Optimal decision-making under constraints and uncertainty

Latour, A.L.D.

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.