**Erratum:** This is an updated version of the paper (updated 5/2019). A script preparing challenge problems in the IPC 2018 (specifically the script that compiles enum-valued into binary fluents which are used by our system) had an error. This was corrected and the numerical scores/results in Table 1 were updated. The discussion of the results and the conclusions are not affected by this.

# Stochastic Planning with Lifted Symbolic Trajectory Optimization

**Hao Cui**
Tufts University
Hao.Cui@tufts.edu

**Thomas Keller**
University of Basel, Switzerland
tho.keller@unibas.ch

**Roni Khardon**
Indiana University, Bloomington
rkhardon@iu.edu

## Abstract

This paper investigates online stochastic planning for problems with large factored state and action spaces. One promising approach in recent work estimates the quality of applicable actions in the current state through aggregate simulation from the states they reach. This leads to significant speedup, compared to search over concrete states and actions, and suffices to guide decision making in cases where the performance of a random policy is informative of the quality of a state. The paper makes two significant improvements to this approach. The first, taking inspiration from lifted belief propagation, exploits the structure of the problem to derive a more compact computation graph for aggregate simulation. The second improvement replaces the random policy embedded in the computation graph with symbolic variables that are optimized simultaneously with the search for high quality actions. This expands the scope of the approach to problems that require deep search and where information is lost quickly with random steps. An empirical evaluation shows that these ideas significantly improve performance, leading to state of the art performance on hard planning problems.

## Introduction

This paper investigates online stochastic planning for problems with factored state and action spaces. Such problems arise, for example, when multiple units can operate in parallel to achieve a joint objective. These problems are often modeled using factored Markov decision processes (MDP) where a state is specified by a set of variables, and similarly for actions, leading to combinatorially large state and action spaces. Some of the best recent methods for such problems are based on real-time heuristic search (Kolobov et al. 2012; Keller and Helmert 2013).

A related but distinct effort is given by the SOGBOFA algorithm (Cui and Khardon 2016), which extends the well-known rollout algorithm (Tesauro and Galperin 1996). The rollout algorithm performs simulations that start with a given action and are followed by a predefined policy. The quality of each applicable action is estimated by computing the average of the cumulative reward of multiple simulations that start with that action. SOGBOFA builds an explicit computation graph that allows it to improve over the rollout algorithm in two important ways. First, it replaces multiple simulations over concrete states and actions with a single symbolic aggregate simulation with random rollouts. Second, the explicit computation graph allows the usage of automatic differentiation and therefore gradient search can be used to select the best action, avoiding the need to enumerate actions.

This allows SOGBOFA to plan in large scale planning problems with factored state and action spaces where action enumeration is infeasible. This approach shows excellent performance on problems where the aggregate simulation provides a significant speedup over concrete search and where aggregated information offers good guidance for online planning. On the other hand, although the simulation is in aggregate, the algorithm still works with a grounded problem graph and does not take full advantage of problem structure. In addition, as shown by Cui et al. (2015), in some domains random rollouts as used by SOGBOFA are not informative leading to bad planning performance.

We introduce two new ideas that show how these issues can be addressed while still maintaining the advantage of aggregate simulation. Our first contribution is motivated by the recent observation (Cui, Marinescu, and Khardon 2018) that aggregate simulation is closely related to belief propagation (BP). Using this we propose a lifted version of SOGBOFA, taking inspiration from lifted BP. The idea in lifted BP is to restructure the computation to avoid repeated messages in the algorithm. We show that the same effect is obtained in SOGBOFA using a simple modification of the construction of the computation graph. Thus in our case lifting is easy to implement and has low overhead, allowing us to take advantage of structure when it exists without significant overhead in the general case.

The second contribution introduces the idea of conformant approximation for aggregate simulation. The quality of the approximation of SOGBOFA is limited by the fact that it rolls out a random policy. We propose a conformant approximation that learns a fixed sequence of aggregate actions to be used for rollout from the current state. The choice of these aggregate actions is optimized simultaneously with the search for the next action to execute, using the same computation graph and gradient computation. This expands the scope of SOGBOFA to problems that require deep search and where information is lost quickly with random steps.

In addition to these contributions we generalize SOGBOFA

to handle a large set of state and action constraints in a manner that integrates well with the search through aggregate simulation. This enables the application of SOGBOFA to a larger benchmark set, including the problems from the recent International Planning Competition (IPC) 2018 that are used in the empirical evaluation. The results show that Lifted Conformant SOGBOFA has state-of-the-art performance on hard planning problems and achieves a higher IPC score than the planner with the highest score in the competition.

## Aggregate Simulation

Stochastic planning can be formalized using MDPs (Puterman 1994) in factored state and action spaces. In factored spaces (Boutilier, Dean, and Hanks 1995) states are complete variable assignments for a set of state variables $V$, and actions are complete variable assignments for the set of action variables $A$. In this paper we assume that state and action variables are binary. We write $S$ for the set of states, $\mathcal{A}$ for the set of actions and $s[v]$ or $a[v]$ for the value of variable $v$ in state $s$ or action $a$. We denote actions where exactly one action variable $a_i$ is true with $a_i$. Planning with a finite horizon $H$ from an initial state $s_I$ can be captured using a dynamic Bayesian network (DBN) where state and action variables are represented explicitly and the conditional probability tables (CPT) of state variables and the reward $R \in \mathbb{R}$ of applying action $a$ in state $s$ are repeated at each time step.

The planning model is given in a high-level description language from which one can extract the corresponding DBN. Here we focus on RDDL (Sanner 2010) as in the implementation of SOGBOFA but the ideas work for any such language. A planning problem is specified via a set of transition functions $T$ for each state variable $v \in V$ conditioned on actions, a set of constraints $P$, and the reward $R$. We note that some models integrate action preconditions into the transitions functions, but IPC 2018 introduced the use of action preconditions as a separate component of the constraints. We give an example with preconditions as constraints but the approach handles both formulations in the same manner.

