boer (RUN), Semantic Mapping in Video Retrieval
44 Garm Lucassen (UU), Understanding User Stories - Computational Linguistics in Agile Requirements Engineering
45 Bas Testerink (UU), Decentralized Runtime Norm Enforcement
46 Jan Schneider (OU), Sensor-based Learning Support
47 Jie Yang (TUD), Crowd Knowledge Creation Acceleration
48 Angel Suarez (OU), Collaborative inquiry-based learning

2018 01 Han van der Aa (VUA), Comparing and Aligning Process Representations
02 Felix Mannhardt (TUE), Multi-perspective Process Mining

03 Steven Bosems (UT), Causal Models For Well-Being: Knowledge Modeling, Model-Driven Development of Context-Aware Applications, and Behavior Prediction
04 Jordan Janeiro (TUD), Flexible Coordination Support for Diagnosis Teams in Data-Centric Engineering Tasks
05 Hugo Huurdeman (UVA), Supporting the Complex Dynamics of the Information Seeking Process
06 Dan Ionita (UT), Model-Driven Information Security Risk Assessment of Socio-Technical Systems
07 Jieting Luo (UU), A formal account of opportunism in multi-agent systems
08 Rick Smetsers (RUN), Advances in Model Learning for Software Systems
09 Xu Xie (TUD), Data Assimilation in Discrete Event Simulations
10 Julienka Mollee (VUA), Moving forward: supporting physical activity behavior change through intelligent technology
11 Mahdi Sargolzaei (UVA), Enabling Framework for Service-oriented Collaborative Networks
12 Xixi Lu (TUE), Using behavioral context in process mining
13 Seyed Amin Tabatabaei (VUA), Computing a Sustainable Future
14 Bart Joosten (UVT), Detecting Social Signals with Spatiotemporal Gabor Filters
15 Naser Davarzani (UM), Biomarker discovery in heart failure
16 Jaebok Kim (UT), Automatic recognition of engagement and emotion in a group of children
17 Jianpeng Zhang (TUE), On Graph Sample Clustering
18 Henriette Nakad (UL), De Notaris en Private Rechtspraak
19 Minh Duc Pham (VUA), Emergent relational schemas for RDF
20 Manxia Liu (RUN), Time and Bayesian Networks
21 Aad Slootmaker (OUN), EMERGO: a generic platform for authoring and playing scenario-based serious games
22 Eric Fernandes de Mello Araujo (VUA), Contagious: Modeling the Spread of Behaviours, Perceptions and Emotions in Social Networks
23 Kim Schouten (EUR), Semantics-driven Aspect-Based Sentiment Analysis
24 Jered Vroon (UT), Responsive Social Positioning Behaviour for Semi-Autonomous Telepresence Robots
25 Riste Gligorov (VUA), Serious Games in Audio-Visual Collections
26 Roelof Anne Jelle de Vries (UT),Theory-Based and Tailor-Made: Motivational Messages for Behavior Change Technology

27 Maikel Leemans (TUE), Hierarchical Process Mining for Scalable Software Analysis
28 Christian Willemse (UT), Social Touch Technologies: How they feel and how they make you feel
29 Yu Gu (UVT), Emotion Recognition from Mandarin Speech
30 Wouter Beek, The "K" in "semantic web" stands for "knowledge": scaling semantics to the web

2019 01 Rob van Eijk (UL),Web privacy measurement in real-time bidding systems. A graph-based approach to RTB system classification
02 Emmanuelle Beauxis Aussalet (CWI, UU), Statistics and Visualizations for Assessing Class Size Uncertainty
03 Eduardo Gonzalez Lopez de Murillas (TUE), Process Mining on Databases: Extracting Event Data from Real Life Data Sources
04 Ridho Rahmadi (RUN), Finding stable causal structures from clinical data
05 Sebastiaan van Zelst (TUE), Process Mining with Streaming Data
06 Chris Dijkshoorn (VU), Nichesourcing for Improving Access to Linked Cultural Heritage Datasets
07 Soude Fazeli (TUD), Recommender Systems in Social Learning Platforms
08 Frits de Nijs (TUD), Resource-constrained Multi-agent Markov Decision Processes
09 Fahimeh Alizadeh Moghaddam (UVA), Self-adaptation for energy efficiency in software systems
10 Qing Chuan Ye (EUR), Multi-objective Optimization Methods for Allocation and Prediction
11 Yue Zhao (TUD), Learning Analytics Technology to Understand Learner Behavioral Engagement in MOOCs
12 Jacqueline Heinerman (VU), Better Together
13 Guanliang Chen (TUD), MOOC Analytics: Learner Modeling and Content Generation
14 Daniel Davis (TUD), Large-Scale Learning Analytics: Modeling Learner Behavior & Improving Learning Outcomes in Massive Open Online Courses
15 Erwin Walraven (TUD), Planning under Uncertainty in Constrained and Partially Observable Environments
16 Guangming Li (TUE), Process Mining based on Object-Centric Behavioral Constraint (OCBC) Models
17 Ali Hurriyetoglu (RUN),Extracting actionable information from microtexts
18 Gerard Wagenaar (UU), Artefacts in Agile Team Communication

