Programming a Stochastic Constraint Optimisation Algorithm, by Optimisation

Daniël Fokkinga[1] , **Anna Louise D. Latour**[1] , **Marie Anastacio**[1] ,
Siegfried Nijssen[2] and **Holger Hoos**[1,3]
[1]Leiden University, Leiden, The Netherlands
[2]UCLouvain, Louvain-la-Neuve, Belgium
[3]University of British Columbia (UBC), Vancouver, Canada
d.b.fokkinga@umail.leidenuniv.nl, {a.l.d.latour, m.i.a.anastacio, h.h.hoos}@liacs.leidenuniv.nl,
siegfried.nijssen@uclouvain.be

Abstract

Stochastic Constraint Optimisation Problems (SCOPs), such as the *viral marketing problem* and *transmission grid reliability problem*, arise in fields such as industry, governance and science. The recently proposed Stochastic Constraint Probabilistic Prolog (SC-ProbLog) language makes it possible to model and solve such SCOPs. Solving SCOPs exactly is NP-hard, and to solve real-world problems, exact SCOP solving methods must employ highly optimised heuristics. We propose to follow the principle of *Programming by Optimisation (PbO)*: we expose the design choices of a recently proposed SCOP solving method and optimise these using *Automated Algorithm Configuration (AAC)*. For a set of viral marketing problems, our optimised SCOP solver runs up to 26 times faster and solves almost two thirds of the instances that could not be solved within a cutoff time of ten minutes, by an expert-chosen default configuration of the solver. For a set of transmission grid reliability problems, the optimised configuration solves ten percent more instances overall, and solves some instances up to ten times faster.

1 Introduction

Problems in which we have to make optimal decisions under constraints and uncertainty are common in fields like industry, governance and science.

Consider for example the *viral marketing problem*, a well-known problem in the data mining literature [Kempe *et al.*, 2003]. We are given a probabilistic network, where nodes correspond to people and the directed edges to stochastic influence relationships, indicating how likely a person is to influence another. We want to leverage *word-of-mouth* to promote a new product in the network. We are given k free samples to distribute to people in the network, to start this process. Which group of k people is the most influential?

Another example is the *transmission grid reliability problem* [Duenas-Osorio *et al.*, 2017]. Here we are given a powergrid, where nodes correspond to consumers and producers of power, and edges to powerlines. In the event of a natural disaster, like an earthquake or hurricane, powerlines might break. If too many of them do, consumers may become disconnected and lose power. Each powerline has a certain probability of remaining intact during a disaster. By reinforcing powerlines we can increase this probability. This can be expensive, but we are given a budget for powerline maintenance. Which powerlines do we reinforce such that we maximise the expected number of power consumers that are still connected to a power source after a disaster, while not exceeding our budget?

Constraint optimisation problems that involve a constraint or objective function with a stochastic component, are called *Stochastic Constraint Optimisation Problems (SCOPs)*.

A recently developed method for solving SCOPs exactly leverages modelling and solving techniques from the fields of *Constraint Programming (CP)* and *Probabilistic Logic Programming (PLP)* to solve such SCOPs [Latour *et al.*, 2019]. This method consists of three stages: modelling the problem in a logic program, compiling its probability distribution to a data structure that supports tractable probabilistic inference, and finally searching for an optimal solution, in a way that takes advantage of specific properties of this data structure.

While this method has shown its merit in a proof of principle, it has not been optimised yet. Since solving a problem such as the viral marketing problem is an NP-hard task [Kempe *et al.*, 2003], the optimisation of the solving algorithm is key, lest we can only solve the smallest of problems.

The proposed solution to this problem is two-fold. First, many design choices are hard-coded in the current solver. We apply the principle of *Programming by Optimisation (PbO)* [Hoos, 2012] by first exposing those choices as parameters and providing alternatives to these choices, and then applying *automated algorithm configuration (AAC)* [Hoos, 2011] to find which combination of those design choices performs best on a given set of problems.

Our contributions are the following: **1)** We apply PbO to the pipeline presented by Latour *et al.* [2019]: we expose parameters for configuration and implement alternative design choices, before using AAC to optimise the resulting solver[1]; **2)** we show that the automatically configured version of this SCOP solver outperforms the hand-configured version presented earlier by Latour *et al.* [2019].

The remainder of this study is organised as follows. We

[1]Available from ada.liacs.nl/projects/scop-solver.

provide a definition of the SCOPs we study in Section 2, along with an outline of the SCOP solver. Section 3 provides a short introduction to PbO and AAC, and in Section 4 we list the different parameters and design choices we consider in this work. We present experimental results in Section 5 and provide general conclusions and some thoughts on future work in Section 6.

2 SCOPs

Stochastic Constraint Optimisation Problems (SCOPs) are found in diverse areas of application, including industry, governance and science. In this section, we show how to model problems such as the viral marketing problem as SCOPs and provide a short overview of the recently proposed SCOP solver whose automatic configuration we study in this work. We refer the reader to recent work by Latour *et al.* [2019] for a detailed discussion of that solver.

