Stochastic Constraint Programming

David Hemmi
Monash University
Data61/CSIRO, Melbourne, Australia
david.hemmi@monash.edu

Abstract

Combinatorial optimisation problems often contain uncertainty that has to be taken into account to produce realistic solutions. One way of describing the uncertainty is using scenarios, where each scenario describes different potential sets of problem parameters based on random distributions or historical data. While efficient algorithmic techniques exist for specific problem classes such as linear programs, there are very few approaches that can handle general Constraint Programming formulations with uncertainty. The goal of my PhD is to develop generic methods for solving stochastic combinatorial optimisation problems formulated in a Constraint Programming framework.

1 Introduction

Reasoning under uncertainty is important in combinatorial optimisation, since uncertainty is inherent to many problems in an industrial setting. An example of this especially hard class of optimisation problems is the generalised assignment problem [Albareda-Sambola *et al.*, 2006], where jobs with an uncertain duration have to be assigned to computers in a network, or the assignment of tasks to entities when developing a project plan.

The focus of my work is on stochastic optimisation problems (SOP) with a *combinatorial structure*. The random variables describing the uncertainty have finite support, e.g. discrete probability distribution. *Scenarios* describe the stochastic problem when all the random variables are fixed. Each scenario has a probability of occurrence. Stochastic problems are composed of a first and subsequent stages, the simplest being a two stage problem. The *first stage* denotes the problem before information about the random variables is revealed and first stage decisions are taken with respect to all scenarios. Second stage decisions are made once the scenario parameters are observed.

Combinatorial optimisation problems can be formulated using Mixed Integer (MIP) or Constraint Programming (CP) technologies. For the sake of brevity, this abstract only addresses CP, as it is most relevant for my research. In CP, modelling frameworks that allow solver agnostic modelling have been developed, e.g. MiniZinc [Nethercote *et al.*, 2007]. Problems described in MiniZinc can be solved with a range of CP, MIP or SAT solvers without tailoring the model for the specific solvers. The idea is to separate the modelling and solving, such that a model can be expressed without committing to a solving approach. *Stochastic MiniZinc* [Rendl *et al.*, 2014] is an extension of the MiniZinc modelling framework that supports random variables. It can automatically translate a stochastic optimisation problem into standard MiniZinc, by reformulating it into the *Deterministic Equivalent* (DE). However, traditional solver technology often cannot exploit the special model structure of scenario-based stochastic optimisation problems formulated as a DE. As a result, the search performance often scales poorly with increasing numbers of scenarios. The literature on stochastic optimisation problems containing integer variables is diverse, see [Birge and Louveaux, 2011]. Previous works have been concerned with problems that have a specific mathematical structure, for example linear models with integrality requirements. Furthermore, these methods are not available as standalone solvers and it is not possible to easily transform a model defined in a CP framework into a form that can be solved by one of the introduced algorithms.

The goal of my research is to develop algorithms that can be used to solve combinatorial stochastic optimisation problems modelled in a problem independent, high-level modelling language, such as Stochastic MiniZinc.

2 Background

Multiple methods to solve stochastic programs have been developed in the past. The most direct option to formulate the SOP as one large (deterministic) optimisation problem, called the deterministic equivalent. The DE contains a set of variables for the first stage, and a copy of the second stage variables for each scenario. As a result, the DE does not scale well with an increased number of scenarios. The objective is to minimise the cost of the first stage plus the expected cost of the second stages.

2.1 Decomposition Methods

An alternative to the DE is to decompose the SOP, in one of two ways. The problem can either be relaxed by time stages, where a master problem describes the first stage and contains an approximation of the second stage. This is similar to *Benders decomposition* and named *L-Shaped* method in the con-

text of SOP. Complete assignments to the master problem are evaluated against the sub-problems. Secondly, the problem can be decomposed by scenarios. For each scenario, a copy of the first stage variables is introduced. To ensure feasibility, additional consistency constraints that enforce the first stage variables to be identical across all scenarios are added. The consistency constraints can be violated, yielding a relaxation of the SOP. Methods that use the scenario decomposition are Progressive Hedging (PH) and the dual decomposition (DD) for a summary see [Hemmi *et al.*, 2017].

3 Contributions

This section introduces first Ahmed's [Ahmed, 2013] work that provides the foundation for my contribution that is introduced after.