19 Vincent Koeman (TUD), Tools for Developing Cognitive Agents
20 Chide Groenouwe (UU), Fostering technically augmented human collective intelligence
21 Cong Liu (TUE), Software Data Analytics: Architectural Model Discovery and Design Pattern Detection
22 Martin van den Berg (VU),Improving IT Decisions with Enterprise Architecture
23 Qin Liu (TUD), Intelligent Control Systems: Learning, Interpreting, Verification
24 Anca Dumitrache (VU), Truth in Disagreement - Crowdsourcing Labeled Data for Natural Language Processing
25 Emiel van Miltenburg (VU), Pragmatic factors in (automatic) image description
26 Prince Singh (UT), An Integration Platform for Synchromodal Transport
27 Alessandra Antonaci (OUN), The Gamification Design Process applied to (Massive) Open Online Courses
28 Esther Kuindersma (UL), Cleared for take-off: Game-based learning to prepare airline pilots for critical situations
29 Daniel Formolo (VU), Using virtual agents for simulation and training of social skills in safety-critical circumstances
30 Vahid Yazdanpanah (UT), Multiagent Industrial Symbiosis Systems
31 Milan Jelisavcic (VU), Alive and Kicking: Baby Steps in Robotics
32 Chiara Sironi (UM), Monte-Carlo Tree Search for Artificial General Intelligence in Games
33 Anil Yaman (TUE), Evolution of Biologically Inspired Learning in Artificial Neural Networks
34 Negar Ahmadi (TUE), EEG Microstate and Functional Brain Network Features for Classification of Epilepsy and PNES
35 Lisa Facey-Shaw (OUN), Gamification with digital badges in learning programming
36 Kevin Ackermans (OUN), Designing Video-Enhanced Rubrics to Master Complex Skills
37 Jian Fang (TUD), Database Acceleration on FPGAs
38 Akos Kadar (OUN), Learning visually grounded and multilingual representations

2020 01 Armon Toubman (UL), Calculated Moves: Generating Air Combat Behaviour
02 Marcos de Paula Bueno (UL), Unraveling Temporal Processes using Probabilistic Graphical Models

03 Mostafa Deghani (UvA), Learning with Imperfect Supervision for Language Understanding
04 Maarten van Gompel (RUN), Context as Linguistic Bridges
05 Yulong Pei (TUE), On local and global structure mining
06 Preethu Rose Anish (UT), Stimulation Architectural Thinking during Requirements Elicitation - An Approach and Tool Support
07 Wim van der Vegt (OUN), Towards a software architecture for reusable game components
08 Ali Mirsoleimani (UL),Structured Parallel Programming for Monte Carlo Tree Search
09 Myriam Traub (UU), Measuring Tool Bias and Improving Data Quality for Digital Humanities Research
10 Alifah Syamsiyah (TUE), In-database Preprocessing for Process Mining
11 Sepideh Mesbah (TUD), Semantic-Enhanced Training Data AugmentationMethods for Long-Tail Entity Recognition Models
12 Ward van Breda (VU), Predictive Modeling in E-Mental Health: Exploring Applicability in Personalised Depression Treatment
13 Marco Virgolin (CWI), Design and Application of Gene-pool Optimal Mixing Evolutionary Algorithms for Genetic Programming
14 Mark Raasveldt (CWI/UL), Integrating Analytics with Relational Databases
15 Konstantinos Georgiadis (OUN), Smart CAT: Machine Learning for Configurable Assessments in Serious Games
16 Ilona Wilmont (RUN), Cognitive Aspects of Conceptual Modelling
17 Daniele Di Mitri (OUN), The Multimodal Tutor: Adaptive Feedback from Multimodal Experiences
18 Georgios Methenitis (TUD), Agent Interactions & Mechanisms in Markets with Uncertainties: Electricity Markets in Renewable Energy Systems
19 Guido van Capelleveen (UT), Industrial Symbiosis Recommender Systems
20 Albert Hankel (VU), Embedding Green ICT Maturity in Organisations
21 Karine da Silva Miras de Araujo (VU), Where is the robot?: Life as it could be
22 Maryam Masoud Khamis (RUN), Understanding complex systems implementation through a modeling approach: the case of e-government in Zanzibar
23 Rianne Conijn (UT), The Keys to Writing: A writing analytics approach to studying writing processes using keystroke logging
24 Lenin da Nobrega Medeiros (VUA/RUN), How are you feeling, human? Towards emotionally supportive chatbots

