Stochastic Constraint Programming with And-Or Branch-and-Bound

Behrouz Babaki[1], **Tias Guns**[1,2], **Luc De Raedt**[1]
[1]Department of Computer Science, KU Leuven, Belgium
[2]Department of Business Technology and Operations, VUB, Belgium

Abstract

Complex multi-stage decision making problems often involve uncertainty, for example, regarding demand or processing times. Stochastic constraint programming was proposed as a way to formulate and solve such decision problems, involving arbitrary constraints over both decision and random variables. What stochastic constraint programming currently lacks is support for the use of factorized probabilistic models that are popular in the graphical model community. We show how a state-of-the-art probabilistic inference engine can be integrated into standard constraint solvers. The resulting approach searches over the And-Or search tree directly, and we investigate tight bounds on the expected utility objective. This significantly improves search efficiency and outperforms scenario-based methods that ground out the possible worlds.

Introduction

Increasingly, complex decision making requires one to make *decisions* under *constraints* while taking into account the *uncertainty* of the environment. Each of these aspects has intensively been studied by different communities within artificial intelligence. Indeed, constraint programming has focussed on solving constraint satisfaction problems and making decisions while the field uncertainty in artificial intelligence is concerned with probabilistic graphical models and inference. For each of these problems, advanced solutions have been developed and solvers exist that can tackle substantial problems. But today, there is a growing awareness that in many real-life applications, these aspects cannot be addressed in isolation, but rather need to be tackled by an integrated approach. Stochastic constraint programming [Walsh, 2002; Tarim *et al.*, 2006] covers all three aspects as it extends constraint programming with decision making under uncertainty. However, such methods do not yet support standard probabilistic techniques from the graphical model community [Koller and Friedman, 2009]. It is well-known in probabilistic graphical models that factorizing the joint probability distribution is beneficial for modeling, inference and for learning [Koller and Friedman, 2009]. Stochastic constraint programming currently uses trivial factorizations, assuming either that all random variables are marginally independent [Walsh, 2002], or using the joint as the only factor [Tarim *et al.*, 2006]. The latter corresponds to enumerating all possible worlds, also called scenarios. On the other hand, [Mateescu and Dechter, 2008] have integrated constraint programming and probabilistic graphical models, but do not deal with decisions and utilities; and influence diagrams [Jensen *et al.*, 1994b] integrate probabilistic graphical models with decision theory, but do not handle constraints.

Our contribution is as follows:

- We support stochastic constraint programming with factorized joint probability distributions (as in Bayesian networks) and integrate state-of-the-art inference engines for such graphical models.
- We use a generic constraint solver both for the deterministic constraints and for doing constrained branch-and-bound search over an and-or tree.
- We develop and exploit a novel bound for *expected* utility in the search. The key is that we use a probabilistic inference to compute marginal probabilities and interval arithmetic for the utility.

We now introduce the problem and review the two standard approaches, after which we explain and evaluate our method.

Stochastic Constraint Programming

We consider multi-stage decision problems where a number of decisions can be taken (such as the production amount) after which external factors are observed (such as the demand) followed by new decisions etc. The goal is to assign in a stage-wise manner the decision variables so that the expected utility over all possible instantiations of the random variables is maximized. To model the stochastic aspect of the external factors we use random variables with a (joint) discrete probability distribution P.

Example 1. Consider a simple stochastic production problem from [Walsh, 2002] where each stage corresponds to one quarter of the year, and the decisions V_i are how many books to produce at the start of quarter i, and the random variables S_i represent how many books were sold at the end of quarter i. The company is conservative so shortages are not allowed, yet the goal is to minimize the sum of surpluses in each quarter due to stocking costs.

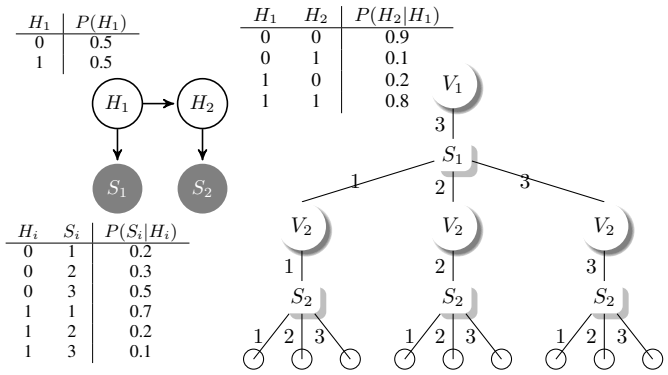


Figure 1: Left: Bayesian network with 2 observed and 2 hidden variables. Right: a tree for Example 1 with $\mathcal{D}(V_1) = \mathcal{D}(S_1) = \mathcal{D}(V_2) = \mathcal{D}(S_2) = \{1, 2, 3\}$.