Evaluate & Cut

Ahmed [Ahmed, 2013] proposes a scenarios decomposition algorithm for SOP's with a binary first stage that works independently of the second stage structure. Every scenario is modelled as an independent deterministic optimisation problem by relaxing the consistency constraints. The algorithm, first solves each scenario independently to optimality, yielding a lower bound and a set of candidates. A candidate is a first stage variable assignment that can be evaluated against all scenarios, i.e. the first stage assignment of a specific scenario solution. Secondly, each candidate is evaluated, by projecting its variable assignment onto the first stage variables of all scenarios and solving them again. This yields a feasible solution to the SOP, as the consistency constraints are satisfied, and an upper bound. Thirdly, by means of *candidate nogoods / cuts*, the evaluated candidates are cut off from the search, by adding a nogood constraint to each scenario. Repeating this three steps is guaranteed to find the optimal solution and terminate once the lower bound meets the upper bound, as the first stage variables have finite support.

Diving

Evaluate & Cut can be applied to a wide range of problems. An optimality gap is provided during the search and according to Ahmed, early candidate solutions are likely to be high quality solutions for the SOP. However, the time to proof optimality can be long, as each candidate nogood only cuts off solution at time. On top of that, the number of evaluation steps required in each iteration is quadratically correlated with the number of scenarios, as the scenarios potentially have distinct solutions, and all the candidates have to be evaluated against all scenarios. In [Hemmi *et al.*, 2017], I have proposed a framework called *Diving*, that generates strong nogoods to prune multiple solutions, without the need to evaluate all candidates, while still complete (able to proof optimality).

After each iteration in Evaluate & Cut the algorithm enters a diving loop with the aim to find *partial nogoods*. In contrast to a candidate nogood, a partial nogood cuts off multiple solutions at once. Consider an SOP with 5 first stage variables $x_1 - x_5$. A candidate nogood contains all first stage variables, e.g. $c_c = \{x_1 \neq 3 \vee x_2 \neq 6 \vee x_3 \neq 1 \vee x_4 \neq 8 \vee x_5 \neq 3\}$. The added constraint cuts off exactly one solution. A nogood composed of only a *subset* of first stage variables would be much stronger. For example, assume that we can prove that even the *partial* assignment $x_1 = 3 \wedge x_2 = 6 \wedge x_3 = 1$ cannot be completed to an optimal solution to the stochastic problem.

During diving, a set of consistency constraints is enforced, e.g. $x_1 = 3 \wedge x_2 = 6 \wedge x_3 = 1$. Then, every scenario is solved independently, yielding a temporary lower bound. The lower bound is temporary as consistency constraints are introduced. If the temporary lower bound exceeds the upper bound, the partially consistent solution can no be extended to a completely consistent first stage assignment that is better that the incumbent. As a results, a partial nogood, excluding all first stage assignments that are a superset of the partial assignment, is added to the scenarios. If the temporary lower-bound remains below the upper bound, an additional consistency constraint is enforced.

4 Future Work

Computational experiments in [Hemmi *et al.*, 2017] have shown that diving can increase the performance of Evaluate & Cut substantially. The heuristic to select the partial assignments to be evaluated strongly impacts the performance of diving. Preliminary results on improved selection heuristics have shown to be effective. Over the coming month, I will be implementing improved selection heuristics as well as additional benchmarks, to strengthen my findings. Furthermore, I will extend the diving framework for multi-stage problems as well as chance constraint optimisation problems. A chance constraint denotes a constraint that must be satisfied across a subset of scenarios.

References

[Ahmed, 2013] Shabbir Ahmed. A scenario decomposition algorithm for 0–1 stochastic programs. *Operations Research Letters*, 41(6):565–569, 2013.

[Albareda-Sambola *et al.*, 2006] Maria Albareda-Sambola, Maarten H Van Der Vlerk, and Elena Fernández. Exact solutions to a class of stochastic generalized assignment problems. *European journal of operational research*, 173(2):465–487, 2006.

[Birge and Louveaux, 2011] John R Birge and Francois Louveaux. *Introduction to stochastic programming*. Springer Science & Business Media, 2011.

[Hemmi *et al.*, 2017] David Hemmi, Guido Tack, and Mark Wallace. Scenario-based learning for stochastic combinatorial optimisation. In Springer, editor, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, 2017.

[Nethercote *et al.*, 2007] Nicholas Nethercote, Peter J Stuckey, Ralph Becket, Sebastian Brand, Gregory J Duck, and Guido Tack. Minizinc: Towards a standard cp modelling language. In *Principles and Practice of Constraint Programming*, pages 529–543. Springer, 2007.

[Rendl *et al.*, 2014] Andrea Rendl, Guido Tack, and Peter J Stuckey. Stochastic minizinc. In *Principles and Practice of Constraint Programming*, pages 636–645. Springer, 2014.