25 Xin Du (TUE), The Uncertainty in Exceptional Model Mining
26 Krzysztof Leszek Sadowski (UU), GAMBIT: Genetic Algorithm for Model-Based mixed-Integer opTimization
27 Ekaterina Muravyeva (TUD), Personal data and informed consent in an educational context
28 Bibeg Limbu (TUD), Multimodal interaction for deliberate practice: Training complex skills with augmented reality
29 Ioan Gabriel Bucur (RUN), Being Bayesian about Causal Inference
30 Bob Zadok Blok (UL), Creatief, Creatieve, Creatiefst
31 Gongjin Lan (VU), Learning better – From Baby to Better
32 Jason Rhuggenaath (TUE), Revenue management in online markets: pricing and online advertising
33 Rick Gilsing (TUE), Supporting service-dominant business model evaluation in the context of business model innovation
34 Anna Bon (MU), Intervention or Collaboration? Redesigning Information and Communication Technologies for Development
35 Siamak Farshidi (UU), Multi-Criteria Decision-Making in Software Production

2021 01 Francisco Xavier Dos Santos Fonseca (TUD), Location-based Games for Social Interaction in Public Space
02 Rijk Mercuur (TUD), Simulating Human Routines:Integrating Social Practice Theory in Agent-Based Models
03 Seyyed Hadi Hashemi (UVA), Modeling Users Interacting with Smart Devices
04 Ioana Jivet (OU), The Dashboard That Loved Me: Designing adaptive learning analytics for self-regulated learning
05 Davide Dell'Anna (UU), Data-Driven Supervision of Autonomous Systems
06 Daniel Davison (UT), "Hey robot, what do you think?" How children learn with a social robot
07 Armel Lefebvre (UU), Research data management for open science
08 Nardie Fanchamps (OU), The Influence of Sense-Reason-Act Programming on Computational Thinking
09 Cristina Zaga (UT), The Design of Robothings. Non-Anthropomorphic and Non-Verbal Robots to Promote Children's Collaboration Through Play
10 Quinten Meertens (UvA), Misclassification Bias in Statistical Learning
11 Anne van Rossum (UL), Nonparametric Bayesian Methods in Robotic Vision
12 Lei Pi (UL), External Knowledge Absorption in Chinese SMEs

13 Bob R. Schadenberg (UT), Robots for Autistic Children: Understanding and Facilitating Predictability for Engagement in Learning
14 Negin Samaeemofrad (UL), Business Incubators: The Impact of Their Support
15 Onat Ege Adali (TU/e), Transformation of Value Propositions into Resource Re-Configurations through the Business Services Paradigm
16 Esam A. H. Ghaleb (UM), BIMODAL EMOTION RECOGNITION FROM AUDIO-VISUAL CUES
17 Dario Dotti (UM), Human Behavior Understanding from motion and bodily cues using deep neural networks
18 Remi Wieten (UU), Bridging the Gap Between Informal Sense-Making Tools and Formal Systems - Facilitating the Construction of Bayesian Networks and Argumentation Frameworks
19 Roberto Verdecchia (VU), Architectural Technical Debt: Identification and Management
20 Masoud Mansoury (TU/e), Understanding and Mitigating Multi-Sided Exposure Bias in Recommender Systems
21 Pedro Thiago Timbó Holanda (CWI), Progressive Indexes
22 Sihang Qiu (TUD), Conversational Crowdsourcing
23 Hugo Manuel Proença (LIACS), Robust rules for prediction and description
24 Kaijie Zhu (TUE), On Efficient Temporal Subgraph Query Processing
25 Eoin Martino Grua (VUA), The Future of E-Health is Mobile: Combining AI and Self-Adaptation to Create Adaptive E-Health Mobile Applications
26 Benno Kruit (CWI & VUA), Reading the Grid: Extending Knowledge Bases from Human-readable Tables
27 Jelte van Waterschoot (UT), Personalized and Personal Conversations: Designing Agents Who Want to Connect With You
28 Christoph Selig (UL), Understanding the Heterogeneity of Corporate Entrepreneurship Programs