More formally for the 2 stage variant:

$$\text{minimize} \quad \min_{V_1} \sum_{S_1} \min_{V_2} \sum_{S_2} P(\mathcal{S}) \times U(\mathcal{V}, \mathcal{S})$$

$$s.t. \quad \sum_{j=1}^{i} (V_j - S_j) \geq 0 \qquad \forall i = 1..2$$

where $U(\mathcal{V}, \mathcal{S}) = V_1 - S_1 + V_2 - S_2$

Factored distributions

We will assume that the probability distribution P is specified as a factored distribution, that is, the probability can be computed as a product of individual factors [Koller and Friedman, 2009]. One popular such representation is a Bayesian network.

An example Bayesian network is shown in Figure 1, left. It has two observed variables (factors) S_1 and S_2, which would typically be two random variables present in the stochastic problem formulation. The Bayesian network is a hidden Markov model with 2 hidden variables (factors) where the probability of observation is influenced by the hidden variables (e.g. market sentiment), and the second hidden variable is influenced by the hidden variable of the previous stage. Such a rich structure models complex interactions, and supports learning them from observations.

Problem Description

We define a multi-stage *Factored Stochastic Constraint Problem* (FSCP) as a 7-tuple $\mathcal{P} = \langle \mathcal{V}, \mathcal{S}, \mathcal{D}, \mathcal{P}, U, \mathcal{C}, \prec \rangle$ where:

- $\mathcal{V}$ and $\mathcal{S}$ are decision variables and random (*s*tochastic) variables, respectively;
- $\mathcal{D}$ is the domain of variables in $\mathcal{V} \cup \mathcal{S}$, namely a mapping from variable to the set of values it can take;
- $\mathcal{P}$ is a factored discrete distribution over $\mathcal{S}$, i.e. $P(\mathcal{S}) = \prod_{\mathcal{S}_i \subset \mathcal{S}} \phi(\mathcal{S}_i)$ with each factor $\phi(\mathcal{S}_i)$ over a subset of $\mathcal{S}$. As in previous works, we assume that the decision variables have no measurable impact on the probability distribution.
- $U(\mathcal{V}, \mathcal{S})$ is a function that computes the *utility* of an instantiation of the decision and random variables.
- $\mathcal{C}$ is a set of deterministic constraints. Each constraint is specified over a non-empty subset of $\mathcal{V}$ and a (possibly empty) subset of $\mathcal{S}$.
- $\prec$ is a partial ordering over $\mathcal{V} \cup \mathcal{S}$ that orders the variables by stage, and within each stage the decision variables before the random variables.

For notational convenience and without loss of generality we will assume one decision variable V_i and one random variable S_i per stage i: $V_1 \prec S_1 \prec \ldots \prec V_T \prec S_T$.

The objective is to maximize (or similarly minimize) the *expected utility* of the multi-stage problem according to $\prec$:

$$\max_{V_1} \sum_{S_1} \max_{V_2} \sum_{S_2} \ldots \max_{V_T} \sum_{S_T} P(S_1, \ldots, S_T) \times U(V_1, \ldots, V_T, S_1, \ldots, S_T) \quad (1)$$

where we note that sum and max are not transitive and hence can not be reordered in this formula. Constraints can range over random variables but are deterministic: they must be satisfied for all possible (non-0 probability) instantiations of the random variables.

An assignment to the variables $\mathcal{V} \cup \mathcal{S}$ that satisfies all constraints is not a solution to the FSCP, rather, it holds only with probability $P(\mathcal{S})$ and hence contributes $P(\mathcal{S}) * u(\mathcal{V}, S)$ to the expected utility. Indeed, the solution is a *policy tree* [Walsh, 2002] where each node corresponds to a variable and for each path in the tree the variables satisfy the ordering $\prec$. Each decision variable V_i has just one child (corresponding to an assignment of this variable) and each random variable $\mathcal{S}_i$ has a child for *each* value in its domain. Figure 1 (right) shows a policy tree for the problem of Example 1, where 1,2 or 3 thousand books can be sold per stage.

An *optimal* policy tree is a policy tree where each decision variable in the tree is assigned to a value such that Eq. (1) is maximized, while always satisfying all constraints.

Scenario-based Search

One approach [Tarim *et al.*, 2006] is to ground out each of the possible worlds and compute their probability, and at the same time flatten the policy tree by creating copies of each decision variable for all instantiations of the random variables preceding it. One assignment to this set of decision variables then corresponds to a policy tree.

For example for the policy tree in Figure 1 (right) we would create a decision variable for every decision node in the tree, so 4 variables in total: one for V_1 and three different ones for V_2, corresponding to cases $S_1 = 1, S_1 = 2$ and $S_1 = 3$.