2.1 Problem Description

In the viral marketing problem, our goal is to maximise the expected number of people buying our product (the *optimisation criterion*). We rely on *word-of-mouth* for people who buy our product to turn other people in their social network into new buyers. Specifically, we assume stochastic relationships between people that determine how likely they are to be influenced by their acquaintances. We start the word-of-mouth process by distributing at most $k \in \mathbb{N}^+$ free samples (the *constraint*) of our product to people in the network. For simplicity, we assume that any person who receives a free sample will love it and buy it in the future. Similarly, we assume that if a person u buys the product and person u influences v, v will also buy the product.

We model this problem as a SCOP using two types of Boolean variables: *decision* variables, which represent whether or not a person receives a free sample, and *stochastic* variables, which represent whether a person has influence on another. Each stochastic variable has an independent probability of either being *True* or *False*, while the values of the decision variables are determined by choice. An assignment of truth values to a set of decision variables is called a *strategy* σ. The objective is to maximise the *objective function*

$$\sum_i \rho_i \cdot v_i, \tag{1}$$

where v_i can be either the value assigned to decision variable i or a conditional probability $P(\phi_i \mid \sigma)$; this conditional probability represents the probability of an event ϕ_i happening, given a strategy σ. The meaning of ϕ_i is further specified depending on the context of the problem. With each v_i we can associate a reward $\rho_i \in \mathbb{R}^+$, such that the objective function represents *expected utility*. In the following, we assume $\rho_i = 1$ for simplicity.

In practice, we are often only interested in a subset of events. We call this subset the *set of interest*, Φ.

The *cardinality* of a solution is the number of decision variables that are chosen to be *True* in the strategy σ. To model the limited supply of free samples, we place an upper bound on the cardinality, in the form of a *linear constraint*.

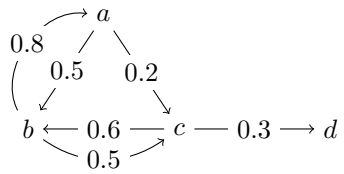


Figure 1: Social network of four persons $\{a, b, c, d\}$. Edges represent the stochastic influence a person has on another.

We solve this SCOP by converting the objective function from Equation (1) into the *stochastic constraint*

$$\sum_i \rho_i \cdot v_i > \theta \tag{2}$$

and solving the resulting constraint *satisfaction* problem repeatedly, for increasing $\theta \in \mathbb{R}^+$, until no solution can be found. Here, θ takes the best value for $\sum_i \rho_i \cdot v_i$ found so far for a valid solution.

Example 2.1. We formulate a viral marketing problem for the network of Figure 1 as follows. For every node i, we define a decision variable d_i, and for every edge (i, j) a stochastic variable t_{ij} with a probability of evaluating to *True* that is equal to the label on edge (i, j) in Figure 1. We represent the event that person i buys our product with a propositional formula ϕ_i over decision variables and stochastic variables.

Suppose that the people in this network can be divided in two categories, based on geographical location, and our product is only sold in one of the two geographical locations. We then only care about turning people in that location into customers. We model this by defining our set of interest to be, e.g., $\Phi = \{\phi_a, \phi_b\}$.

The objective is to find a strategy σ that maximises $\sum_{\phi \in \Phi} P(\phi \mid \sigma)$. Finally, we constrain the number of people that receive a free sample: $\sum_{i \in \{a,b,c,d\}} d_i \leq k$.

Note that in this viral marketing problem, the decision variables are associated with the *nodes* in the network. We also study a power transmission grid reliability problem, in which the decision variables are associated with the *edges* of the probabilistic network, instead.

In this problem we are given a network in which nodes represent *power producers* (such as powerplants), *power consumers* or *intermediate grid nodes*. The nodes are connected by *powerlines*. We want to maximise the expected number of consumers that are still connected to at least one power producer in the case of a natural disaster, during which powerlines can break. Each powerline has a probability that it remains intact during a natural disaster. By reinforcing a powerline, we can increase this probability. We are given a budget for such reinforcements. Which powerlines do we reinforce such that we maximise the number of consumers that are still connected to producers after a natural disaster, while respecting our budget?

Example 2.2. For the network in Figure 2, we model this problem as follows. For every powerline l we define a decision variable d_l that indicates if the powerline is chosen to be reinforced, and a stochastic variable t_l, which indicates if the

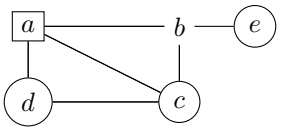


Figure 2: Network of powerlines between a producer a, three consumers $\{c, d, e\}$ and an intermediate grid node b.

powerline remains intact after a disaster. In our model, the probability p_l that t_l is *True* is defined as

$$p_l = \begin{cases} p_{l,1} & \text{if } d_l = \textit{False}; \\ p_{l,2} & \text{otherwise,} \end{cases}$$

with $p_{l,1} < p_{l,2}$. We represent the event that a power consumer i is still connected to at least one power producer with a propositional formula ϕ_i over decision variables and stochastic variables. We define a set of interest $\Phi = \{\phi_d, \phi_e\}$, if we are only interested in a specific subset of power consumers. The objective is to find a strategy σ that maximises $\sum_{\phi \in \Phi} P(\phi \mid \sigma)$. We constrain the powerlines that we reinforce: $\sum_{l \in \text{lines}} d_l \cdot \gamma_l \leq \beta$, where $\beta \in \mathbb{R}^+$ is our budget, and γ_l is the cost for reinforcing powerline l.