2022 1 Judith van Stegeren (UT), Flavor text generation for role-playing video games
2 Paulo da Costa (TU/e), Data-driven Prognostics and Logistics Optimisation: A Deep Learning Journey
3 Ali el Hassouni (VUA), A Model A Day Keeps The Doctor Away: Reinforcement Learning For Personalized Healthcare
4 Ünal Aksu (UU), A Cross-Organizational Process Mining Framework
5 Shiwei Liu (TU/e), Sparse Neural Network Training with In-Time Over-Parameterization

6 Reza Refaei Afshar (TU/e), Machine Learning for Ad Publishers in Real Time Bidding
7 Sambit Praharaj (OU), Measuring the Unmeasurable? Towards Automatic Co-located Collaboration Analytics
8 Maikel L. van Eck (TU/e), Process Mining for Smart Product Design
9 Oana Andreea Inel (VUA), Understanding Events: A Diversity-driven Human-Machine Approach
10 Felipe Moraes Gomes (TUD), Examining the Effectiveness of Collaborative Search Engines
11 Mirjam de Haas (UT), Staying engaged in child-robot interaction, a quantitative approach to studying preschoolers' engagement with robots and tasks during second-language tutoring
12 Guanyi Chen (UU), Computational Generation of Chinese Noun Phrases
13 Xander Wilcke (VUA), Machine Learning on Multimodal Knowledge Graphs: Opportunities, Challenges, and Methods for Learning on Real-World Heterogeneous and Spatially-Oriented Knowledge
14 Michiel Overeem (UU), Evolution of Low-Code Platforms
15 Jelmer Jan Koorn (UU), Work in Process: Unearthing Meaning using Process Mining
16 Pieter Gijsbers (TU/e), Systems for AutoML Research
17 Laura van der Lubbe (VUA), Empowering vulnerable people with serious games and gamification
18 Paris Mavromoustakos Blom (TiU), Player Affect Modelling and Video Game Personalisation
19 Bilge Yigit Ozkan (UU), Cybersecurity Maturity Assessment and Standardisation
20 Fakhra Jabeen (VUA), Dark Side of the Digital Media — Computational Analysis of Negative Human Behaviors on Social Media
21 Seethu Mariyam Christopher (UM), Intelligent Toys for Physical and Cognitive Assessments
22 Alexandra Sierra Rativa (TiU), Virtual Character Design and its potential to foster Empathy, Immersion, and Collaboration Skills in Video Games and Virtual Reality Simulations
23 Ilir Kola (TUD), Enabling Social Situation Awareness in Support Agents
24 Samaneh Heidari (UU), Agents with Social Norms and Values — A framework for agent based social simulations with social norms and personal values

C

Curriculum Vitae

Anna Latour was born in Leiden to parents who fully expected her to study arts or literature. Instead, after completing her high school education at the Haags Montessori Lyceum, she obtained a Bachelor's degree in Physics and Astrophysics from the University of Amsterdam. She then worked at the IT department of a pharmaceutical company before starting a pre-Master program at Leiden University to prepare herself for a Master in Artificial Intelligence at that same university.

Anna did the research for her Master thesis at Declarative Languages and Artificial Intelligence (DTAI) group of KU Leuven in Belgium, under the supervision of Dr. Siegfried Nijssen and the DTAI team. She graduated from the Master's program cum laude in 2016 and was awarded with the thesis award from the Koninklijke Nederlandse Vereniging voor Informatieprofessionals (KNVI) and the Koninklijke Hollandsche Maatschappij der Wetenschappen (KHMW) for her Master thesis: *Incremental algorithms for solving stochastic constraint optimisation problems with probabilistic logic programming*.