Constraints are added over these decision variables as needed, and the expected utility function becomes a linear sum over all possible worlds $\sum_s P(s) * u(\mathcal{V}, s)$ with s a possible world (also called scenario). For the example in Figure 1 (right), there would be $3 * 3$ scenarios corresponding to the possible worlds.

An obvious downside of this approach is that the number of decision variables and the number of scenarios grows exponentially in the number of random variables, with the base of the exponent determined by the number of possible values for the random variables.

And-Or Search

Initially, a simple And-Or search algorithm (plain backtracking and forward checking) for stochastic constraint programming was proposed [Walsh, 2002].

The And-Or search tree has two types of internal nodes: the AND nodes correspond to random variables, and the OR nodes correspond to the decision variables. The leaves do not correspond to a variable. An outgoing edge from an internal node represents the assignment of a value to the variable associated with that node.

Every path from the root to a node corresponds to a partial assignment to $\mathcal{V} \cup \mathcal{S}$, and must respect the ordering $\prec$. Given an assignment, we denote by v_i the value of variable $V_i \in \mathcal{V}$ in the assignment. We label each node n by the partial assignment $(v_1, s_1, \ldots, v_k, s_k)$ represented by the path from the root to this node and denote it by $label(n)$. The *value* of a labeled node n is then defined as follows:

$$val(v_1,s_1,\ldots,v_k,s_k) = \max_{V_{k+1}} \sum_{S_{k+1}} \ldots \max_{V_T} \sum_{S_T} \quad (2)$$
$$P(s_{1:k}, S_{k+1:T}) \times U(v_{1:k}, V_{k+1:T}, s_{1:k}, S_{k+1:T})$$

where we use $v_{1:k}$ as shorthand for $(v_1, \ldots, v_k)$. The value of a label $(v_1, s_1, \ldots, v_k)$ ending in a decision variable is defined analogously. The notation follows [Jensen *et al.*, 1994a].

Observe how the expression in Eq. (1) corresponds to the above value function when none of the variables are assigned, that is, the value of the root node of the And-Or tree.

The value of any node n in the And-Or tree can be computed recursively as follows:

1. The value of a leaf with $label(n) = (v_{1:T}, s_{1:T})$ is $P(s_{1:T}) \times U(v_{1:T}, s_{1:T})$
2. If n corresponds to a random variable, then $val(label(n)) = \sum_{n' \in \text{children}(n)} val(label(n'))$
3. If n corresponds to a decision variable, then $val(label(n)) = \max_{n' \in \text{children}(n)} val(label(n'))$

A generic depth-first search procedure to recursively compute this function, while also ensuring satisfaction of all constraints in non-0 probability worlds, is shown in Algorithm 1. The symbol $\mathcal{D}$ represents the *domain*, that is, a mapping from variables to the values they can take.

First, the AndOr() procedure on line 2 does *propagation*, which is the act of removing those values from the domain that would violate a constraint. If all variables are assigned (their domain has size 1), then the value of this leaf is computed and returned. Next, on line 6, the variable to expand in this node is selected in such a way that the order $\prec$ is respected and the value for each of the children in the domain is recursively computed. In case of an AND node, first one has to verify that all children (not just those with a value in the domain) were visited and did not fail (line 13) because all possible worlds must be possible in a policy tree. If so, the sum of child values is returned. For OR nodes the maximum of the child values is returned.

Algorithm 1 And-Or search over domain $\mathcal{D}'$ following $\prec$

```
1:  procedure ANDOR(D')
2:      if propagate(D') == failure and probability > 0 then return failure
3:      if ∀x ∈ V ∪ S : |D'(x)| = 1 then                    ▷ In leaf
4:          return val(label(D'))
5:      end if
6:      Select unassigned variable X according to ≺
7:              ▷ Expand this node by assigning values to X
8:      for x ∈ D'(X) do        ▷ For all children of this node
9:          D'' := D' and set D''(X) := {x}
10:         childval_x := AndOr(D'')
11:     end for
12:     if X ∈ S then                                       ▷ AND node
13:         if one of the children failed then return failure
14:         else return Σ_{x∈D(X)} childval_x
15:     else                                                ▷ OR node
16:         return max_{x∈D(X)} childval_x
17:     end if
18: end procedure
```

Method: branch-and-bound And-Or search

We improve on the above And-Or search in the following two ways, which requires the probabilities of partial assignments:

- before exploring a node in the tree, we verify that the probability of the assignment to the stochastic variables explored so far (a partial assignment) is not 0. If it is, then all leaves that are descendants of this node will have 0 probability and hence the value of this node will always be 0 and it should not be explored. This was studied in [Qi and Poole, 1995] where it was shown that even a naive and-or search can be sped up significantly in case of determinism (0 probabilities) in the graphical model.
- we compute upper bounds on the expected utility achievable by a node in the and-or tree to prune the search.