2.2 Solving SCOPs

In the following, we study the automatic configuration of a recently developed solving method for a sub-class of SCOP: those whose underlying probability distribution has a monotonic property [Latour *et al.*, 2019]. We will refer to this method as SCOP SOLVER. In this section we provide a high-level overview of the three stages of SCOP SOLVER: modelling, compilation and solving.

The *modelling stage* uses the probabilistic programming language SC-ProbLog [Latour *et al.*, 2017], an extension of ProbLog [De Raedt *et al.*, 2007].

In the *compilation stage*, SC-ProbLog uses *knowledge compilation* [Darwiche and Marquis, 2001] to obtain a compact representation of the probability distribution in the form of an *Ordered Binary Decision Diagram (OBDD)* [Bryant, 1986]. OBDDs are undirected acyclic graphs that allow online tractable probabilistic inference, a task which is #P-complete [Roth, 1996] in general, at the cost of compilation. Once compiled, OBDDs allow the user to evaluate the quality of any strategy σ in time linear in the size of the OBDD (by a bottom-up traversal of the arithmetic circuit associated with the OBDD), making knowledge compilation widely used for probabilistic inference [De Raedt *et al.*, 2008].

For the *solving stage*, Latour *et al.* [2019] propose to leverage the search and propagation technology of *Constraint Programming (CP)* to efficiently traverse the search space of possible solutions to find the optimal strategy σ^*. A CP solver induces the search tree by *branching*: assigning either *True* or *False* to decision variables, to obtain a partial assignment σ' and propagating the consequences. This process continues until either a solution is found or σ' leads to a violation of one or more constraints, at which point the solver backtracks to explore another part of the search space.

In this work, we apply PbO to the compilation and solving stages of SCOP SOLVER. We describe parameters and design choices relevant to these stages in Section 4.

3 Programming by Optimisation

To improve the performance of SCOP SOLVER, we apply the Programming by Optimisation paradigm. We provide an introduction to this concept, and then an introduction to automated algorithm configuration, which critically enables PbO. We close the section with a discussion of our choice for a specific tool for automatically configuring SCOP SOLVER.

3.1 Programming by Optimisation

During algorithm or sofware development, there are typically multiple possible ways of achieving each subtask. However, often only one of these *design choices* is implemented in the final version of an algorithm or software system. This choice is often made based on limited experimentation, with a specific application in mind, while the alternatives are abandoned. These design choices have no effect on correctness, but can affect performance, especially when dealing with computationally challenging problems.

The paradigm of *Programming by Optimisation (PbO)* [Hoos, 2012] introduces a different approach to design choices. Developers who take a PbO-based approach to software or algorithm design, implement multiple alternatives for many element or components. They provide the end user with the choice between these options, by exposing them as a configurable parameters. Following the PbO paradigm, developers focus on exploring alternatives for design choices instead of determining the best instantiations for specific applications. Moreover, expanding the design space of a given algorithm or software system and exposing choices as configurable parameters provides the basis for using automated algorithm configuration techniques for performance optimisation. The existence of effective automated algorithm configuration procedures hence critically enables PbO-based algorithm and software design.

3.2 Automated Algorithm Configuration

When the PbO paradigm is followed, the resulting algorithm comes with a set of parameters. The parameter settings of such an algorithm (its *configuration*) can have a substantial impact on its performance, and the optimal choice may vary for different sets of problems. This also applies to many state-of-the-art algorithms that naturally come with many parameters. Using the right parameter setting is then critical for reaching state-of-the-art performance – even more so for NP-hard problems, such as SCOPs. The process of finding an optimal configuration for a problem set is called *automated algorithm configuration (AAC)* [Hoos, 2011].

The *algorithm configuration problem* is defined as follows. Given:

- a target algorithm A;
- a list of parameters $q_1, \ldots, q_n$ of A;
- a configuration space C that defines for each $q_\ell, for \ell \in \{1, \ldots, n\}$, its domain (e.g., *integer*, *real-valued* or *categorical*) and possible values; each set of such values is a configuration $c \in C$;
- a set of problem instances I;
- a performance metric m that measures the performance of the target algorithm A on the instances of I for a given configuration c,

find a configuration $c^* \in C$ that optimises performance metric m of A on I.

This configuration is expected to be one that performs well on instances similar to the ones in I. To tackle the configuration problem, many different algorithms have been developed, and we refer to those as *configurators*.

3.3 Application to SCOPs

As explained above, AAC is an essential tool for PbO-based software development. It has been applied successfully to algorithms that solve NP-hard problems, such as the Boolean satisfiability problem (SAT) [Hutter *et al.*, 2017] and the travelling salesperson problem (TSP) [López-Ibánez *et al.*, 2011]. Hence, we expect it to also be beneficial to NP-hard problems, such as SCOPs.

There are several state-of-the-art configurators available [Balaprakash *et al.*, 2007; Hutter *et al.*, 2011; Ansótegui *et al.*, 2015]. In particular, SMAC [Hutter *et al.*, 2011] and GGA++ [Ansótegui *et al.*, 2015] are *model-based*: They build a model that captures the dependency of the performance of the target algorithm on its configuration. This model is used to predict the performance of configurations on multiple instances and to select promising candidate configurations. This method supports diversity of domains of parameters and conditional parameters, whose activation depends on other parameters' values.