In 2017, Anna started as a PhD student of Leiden University, under the supervision of Prof. Dr. Joost N. Kok and Dr. Siegfried Nijssen. Her research was funded by an NWO TOP grant awarded to Dr. Nijssen for his PRObabilistic Features for Intelligent Declarative Data Science (PROFIDDS) project. Anna spent the first year of her PhD with the Artificial Intelligence & Algorithms (AIA) group at the Institute of Information and Communication Technologies, Electronics and Applied Mathematics (ICTEAM) at Université catholique de Louvain in Louvain-la-Neuve, Belgium. After moving back to The Netherlands in 2018, she joined Prof. Dr. Holger H. Hoos's Automated Design of Algorithms (ADA) group at Leiden University. In 2019, Anna visited Prof. Dr. Fahiem Bacchus's group at the University of Toronto to work on weighted model counting techniques.

During her PhD, Anna was a member of the Klankbordgroep Diversiteitsbeleid, the diversity policy feedback group of Leiden University. Additionally, she was a member of Leiden University's Studium Generale programme committee.

In 2018, Anna was awarded Google's Women Techmakers Scholarship for having demonstrated 'outstanding academic achievement, leadership and community involvement', in part for her efforts to increase the diversity, equity and inclusion in STEM (science, technology, engineering and mathematics). Anna's other awards include a research pitch prize for presenting her research 'with great clarity, content and charisma' (ranking highest in the jury report and receiving three times as many audience votes as the runner-up) and the AAAI 2021 outstanding reviewer award, for the 'exceptional care, thoroughness, and thoughtfulness' with which she approached the reviews and discussions of the papers assigned to her.

Anna started the next chapter of her scientific career in February 2022, as a Research Fellow in Prof. Dr. Kuldeep Meel's research group at the School of Computing (SoC) of the National University of Singapore (NUS). She is working on topics related to Boolean satisfiability and the field of *Beyond* $\mathcal{NP}$, and aims to pursue a career in academia.

There are few things that make Anna happier than the opportunity to search for geocaches and Annunciations while travelling for work.

D

List of Publications

Conference papers

- Anna L.D. Latour, Behrouz Babaki, Anton Dries, Angelika Kimmig, Guy Van den Broeck, and Siegfried Nijssen. '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)*. 2017, pp. 495–511.

- Anna Louise D. Latour, Behrouz Babaki, Siegfried Nijssen. 'Stochastic Constraint Propagation for Mining Probabilistic Networks'. In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI 2019)*. 2019, pp. 1137–1145.

Journal papers

- Anna L.D. Latour, Behrouz Babaki, Daniël Fokkinga, Marie Anastacio, Holger H. Hoos, and Siegfried Nijssen. 'Exact Stochastic Constraint Optimisation with Applications in Network Analysis'. In: *Artificial Intelligence, vol 304, 2022.*

Workshop papers

- Anna L.D. Latour, Behrouz Babaki, and Siegfried Nijssen. 'Stochastic Constraint Optimization using Propagation on Ordered Binary Decision Diagrams'. In: *Eighth International Workshop on Statistical Relational AI (StarAI 2018), colocated with IJCAI 2018, Stockholm, Sweden, 2018*.
- Daniël Fokkinga, Anna Louise D. Latour, Marie Anastacio, Siegfried Nijssen, and Holger Hoos. 'Programming a Stochastic Constraint Optimisation Algorithm, by Optimisation'. In: *Data Science meets Optimization workshop 2019 (DSO 2019), colocated with IJCAI 2019, Macao, 2019*.

Extended abstracts

- Anna Louise D. Latour, Behrouz Babaki, and Siegfried Nijssen. 'Stochastic Constraint Propagation for Mining Probabilistic Networks'. In: *Proceedings of the Reference AI & ML Conference for Belgium, Netherlands & Luxembourg, BNAIC/BENELEARN 2019, Brussels, 2019*.
- Anna L.D. Latour, Behrouz Babaki, Daniël Fokkinga, Marie Anastacio, Holger H. Hoos, and Siegfried Nijssen. 'Stochastic Constraint Optimisation with Applications in Network Analysis'. In: *International Workshop on Model Counting (MCW), colocated with SAT 2020*.
- Jeroen G. Rook, Anna L.D. Latour, Siegfried Nijssen, and Holger H. Hoos. 'Better Caching for Better Model Counting'. In: *International Workshop on Model Counting (MCW), colocated with SAT 2020*.
- Jeroen G. Rook, Anna L.D. Latour, Siegfried Nijssen, and Holger H. Hoos. 'Caching in Model Counters: A Journey through Space and Time'. In: *International Workshop on Counting and Sampling, colocated with SAT 2021*.