Querying the probability of partial assignments

The probability of a partial assignment can be obtained through marginalization; this has been studied for many years by the uncertainty in AI community [Pearl, 1989]. Given a distribution over T variables, the marginal probability of a subset of k variables is obtained by marginalizing out all other variables: $P(S_1, ..., S_k) = \sum_{S_{k+1}} \ldots \sum_{S_T} P(S_1, ..., S_T)$.

Probabilistic inference methods can efficiently exploit structure and determinism in the factored distribution when queried for a marginal probability. To take advantage of this during search, we integrate an existing query inference engine into our approach.

The characteristics of our queries are that: 1) we will query the engine many times during search (at every node in the tree); 2) our queries will always contain random variables satisfying the order $\prec$, following the depth-first search; 3) we wish to obtain the partial probabilities of all possible values of a random variable at once. Motivated by this, we chose the ACE engine [Chavira and Darwiche, 2008] as query engine, because 1) it first compiles the Bayesian network into an arithmetic circuit (AC). While this circuit may have worst-case exponential size, once the AC is compiled computing a (marginal) probability takes time linear in the size of the AC. In practice, the size of the AC is often reasonably

small making this approach very attractive to many applications [Chavira and Darwiche, 2005]; 2) the ACE engine allows to incrementally *commit* and *retract* assignments which fits the depth-first search procedure well and prevents the engine from traversing the AC from scratch each time; 3) it can return, at low extra cost, the probabilities for all values of a random variable. For more details on the inference engine, see [Darwiche, 2003].

Interval arithmetic for the utility

The computation of our novel bounds during the branch-and-bound And-Or search relies on the following key ideas: to query the inference engine for an upper bound on the probability of a partial assignment, to use *interval arithmetic* [Moore, 1966] to get an upper-bound on the utility of a partial assignment and to combine this into an upper bound on the value of any node in the And-Or tree.

The basic idea of interval arithmetic is that, given an equation over intervals, using just the upper and lower bounds one can derive an upper and lower bound on the outcome of the equation [Moore, 1979]. For example given $f(X,Y) = 3X - Y$, then using interval arithmetic we can derive that $f([X_{min}, X_{max}], [Y_{min}, Y_{max}]) = [3X_{min} - Y_{max}, 3X_{max} - Y_{min}]$ where $[\cdot,\cdot]$ represents the minimum and maximum value of an interval. Interval arithmetic is a technique that constraint solvers often use internally to obtain bounds-consistency [Choi *et al.*, 2006].

Given a partial assignment $(v_1, s_1, \ldots, v_k, s_k)$, we wish to derive an upper bound on the value of $U(v_{1:k}, V_{k+1:T}, s_{1:k}, S_{k+1:T})$ over all possible assignments to $V_{k+1:T}$ and $S_{k+1:T}$. We define the upper bound on the utility as: $\bar{U}(v_{1:k}, s_{1:k}) = \max_{V_{k+1}} \max_{S_{k+1}} \ldots \max_{S_T} U(v_{1:k}, V_{k+1:T}, s_{1:k}, S_{k+1:T})$. The bound is obtained by applying interval arithmetic to U with interval $[v_i, v_i]$ for each assigned variable and $[min(\mathcal{D}(V_i)), max(\mathcal{D}(V_i))]$ for each unassigned variable V_i; likewise for the S_i. For simple utilities, as is the case here, this interval bound computation can be provided by hand.

Shallow upper bound

Theorem 1. For each partial assignment $label(n) = (v_1, s_1, \ldots, v_k, s_k)$, we have:

$$val(v_1, s_1, \ldots, v_k, s_k) \leq P(s_{1:k}) \times \bar{U}(v_{1:k}, s_{1:k}) \qquad (3)$$

Proof.

$$\begin{aligned}
&\max_{V_{k+1}} \sum_{S_{k+1}} \ldots \max_{V_T} \sum_{S_T} P(s_{1:k}, S_{k+1:T}) \\
&\qquad \times U(v_{1:k}, V_{k+1:T}, s_{1:k}, S_{k+1:T}) \\
&\leq \max_{V_{k+1}} \sum_{S_{k+1}} \ldots \max_{V_T} \sum_{S_T} P(s_{1:k}, S_{k+1:T}) \times \bar{U}(v_{1:k}, s_{1:k}) \\
&= \bar{U}(v_{1:k}, s_{1:k}) \times \max_{V_{k+1}} \sum_{S_{k+1}} \ldots \max_{V_T} \sum_{S_T} P(s_{1:k}, S_{k+1:T}) \\
&= \bar{U}(v_{1:k}, s_{1:k}) \times \sum_{S_{k+1}} \ldots \sum_{S_T} P(s_{1:k}, S_{k+1:T}) \\
&= \bar{U}(v_{1:k}, s_{1:k}) \times P(s_{1:k})
\end{aligned}$$

□

Hence, by efficiently querying for the probability $P(s_{1:k})$ and computing $\bar{U}(v_{1:k}, s_{1:k})$, we obtain an upper bound on the value of this node.