Because of the nature of the parameters of SCOP SOLVER (Section 2.2), we expect that such a model-based search strategy would yield the best results. We chose the general-purpose configurator SMAC, because it is one of the best-performing configurators and is freely available.

4 Approach

In the development of SCOP SOLVER (Section 2.2), choices were made on how to approach certain tasks. We refer to these choices as *design choices*. We add alternative design choices and expose hard-coded choices as parameters, for the compilation and solving stages of the solver. This enables us to apply AAC to optimise SCOP SOLVER. In this section we discuss the different design choices we consider in this work.

4.1 Compilation Stage

The *variable order*, which specifies the order in which variables are encountered during the traversal of the OBDD from root to leaves, has a significant impact on the shape and size of an OBDD. Finding the optimal order is an NP-complete problem, and many different OBDD minimisation algorithms exist [Bollig and Wegener, 1996].

The configuration space we consider for the compilation stage of SCOP SOLVER consists of: a Boolean variable that indicates whether to minimise an OBDD, or to leave it with default variable order; the different OBDD minimisation algorithms; and their parameters. We summarise this configuration space in Table 1.

4.2 Solving Stage

During the solving stage, *branching heuristics* determine how SCOP SOLVER traverses the search space. A CP search algorithm uses branching heuristics to decide in which order unbound decision variables are instantiated and to decide for each variable which value to explore first: *True* or *False*. In our description of the heuristics below, we describe how the next decision variable is chosen, and indicate which value is assigned to that decision variable first. After backtracking, the CP search algorithm may assign the other value.

Existing heuristics

Latour *et al.* [2019] describe six different branching heuristics. *Top*-0 (*Bottom*-0) branches on the highest (lowest) unbound decision variable in the OBDD's variable order and assigns to this variable the value *False* (*True*) first. These heuristics are static, and inspired by the size and shape of the OBDD.

Derivative-0 and *Derivative*-1 are regret-based heuristics, aiming to quickly find a high-quality solution. The regret is defined as the change in objective value for a change in strategy, and is called the *derivative* of a decision variable. Derivatives are recomputed at each node of the search tree. The heuristic *Derivative*-0 (*Derivative*-1) assigns first the value *False* (*True*) to the variable with the *smallest* (*largest*) absolute derivative. These heuristics are dynamic (they are (re)computed during search) and based on the relative significance of the decision variables at each point in the search.

New heuristics

We propose four new heuristics that take a different approach: they are derived directly from the (social) network on which the viral marketing problem is defined.

Two heuristics select decision variables based on the *degree* of their corresponding nodes. *Degree*-0 (*Degree*-1) assigns first the value *False* (*True*) to the variable with the *smallest* (*largest*) degree.

The third and fourth heuristic use an approximation of the *influence* for the nodes that correspond to the decision variables, inspired by work on social influence [Borgs *et al.*, 2014]. We calculate the influence of a node i by starting n walks through the graph starting at i, where n is the degree of i. We include each node j according to the probability on the label of the edge (i, j) and repeat this process recursively for each j that has been included. The influence of node i is the number of unique nodes visited during these walks. We limit each path length by a maximum of `MaxDepth` (a configurable parameter of this heuristic). Hence, this influence approximation measures the expected number of people each person can influence in their `MaxDepth`-neighbourhood. *Influence*-0 (*Influence*-1) first assigns the value *False* (*true*) to the variable with the *smallest* (*largest*) influence.

Translating new heuristics

The four new heuristics (*Degree*-0, *Degree*-1, *Influence*-0 and *Influence*-1) are defined with a variable marketing problem in mind as application. In these problems, each decision variable corresponds to a node in the network. However, for the transmission grid reliability problem, the decision variables correspond to an edge instead. To translate these heuristics such that they can be used to select edges from the network, we do the following: if node i has the value X and node j the value Y according to the chosen heuristic, then edge (i, j) receives the value $X + Y$ for this same heuristic.

Compilation stage

`Minimise`, domain: {*False*, *True*}
Minimise the OBDD.

`VarOrder`, domain: {*Sif*, *GSif*, *WP*, *SA*, *GA*, *Rand*}
Sifting (*Sif*) [Rudell, 1993], Group Sifting (*GSif*) [Panda and Somenzi, 1995], Window Permutation (*WP*) [Ishiura *et al.*, 1991], Genetic Algorithm (*GA*) [Drechsler *et al.*, 1996], Simulated Annealing (*SA*) [Bollig *et al.*, 1995], Random (*Rand*).

`Converging`, domain: {*False*, *True*}
Repeat variable reordering algorithm until no improvement in OBDD size is found (if `VarOrder` ∈ {*Sif, GSif, WP*}).

`SymSifting`, domain: {*False*, *True*}
If the size of the OBDD is invariant under swapping two variables in the variable order, group them together and swap as a group, not individually (if `VarOrder` ∈ {*Sif*, *GSif*}).

`MaxSwap`, domain: $\mathbb{N}^+$
Upper bound on number of times two variables can be swapped in the variable order (if `VarOrder` ∈ {*Sif*, *GSif*}).