Pruning OR nodes. This upper bound can be used to prune children of an OR node, as only the child with the maximum value is sought. In the search, every OR node will store as lower bound the value of its best child so far. If the upper bound of the next child to explore is below this lower bound, the child does not need to be visited and can hence be pruned.

Pruning AND nodes. We observed that sometimes it is possible to prune an AND node before all its children have been explored. In those cases, the values of the children already visited are too low for the AND node to improve its closest parent OR node.

Assume that every AND node can receive from its parent what the minimum value is that it should achieve, that is, its lower bound LB. Let 'Ch' be the set of children of an AND node and let 'Pre' be the children already explored during search. Then, the following needs to hold:

$$LB \leq \sum_{i \in \text{Ch}} val(label(i)) \qquad (4)$$

$$\rightarrow LB \leq \sum_{i \in \text{Pre}} val(label(i)) + \sum_{i \in \text{Ch} \setminus \text{Pre}} UB(i) \qquad (5)$$

where $UB(i)$ is the upper bound on child i computed using Theorem 1. Hence, after exploring a child of an AND node, we can verify whether Eq. (5) holds and if it doesn't we do not need to visit the remaining children.

For this to work, all nodes must be able to pass a lower bound LB to its children. For an OR node, the lower bound of a child is simply the lower bound of the OR node itself. For AND nodes, we can derive from Eq. (5) the following lower bound on an unvisited child c:

$$LB - \sum_{i \in \text{Pre}} val(label(i)) - \sum_{i \in \text{Ch} \setminus (\text{Pre} \cup \{c\})} UB(i) \leq val(label(c))$$

hence the value of child node c needs to be larger than the left-hand side of this equation.

Deep upper bound

A tighter upper bound on the value of a node can be computed by realizing that the summation of an AND node S_k corresponds to a weighted sum of the children's value, weighted by their (conditional) probability. Hence, we can obtain a tighter bound by computing the probability and upper bound on the utility for each child of this AND node separately:

$$val(v_1, s_1, \ldots, v_k) \leq \sum_{S_k} P(s_{1:k-1}, S_k) \times \bar{U}(v_{1:k}, s_{1:k-1}, S_k)$$

The same reasoning is valid for any sequence of unassigned random variables, that is, up to any depth:

Theorem 2. Given the partial assignment $(v_1, s_1, \ldots, v_k)$ and an arbitrary *depth* d such that $k + d \leq T$, we have:

$$\begin{aligned}
&val(v_1, s_1, \ldots, v_k) \\
&\leq \sum_{S_k} \ldots \sum_{S_{k+d}} P(s_{1:k-1}, S_{k:k+d}) \times \bar{U}(v_{1:k}, s_{1:k-1}, S_{k:k+d})
\end{aligned}$$

Proof.

$$\sum_{S_k} \max_{V_{k+1}} \dots \sum_{S_T} P(s_{1:k-1}, S_{k:T}) \times U(v_{1:k}, V_{k+1:T}, s_{1:k-1}, S_{k:T})$$

$$\leq \sum_{S_k} \max_{V_{k+1}} \dots \sum_{S_{k+d}} P(s_{1:k-1}, S_{k:k+d}) \times \bar{U}(v_{1:k}, V_{k+1:k+d}, s_{1:k-1}, S_{k:k+d})$$

$$\leq \sum_{S_k} \max_{V_{k+1}} \dots \sum_{S_{k+d}} P(s_{1:k-1}, S_{k:k+d}) \times \bar{U}(v_{1:k}, s_{1:k-1}, S_{k:k+d})$$

$$= \sum_{S_k} \sum_{S_{k+1}} \dots \sum_{S_{k+d}} P(s_{1:k-1}, S_{k:k+d}) \times \bar{U}(v_{1:k}, s_{1:k-1}, S_{k:k+d})$$

the last step is possible because the remaining P and $\bar{U}$ computation is independent of the assignments to $V_{k+1:T}$, as it was taken out of $\bar{U}$ in the step before. □

This bound can be recursively computed by a simple depth-first search over the assignments to the next d unassigned random variables. At depth d of the recursive computation, the probability $P(s_{1:k-1}, S_{k:k+d})$ is queried and the upper bound on the utility is computed. Even if all probabilities of all random variables are equal, this bound will be tighter than the shallow bound thanks to the utility being computed for the actual value of the random variables explored. The complexity of this computation is exponential with respect to the depth.

And-Or search in a constraint solver

A strong argument in favor of scenario-based search in [Tarim *et al.*, 2006] is the possibility to use any existing constraint solver, and hence the expressivity and the constraints available in such solvers. We also support this argument and use a generic constraint solver for our And-Or search and the constraints. The key issue is that such a solver performs depth-first backtracking search but is oblivious to nodes being AND or OR nodes. What it considers a *solution*, namely an assignment to all of the variables, is just a leaf in the And-Or tree.

However this is not a fundamental issue, as constraint solvers are inherently *modular* depth-first search engines. To obtain a correctly working And-Or search, we added three modules to a constraint solver, which can be added to most if not all modern CP solvers:

1. a *global* constraint that verifies whether no values are removed from the domain of random variables, except by the search method, and otherwise fails because the constraints should be satisfied in all possible worlds;
2. a variable and value ordering module that respects $\prec$, that will fail the remaining children of an AND node if one of its children failed, and that also computes and prunes using the upper bounds, as well as pushing the lower bound to the child nodes;
3. a module that is called each time a complete assignment is found, and that updates the expected utility values in its ancestor nodes appropriately.

At a technical level, we maintain the state of the policy tree in an object that is shared by all three modules and that is not backtracked over by the regular backtracking mechanism. This object also provides an interface to the probabilistic inference engine and caches all queries to increase efficiency.

Experiments

We investigate the following questions: **Q1:** Does our proposed method improve over existing approaches? **Q2:** What is the impact of bounding depth on the efficiency of search? **Q3:** What is the interplay between bounding and constraint propagation?

We used two problems in our experiments:

Knapsack (based on an example from [Hnich *et al.*, 2011]) As items arrive, we must decide whether to pick or leave the item. The weight and cost of an item are stochastic, and only revealed immediately after a decision is made for that item. The weight (5 possibilities) and cost (3 possibilities) in each stage depend on a hidden variable (2 possibilities), which itself depends on the hidden variable of the previous stage. The goal is to maximize the expected sum of values under the constraint that the total weight does not exceed the capacity.

Investment At the start of each season, a company can invest in option **A** or **B**. The stochastic return is revealed at the end of the season. The two return values in each season (4 possibilities for each option) depend on the market sentiment (2 possibilities), which itself depends on the market sentiment in the previous season. Option **A** has a higher return on average, but a major tax relaxation is applied at the end of the horizon if the majority of income comes from investment in **B**. The goal is to maximize the expected returns under the constraint that the tax relaxation applies.

The Bayesian Networks are HMMs similar to Figure 1 but with two observed variables per hidden variable (corresponding to each stage). The hidden variables have 0.9 probability for staying in the same state. Other distribution parameters for both problems were generated randomly, such that at each stage for Knapsack the weight and cost have positive correlation and for Investment the returns have negative correlation. Higher-order HMMs did not impact compilation or runtime much because of the small number of variables/depth.

We ran the experiments on Linux machines with 32 GB of memory. The time-out used was 1800 seconds. The CP solver used is Gecode-4.4.0 and the MIP solver is Gurobi-6.5[1]. We used the the ACE[2] package to compile the Bayesian networks into arithmetic circuits, and ported the inference library to C++ for integration with Gecode. The code and data are available online [3].

Results

To answer *Q1*, we compare our method with a scenario-based approach that copies the decision variables, using both a CP solver and a MIP solver. The latter is possible because both problems only have linear constraints, though our method can handle any CP constraint. Table 1 shows the results. Standard CP quickly fails due to the huge search space and weak pruning. ILP is more effective thanks to its presolving ($> 50\%$ reductions in variables) and cutting planes. However, our native method is faster and can handle larger number of stages.

To answer *Q2*, we compare the runtime of our method for different depths of the bound. Figure 2 shows that even the

[1] http://www.gecode.org and http://www.gurobi.com

[2] http://reasoning.cs.ucla.edu/ace

[3] https://github.com/Behrouz-Babaki/FactoredSCP

	knapsack				investment			
#stages	scen-vars	scen-**CP**	scen-**ILP**	**AOB&B**	scen-vars	scen-**CP**	scen-**ILP**	**AOB&B**
1	1	0.000089	0.000416	0.000120	2	0.000134	0.000392	0.000213
2	16	0.001410	0.002101	0.001291	34	0.000667	0.001199	0.001141
3	241	T(S)	0.013641	0.016774	546	T(S)	0.023160	0.015718
4	3616	T(S)	0.202595	0.19052	8738	T(S)	0.724038	0.220448
5	54241	T(S)	4.092760	4.02639	139810	M(S)	34.09311	5.84844
6	813616	T(S)	117.2034	75.9437	2236962	T(S)	779.4099	162.892
7	12204241	M(G)	M(G)	1494.54	35791394	M(G)	M(G)	T(S)

Table 1: Comparing the runtime (s) of our method (AOB&B) with scenario-based approaches (CP and ILP). The unsuccessful cases either ran out of time (T) or memory (M) during generation of scenario-based problem (G) or solving the problem (S).