`MaxSift`, domain: $\mathbb{N}^+$
Upper bound on number of variables that are sifted, i.e. moved up and/or down the variable order by swapping with other variables (if `VarOrder` ∈ {*Sif*, *GSif*}).

`MaxGrowth`, domain: $\mathbb{R}^+$
Maximum relative increase of OBDD size during minimisation (if `VarOrder` ∈ {*Sif*, *GSif*}).

`WSizes`, domain: {2, 3, 4}
Evaluate different permutations of `WSizes` consecutive variables in the variable order at a time (if `VarOrder` = *WP*).

Solving stage

`BranchHeur`, domain: {*Top*-0, *Top*-1, *Bottom*-0, *Bottom*-1, *Derivative*-0, *Derivative*-1, *Degree*-0, *Degree*-1, *Influence*-0, *Influence*-1}
Branching heuristics used for selection of variables and values.

`MaxDepth`, domain: $\mathbb{N}^+$
Maximum length of paths traversed
(if `BranchHeur` ∈ {*Influence*-0,*Influence*-1}).

Table 1: SCOP SOLVER parameters, their domains, short descriptions, and conditions. Except for MINIMISE itself, all compilation stage parameters are conditioned on MINIMISE = *True*.

For the influence-based heuristics we do not perform random walks, but simply sum the sizes of the `MaxDepth`-neighbourhoods of the endpoints of each line. This is because, in our dataset, $p_{l,1}$ is the same for each line l, and because the value of the probability on each line of the network depends on the value of the associated decision variable.

5 Experiments

In this section, we describe the automated configuration of SCOP SOLVER and its experimental evaluation.

Our experiments are guided by the hypothesis that exposing parameters of SCOP SOLVER, providing alternative design choices and automatically configuring the resulting algorithm for any set of given SCOP instances, provides a configured SCOP SOLVER that outperforms the original in terms of running time and number of solved instances for a given cutoff time.

5.1 Datasets

To test our hypothesis, we have performed experiments on two different datasets. We summarise some characteristics of these datasets in Table 2.

Viral marketing dataset

We formulated a viral marketing problem on directed multi-graph data from Facebook representing user interactions [Viswanath *et al.*, 2009]. This dataset consists of 46 952 nodes (users) and 876 993 edges (wall posts). We used community detection [Blondel *et al.*, 2008] to extract all communities of twenty to thirty nodes.

To convert these communities into probabilistic networks, we used the *independent cascade model* for spread of influence [Kempe *et al.*, 2003]. When a user posts multiple messages on the wall of another user, there are multiple (parallel) edges between these two users. Parallel edges from node u to v are replaced by a single edge with weight of $1-(1-p)^{c_{uv}}$, where c_{uv} is the number of edges from u to v, and $p = 0.1$ is a constant probability, which is interpreted as the probability a single wall post has to influence the user that receives it.

Dataset	Size training	Size test	Problem size (# nodes)
Vm	197	196	20–30
Pg	68	68	20–100

Table 2: Size of the training and test of both the Viral marketing (Vm) dataset and Powergrid (Pg) dataset and the size of the individual problem instances.

The *set of interest* Φ on which we define our objective function consists of the fifty percent highest-degree nodes in a community. We chose an upper bound of $k = 10$ on the cardinality of the solution for all networks.

The resulting set contains 393 problem instances, which we divided into a training and test set such that their distributions of communities with different numbers of nodes are the same.

Powergrid dataset

The instances of the transmission grid reliability problem are defined on network models of the European and North-American high-voltage power grids [Wiegmans, 2016], extracted by `GridKit`[2]. These networks are undirected graphs consisting of power producers, consumers and minor grid nodes (nodes) and powerlines that connect them (edges).

For each powergrid from a single European country or North-American state, we extracted the greatest connected components that contain at least one power producer. We se-

[2] Available from github.com/bdw/GridKit.

lected the components that have at least twenty, and at most one hundred nodes, resulting in a set of 34 networks. The set of interest Φ, consists of a randomly selected set of power consumers for each network. The size of this set is equal to 50% of the total number of power consumers in each network.

We used a probability of 0.4 that a powerline remains intact during a natural disaster and 0.875 if it is reinforced [Duenas-Osorio *et al.*, 2017]. We assume a uniform cost of $\gamma_l = 1$ for reinforcing a powerline l. We chose a budget of $\beta = 10$ for all problem instances.

For each of the 34 networks, we created four instances with a different Φ, which are distributed equally over our training and test sets.

5.2 Hardware and Software

For our experiments we used SMAC [Hutter *et al.*, 2011], as stated in Section 3.2.

SCOP SOLVER makes use of an `SC-ProbLog` version based on `ProbLog 2.1` [Fierens *et al.*, 2015] for modelling, the `dd 0.5.4` library [Filippidis, 2018] for OBDD compilation and the `Scala 2.12` library `OscaR 4.0.0` [OscaR Team, 2012] for solving. We used the Cython binding of the `dd` library to `CUDD 3.0.0` [Somenzi, 2004] for the implementation of alternative minimisation methods for the OBDDs. Our configuration experiments were run on a cluster with 32 nodes, each equipped with 94 GB RAM and two Intel Xeon E5-2683 CPUs with 16 cores, running at 3.0 GHz using CentOS Linux 7.6.1810; for our configuration experiments, we used SMAC v3.