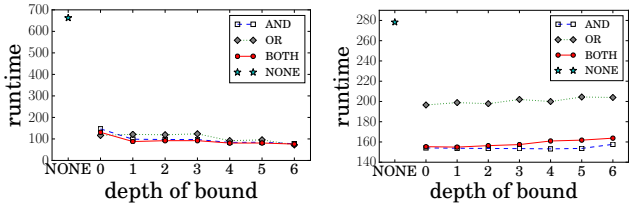
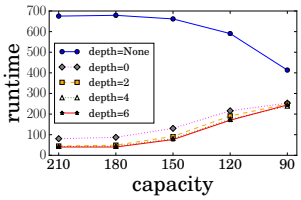


Figure 2: Effect of depth of bounds on instances of the *knapsack* problem (left) and *investment* problem (right).



	no bound		depth=6	
C	**#failures**	**#nodes**	**#failures**	**#nodes**
210	0.0E+00	1.1E+09	8.1E+05	1.8E+07
180	5.9E+04	1.1E+09	8.1E+05	1.8E+07
150	7.6E+05	1.0E+09	2.3E+06	5.4E+07
120	3.2E+06	9.3E+08	6.3E+06	1.5E+08
90	5.7E+06	6.4E+08	9.1E+06	2.4E+08

Figure 3: The effect of tightening the *knapsack* constraint (C) on runtime (left) and on the number of nodes and failures (right) in a 6-stage problem.

shallow bound is much better than no bound. Deeper bounds have a marginal gain in runtime for knapsack and a marginal overhead for investment but always lead to smaller search spaces (not shown). On the investment problem, it is clear that also bounding the AND nodes (or both) is better than only the OR nodes.

To answer *Q3*, we gradually tighten the capacity constraint in the *knapsack* problem and observe the effect on runtime, number of nodes, and number of failures. Figure 3 shows that in the absence of bounds, tightening the constraint leads to more failures and fewer nodes; meaning that the search space becomes smaller. When bounding is employed, tighter constraints increase both the number of nodes and failures. This indicates fewer solutions and weaker bounding. Such interactions demonstrate the need for approaches that can both handle constraints and prune with bounds.

Related work

[Walsh, 2002] also employed And-Or search (backtracking and forward checking) but no bounds, independent random variables and one global stochastic constraint. Nested constraint programming [Chu and Stuckey, 2014] is a related framework in which stochastic CP as well as other *nested* problems can be expressed; their clause learning solver performs and-or search and caches the valuation of identical subproblems/nodes, but it assumes independent random variables. Quantified constraint optimization [Benedetti *et al.*, 2008] nests existential and universal quantification but does not consider probability distributions. Arbitrary non-factored probability distributions have been considered before in a scenario-based setting [Tarim *et al.*, 2006]. Global chance constraints have been investigated in that setting too [Hnich *et al.*, 2012]. Another work investigates relaxation methods for convex expected utility functions [Rossi *et al.*, 2008]; see [Hnich *et al.*, 2011] for a survey on such methods.

FSCPs are also related to influence diagrams [Jensen *et al.*, 1994b] but are different in that the utility function in influence diagrams is assumed to be additive and that (F)SCPs support arbitrary complex constraints over both decision and random variables; The typical jointree algorithms [Jensen *et al.*, 1994b] can have prohibitive memory requirements. A depth-first branch-and-bound algorithm was investigated [Yuan *et al.*, 2010] but the bound uses (simplified) influence diagrams itself. Other connections between constraint programs and probabilistic graphical exist. Notably probabilistic queries on *mixed deterministic and probabilistic networks* [Mateescu and Dechter, 2008] and a more theoretical, unifying framework of algebraic graphical models [Pralet *et al.*, 2007].

Conclusion and future work

We presented a novel stochastic constraint programming method with three distinguishing features: a novel bound that works on the And-Or search space directly; the use of (non-trivial) factorized probability distributions and querying during search using a state-of-the-art inference engine from the UAI community; and within a generic constraint solver meaning that any existing global constraint can be used. This allows to reason over larger problems for which grounding out all possible worlds is not feasible or incurs too much overhead. Our CP-based method can handle complex constraints and not just linear/convex constraints as MIP solvers do.