5.3 Experimental Protocol

For each of the datasets described in Section 5.1, we performed fifteen independent 48-hour runs of SMAC on the training set. We minimised PAR10 (penalised average running time with penalty factor 10) and a cutoff time of 600 CPU seconds, meaning that we measured the average running time of SCOP SOLVER over all instances in a given set and counted each timed-out run as ten times the cutoff time.

The cutoff time was selected based on initial experiments on the viral marketing dataset, such that a reasonable success rate was achieved for the default configuration. The time limit for each SMAC run was adjusted according to the cutoff time, following the example of scenarios with a similar cutoff time from a well-known configuration library [Hutter *et al.*, 2014].

For each of these fifteen runs, we evaluated the final incumbent (the configuration with the best PAR10 value) on the training set and selected the configuration with the best performance. We then evaluated this *optimised configuration* on the test set and comparing it with the *default configuration*. We note that the default configuration of SCOP SOLVER was based on the results from previous experiments performed by Latour *et al.* [2019], thus providing a strong baseline for our configuration experiments.

5.4 Results

In this subsection we present and discuss the results from our experiments.

		All instances		Solved instances	
	Configuration	Training	Test	Training	Test
Vm	Default	1030.6	822.1	89.8	90.2
	Optimised	298.4	204.5	25.2	21.5
Pg	Default	3294.8	3283.0	113.2	104.2
	Optimised	3105.7	2928.6	52.6	32.6

Table 3: PAR10 values with a cutoff of 600 seconds of both configurations on the instances from the Viral marketing (Vm) dataset and Powergrid (Pg) dataset on all instances and only the instaces that were solved by either configuration within the cutoff time.

Viral marketing dataset results

To test our hypothesis on the dataset of viral marketing problems, we evaluated the performance of the optimised configuration and that of the default configuration on both the training and test set of viral marketing data.

The running times of these two configurations on the training set and test set are shown in Figure 3a. The optimised configuration is able to find a solution up to 26 times faster than the default configuration. For the majority of instances, the speedup ranges from zero- to ten-fold. As the running time of the default configuration increases, the speedup also tends to increase. There are no instances that can be solved by the default, but not by the optimised configuration.

The PAR10 values for both configurations are shown in Table 3. Compared to the default configuration, the optimised configuration yields a 71% decrease in the PAR10 for the training set, and a 75% decrease for the test set. There are more than twice as many time-outs on the training than on the test set for the optimised configuration, compared to 30% more timeouts for the default configuration; this indicates that the instances in the training set seem to be slightly harder to solve than those in the test set.

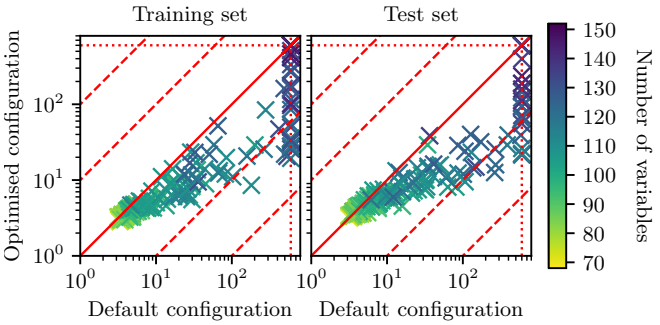
Table 3 also shows the PAR10 values for those instances that were solved by either the default, the optimised configuration, or both. From these values we conclude that the speedup for the solved instances is similar to that of the overall speedup.

Powergrid dataset results

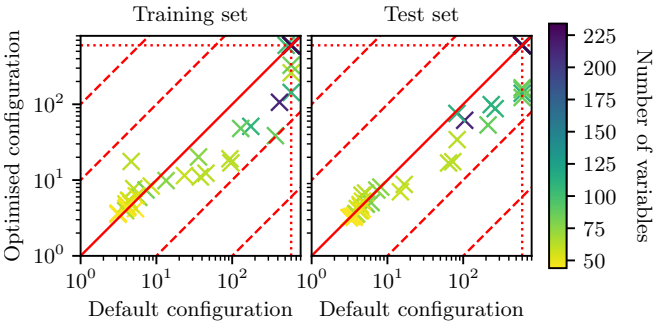
We also evaluated the performance of the optimised configuration from the powergrid data and compared it to that of the default configuration. The running times of these two configurations on the training and test data are shown in Figure 3b.

The default configuration outperforms the optimised configuration on a few instances, of which the majority is solved within ten CPU seconds. There is a single test set instance that is only solved by the default configuration within the cutoff time. For all other instances, the optimised configuration outperforms the default configuration with a speedup ranging from zero- to ten-fold and solves 6 more instances (out of a 136 in total) than the default configuration.

If we consider the PAR10 values reported in Table 3, the optimised configuration yields a decrease of 6% in the PAR10 value for the training set and a decrease of 11% for the test set. The speedup in PAR10 on the instances that are solved by either configuration is much higher (two- and three-fold for training and test set, respectively) than the speedup over



(a) Viral marketing dataset. The optimised configuration has 14 timeouts on the training and 6 on the test set. The default configuration has 33 timeouts on the training and 26 on the test set.