In the future, we aim to investigate a generic mechanism for global *chance constraints* [Hnich *et al.*, 2012], which can be violated in a small amount of possible worlds [Walsh, 2002]. An option is also to to find approximate solutions by not exploring worlds with a very small probability/expected utility. A last promising avenue is to reason over probability distributions that are influenced by the *decisions* too, where our method has the advantage that it reasons over the (non-ground) problem structure directly.

References

[Benedetti *et al.*, 2008] Marco Benedetti, Arnaud Lallouet, and Jérémie Vautard. *Quantified Constraint Optimization*, pages 463–477. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[Chavira and Darwiche, 2005] Mark Chavira and Adnan Darwiche. Compiling bayesian networks with local structure. In *IJCAI*, pages 1306–1312. Professional Book Center, 2005.

[Chavira and Darwiche, 2008] Mark Chavira and Adnan Darwiche. On probabilistic inference by weighted model counting. *Artif. Intell.*, 172(6-7):772–799, 2008.

[Choi *et al.*, 2006] Chiu Wo Choi, Warwick Harvey, Jimmy Ho-Man Lee, and Peter J Stuckey. Finite domain bounds consistency revisited. In *Australasian Joint Conference on Artificial Intelligence*, pages 49–58. Springer, 2006.

[Chu and Stuckey, 2014] Geoffrey Chu and Peter J. Stuckey. Nested constraint programs. In *Principles and Practice of Constraint Programming - 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings*, pages 240–255, 2014.

[Darwiche, 2003] Adnan Darwiche. A differential approach to inference in bayesian networks. *J. ACM*, 50(3):280–305, 2003.

[Hnich *et al.*, 2011] Brahim Hnich, Roberto Rossi, S. Armagan Tarim, and Steven Prestwich. A survey on CP-AI-OR hybrids for decision making under uncertainty. In Pascal van Hentenryck and Michela Milano, editors, *Hybrid Optimization: The Ten Years of CPAIOR*, pages 227–270. Springer, 2011.

[Hnich *et al.*, 2012] Brahim Hnich, Roberto Rossi, S. Armagan Tarim, and Steven Prestwich. Filtering algorithms for global chance constraints. *Artificial Intelligence*, 189:69 – 94, 2012.

[Jensen *et al.*, 1994a] Frank Jensen, Finn V. Jensen, and Sren L. Dittmer. From influence diagrams to junction trees. In *PROCEEDINGS OF THE TENTH CONFERENCE ON UNCERTAINTY IN ARTIFICIAL INTELLIGENCE*, pages 367–373. Morgan Kaufmann, 1994.

[Jensen *et al.*, 1994b] Frank Jensen, Finn Verner Jensen, and Søren L. Dittmer. From influence diagrams to junction trees. In *UAI*, pages 367–373. Morgan Kaufmann, 1994.

[Koller and Friedman, 2009] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models - Principles and Techniques*. MIT Press, 2009.

[Mateescu and Dechter, 2008] Robert Mateescu and Rina Dechter. Mixed deterministic and probabilistic networks. *Ann. Math. Artif. Intell.*, 54(1-3):3–51, 2008.

[Moore, 1966] Ramon E. Moore. *Interval Analysis*. Prentice-Hall, Englewood Cliffs, N.J., 1966.

[Moore, 1979] Ramon E Moore. *Methods and applications of interval analysis*. SIAM, 1979.

[Pearl, 1989] Judea Pearl. *Probabilistic reasoning in intelligent systems - networks of plausible inference*. Morgan Kaufmann series in representation and reasoning. Morgan Kaufmann, 1989.

[Pralet *et al.*, 2007] Cédric Pralet, Gérard Verfaillie, and Thomas Schiex. An algebraic graphical model for decision with uncertainties, feasibilities, and utilities. *J. Artif. Int. Res.*, 29(1):421–489, August 2007.

[Qi and Poole, 1995] Runping Qi and David Poole. A new method for influence diagram evaluation. *Computational Intelligence*, 11(3):498–528, 1995.

[Rossi *et al.*, 2008] Roberto Rossi, S. Armagan Tarim, Brahim Hnich, and Steven Prestwich. *Cost-Based Domain Filtering for Stochastic Constraint Programming*, pages 235–250. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[Tarim *et al.*, 2006] Armagan Tarim, Suresh Manandhar, and Toby Walsh. Stochastic constraint programming: A scenario-based approach. *Constraints*, 11(1):53–80, 2006.

[Walsh, 2002] Toby Walsh. Stochastic constraint programming. In *ECAI*, pages 111–115. IOS Press, 2002.

[Yuan *et al.*, 2010] Changhe Yuan, XiaoJian Wu, and Eric A. Hansen. Solving multistage influence diagrams using branch-and-bound search. In *UAI*, pages 691–700. AUAI Press, 2010.