(b) Powergrid dataset. The optimised configuration has 35 timeouts on the training and 33 on the test set. The default configuration has 37 timeouts each on the training and test set.

Figure 3: Running time [CPU s] of both configurations on both datasets, with a cutoff time of 600 seconds (indicated by dotted lines). The diagonal lines represent differences in running time of factors of 0, 10 and 100.

all instances.

Analysis and Discussion

The default configuration to which we compare the optimised configurations from both datasets uses *Derivative*-1 for branching and does not perform any OBDD minimisation. Both optimised configurations perform OBDD minimisation using the symmetric, converging variant of the Sifting algorithm with specifically tuned parameters. The optimised configuration for the viral marketing and powergrid datasets use *Degree*-1 and *Derivative*-1 for branching, respectively.

On both datasets, the optimised configuration outperforms the default configuration, which confirms our hypothesis for these sets of SCOPs. We attribute this to the fact that by applying AAC, we found a previously unexplored configuration of our highly parametric SCOP SOLVER framework that is better suited to the problems from these datasets than the default configuration. Specifically, the optimised configuration for the compilation stage results in better performance than the default on the powergrid dataset. For the viral marketing dataset, the branching heuristic also differs.

While the optimised configuration outperforms the default on the powergrid dataset, the improvement is less pronounced than that observed for the viral marketing dataset. We suspect this has two reasons.

First, the problem instances from this dataset are on average harder to solve than the viral marketing instances. According to the PAR10 values of the default configuration on the test set, the powergrid dataset is three times as difficult. This difficulty leads to a low success rate, which hurts the quality of the configuration process. We expect that by increasing the cutoff time, the success rate could decrease. Consequently, SMAC obtains more information from the training set, which improves its ability to find promising configurations and possibly also the performance of the optimised configuration.

Second, the new heuristics we introduced in Section 4 were inspired by the application to viral marketing problems, specifically to approximate the influence of nodes, and end up being used only in the optimised configuration for the viral marketing dataset. We expect that implementing heuristics specifically aimed at solving SCOPs with decision variables on edges, such as the transmission grid reliability problem, would result in further performance increases for configurations optimised for this problem. An example of such a heuristic would be the *betweenness centrality* of an edge, which can be approximated such that is easy to compute, similar to the *degree* of a node.

6 Conclusion and Future Work

We presented an approach to automatically optimise the configuration of a high-performance method for solving SCOPs [Latour *et al.*, 2019]. Following the PbO paradigm [Hoos, 2012], we considered alternatives to the design choices made for several key components of this method and exposed those as parameters. We then applied automated algorithm configuration to the resulting, highly parametric algorithm framework, using SMAC [Hutter *et al.*, 2011], in order to optimise its configuration for a set of SCOPs. We evaluated the running time of the automatically configured solver against that of expert-chosen default settings, namely was the best performing configuration from the experiments of Latour *et al.* [2019], on two benchmarks.

On a set of viral marketing problems, the optimised configuration solved 13% more instances than the default configuration within a cutoff time of ten minutes, and achieved up to a 26-fold speedup on the solved instances. For a set of power transmission grid reliability problems, the optimised configuration solved 10% more instances within the same cutoff time and achieved up to a 10-fold speedup on the solved instances.

In future work, we aim to improve the quality of the branching heuristics, specifically developing heuristics aimed at the power transmission grid reliability problem. We will also consider other techniques that might result in increased performance, such as the grouping of decision variables together in the OBDD's variable order. To compare our approaches to other state-of-the-art SCOP solving techniques, we intend to evaluate a previously developed decomposition method that leverages existing MIP solvers [Latour *et al.*, 2017], since previous work as shown the successful application of AAC on MIP solvers [Hutter *et al.*, 2010]. Fi-

nally, we will perform experiments in which we include instance features for the problems under consideration, such as graph statistics. These features are used by SMAC to predict the performance of a configuration on different problem instances and are expected to improve the configuration process.

Acknowledgements. We thank Roger Paredes and Leonardo Dueñas-Osorio for their advice regarding the formulation of the powergrid reliability problem and the interpretation of the high-voltage powergrid data. This work was supported by the Netherlands Organisation for Scientific Research (NWO).

References

[Ansótegui *et al.*, 2015] C. Ansótegui, Y. Malitsky, H. Samulowitz, M. Sellmann, and K. Tierney. Model-based genetic algorithms for algorithm configuration. In *Proceedings of IJCAI*, 2015.

[Balaprakash *et al.*, 2007] P. Balaprakash, M. Birattari, and T. Stützle. Improvement strategies for the F-Race algorithm: Sampling design and iterative refinement. In *Proceedings of HM*, pages 108–122. Springer, 2007.

[Blondel *et al.*, 2008] V.D. Blondel, J. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *JSTAT*, 2008(10):P10008, 2008.

[Bollig and Wegener, 1996] B. Bollig and I. Wegener. Improving the variable ordering of OBDDs is NP-complete. *IEEE Transactions on computers*, 45:993–1002, 1996.

[Bollig *et al.*, 1995] B. Bollig, M. Löbbing, and I. Wegener. Simulated annealing to improve variable orderings for OBDDs. In *Proceedings of IWLS*, 1995.

[Borgs *et al.*, 2014] C. Borgs, M. Brautbar, J. Chayes, and B. Lucier. Maximizing social influence in nearly optimal time. In *Proceedings of ACM-SIAM SODA*, pages 946–957. SIAM, 2014.

[Bryant, 1986] R.E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691, 1986.

[Darwiche and Marquis, 2001] A. Darwiche and P. Marquis. A perspective on knowledge compilation. In *Proceedings of IJCAI*, pages 175–182. Morgan Kaufmann Publishers Inc., 2001.

[De Raedt *et al.*, 2007] L. De Raedt, A. Kimmig, and H. Toivonen. ProbLog: a probabilistic prolog and its application in link discovery. In *Proceedings of IJCAI*, pages 2468–2473, 2007.

[De Raedt *et al.*, 2008] L. De Raedt, K. Kersting, A. Kimmig, K. Revoredo, and H. Toivonen. Compressing probabilistic prolog programs. *Machine Learning*, 70(2-3):151–168, 2008.

[Drechsler *et al.*, 1996] R. Drechsler, B. Becker, and N. Gockel. Genetic algorithm for variable ordering of OBDDs. *IEEE Computers and Digital Techniques*, 143(6):364–368, 1996.

[Duenas-Osorio *et al.*, 2017] L. Duenas-Osorio, K.S. Meel, R. Paredes, and M.Y. Vardi. Counting-based reliability estimation for power-transmission grids. In *Proceedings of AAAI*, 2017.

[Fierens *et al.*, 2015] D. Fierens, G. Van den Broeck, J. Renkens, D. 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.

[Filippidis, 2018] I. Filippidis. dd python library, 2018. Available from `https://pypi.org/project/dd/`.

[Hoos, 2011] H.H. Hoos. Automated algorithm configuration and parameter tuning. In *Autonomous search*, pages 37–71. Springer, 2011.

[Hoos, 2012] H.H. Hoos. Programming by optimization. *Communications of the ACM*, 55(2):70–80, 2012.

[Hutter *et al.*, 2010] F. Hutter, H.H. Hoos, and K. Leyton-Brown. Automated configuration of mixed integer programming solvers. In *Proceedings of CPAIOR*, pages 186–202. Springer, 2010.

[Hutter *et al.*, 2011] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Proceedings of LION*, pages 507–523. Springer, 2011.

[Hutter *et al.*, 2014] F. Hutter, M. López-Ibáñez, C. Fawcett, M. Lindauer, H.H. Hoos, K. Leyton-Brown, and T. Stützle. AClib: A benchmark library for algorithm configuration. In *Proceedings of LION*, pages 36–40. Springer, 2014.

[Hutter *et al.*, 2017] F. Hutter, M. Lindauer, A. Balint, S. Bayless, H.H. Hoos, and K. Leyton-Brown. The configurable SAT solver challenge (CSSC). *Artificial Intelligence*, 243:1–25, 2017.

[Ishiura *et al.*, 1991] N. Ishiura, H. Sawada, and S. Yajima. Minimization of binary decision diagrams based on exchanges of variables. In *Proceedings of IEEE ICCAD*, pages 472–475. IEEE, 1991.

[Kempe *et al.*, 2003] D. Kempe, J. Kleinberg, and É. Tardos. Maximizing the spread of influence through a social network. In *Proceedings of ACM SIGKDD*, pages 137–146. ACM, 2003.

[Latour *et al.*, 2017] A.L.D. Latour, B. Babaki, A. Dries, A. Kimmig, G. Van den Broeck, and S. Nijssen. Combining stochastic constraint optimization and probabilistic programming. In *CP*, pages 495–511. Springer, 2017.

[Latour *et al.*, 2019] A.L.D. Latour, B. Babaki, and S. Nijssen. Constraint propagation on Binary Decision Diagrams for mining probabilistic networks. IJCAI, Macao, China (forthcoming), 2019.

[López-Ibánez *et al.*, 2011] M. López-Ibánez, J. Dubois-Lacoste, T. Stützle, and M. Birattari. The irace package, iterated race for automatic algorithm configuration. Technical report, IRIDIA Technical Report Series, 2011.

[OscaR Team, 2012] OscaR Team. OscaR: Scala in OR, 2012. Available from `https://bitbucket.org/oscarlib/oscar`.

[Panda and Somenzi, 1995] S. Panda and F. Somenzi. Who are the variables in your neighborhood. In *Proceedings of IEEE/ACM ICCAD*, pages 74–77. IEEE Computer Society, 1995.

[Roth, 1996] D. Roth. On the hardness of approximate reasoning. *Artificial Intelligence*, 82(1-2):273–302, 1996.

[Rudell, 1993] R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *Proceedings of IEEE/ACM ICCAD*, pages 42–47. IEEE, 1993.

[Somenzi, 2004] F. Somenzi. CUDD: CU Decision Diagram package-release 2.4.0, 2004. University of Colorado at Boulder.

[Viswanath *et al.*, 2009] B. Viswanath, A. Mislove, M. Cha, and K.P.Bimal Gummadi. On the evolution of user interaction in Facebook. In *Proceedings of ACM SIGCOMM WOSN*, pages 37–42, 2009.

[Wiegmans, 2016] B. Wiegmans. Gridkit: European and North-American extracts, 2016.