The transition function $t_v \in T$ for $v \in V$ is translated into a set of mutually exclusive and exhaustive conditional effects $t_v = \{c_1 \rhd p_1, \ldots, c_n \rhd p_n\}$ where $p_i$ is $\mathbb{P}(v{=}\top)$ and the $c_i$ are conjunctions over the variables in $V$ and $A$ and possibly next state variables $V'$.[1] Constraints $P$ include both action constraints and action preconditions. Each constraint $p \in P$ is a propositional formula over $V$ and $A$. An action $a \in \mathcal{A}$ is applicable in state $s \in S$ iff $s, a \models p$ for each $p \in P$, and the application of an applicable action $a$ in $s$ leads to the successor state $s'$ where $s'[v]$ is true with probability $t_v(s, a)$, where $t_v(s, a)$ is the unique $p_i$ for which $s, a \models c_i$. Correlation among next state variables is achieved by allowing $c_i$ to depend on $V'$, where the dependence among $V'$ variables is acyclic. Finally, the reward $R$ is provided as an arithmetic expression over $V$ and $A$.

[1] To avoid computational burden, the implementation of SOGBOFA skips simplification of $t_v$ into a disjoint set of *conjunctions* and allows for more complex $c_i$. However, its approximate correctness properties discussed below require this preprocessing.

**Example 1.** *As running example, we use the MDP with* $V = \{v_1, v_2, v_3\}$, $A = \{a_1, a_2, a_3\}$, $T = \{t_{v_1}, t_{v_2}, t_{v_3}\}$, $P = \{p_1, p_2, p_3\}$ *and* $s_I = \{v_1 \to 0, v_2 \to 1, v_3 \to 0\}$ *with*

$$
\begin{aligned}
p_1 &:= a_1 \implies (v_2 \vee v_3),\\
p_2 &:= a_2 \implies (v_2 \vee v_3),\\
p_3 &:= a_1 + a_2 + a_3 \leq 1,\\
t_{v_1} &:= \{a_1 \rhd 1, \neg a_1 \rhd s[v_1]\},\\
t_{v_2} &:= \{a_2 \rhd 0.1, \neg a_2 \rhd 0\},\\
t_{v_3} &:= \{a_3 \rhd 0.5, \neg a_3 \rhd 0\},\\
R &:= v_1 - (6 \cdot a_1 \cdot v_1).
\end{aligned}
$$

In the example $p_1, p_2$ are action preconditions but $p_3$ is an action constraint. The objective is to compute a policy $\pi$ that maximizes the expected reward $V^\pi(s_I)$ over $H$ steps. In our example, an optimal policy executes noop in all states $s$ where $v_1$ holds, executes $a_1$ if $s \models (v_2 \vee v_3) \wedge \neg v_1$ and $a_3$ otherwise. Solving factored MDPs is computationally hard and off-line computation of a policy is often intractable in practice. On-line planning is an alternative approach where planning for the current state with a fixed time limit $t$ is interleaved with the execution of the taken decision.

The rollout algorithm of Tesauro and Galperin (1996) performs lookahead from the current state $s$ under a given baseline policy $\pi$. For each action that is applicable in $s$, it generates a set of trajectories that start with $a$ and follow $\pi$ afterwards. The cumulative reward of each trajectory constitutes a random sample of the action-value function $Q^\pi(s, a)$. The algorithm averages the cumulative rewards for each applicable action and executes the action maximizing this value.

Although this idea is simple and effective in many cases, it has two computational difficulties. First, generating enough trajectories that provide an accurate estimate of the action-values can be expensive. Second, enumerating all actions applicable in the current state is infeasible in large action spaces. AROLLOUT (Cui et al. 2015), short for *Algebraic Rollout*, uses aggregate simulation to address the first issue.

AROLLOUT transforms the transition functions $t_v = \{c_1 \rhd p_1, \ldots, c_n \rhd p_n\}$ into disjoint sum form $\sum_{i=1}^{n}(c_i \cdot p_i)$ and the Boolean expressions $c_i$ into algebraic expressions using standard transformations from a logical to a numerical representation. Instead of simulating individual trajectories, AROLLOUT approximates the state distributions in a simulation by computing marginal probabilities over individual variables. In each forward step, it calculates an approximation $\hat{p}(v)$ of the true marginal $p(v)$ for each $v \in V$. In particular, $\hat{p}(v)$ of state variable $v$ that depends on the state and action variables $x_1, \ldots, x_k$ (i.e., $x_1, \ldots, x_k$ occur in $t_v$) is calculated as a function of $\hat{p}(x_1), \ldots, \hat{p}(x_k)$ under the assumption that the $x_i$ are independent. For example, if the disjoint sum expression for $x_3$ is $(x_1 \wedge \neg x_2) \cdot 0.5 + \neg x_1 \cdot 0.6 + (x_1 \wedge x_2) \cdot 0.7$ we calculate $\hat{p}(x_3) = \hat{p}(x_1) \cdot (1 - \hat{p}(x_2)) \cdot 0.5 + (1 - \hat{p}(x_1)) \cdot 0.6 + \hat{p}(x_1) \cdot \hat{p}(x_2) \cdot 0.7$.

The simulation in SOGBOFA (Cui and Khardon 2016) takes this idea one step further. SOGBOFA exploits the fact that it is possible to use the expressions to construct an explicit directed acyclic graph (DAG) that represents exactly the same computation steps. We call this the SOGBOFA graph. The SOGBOFA graph represents the action for

the current time step as a symbolic input. Given an instantiation for the action, evaluating the computation graph performs the same calculation as AROLLOUT. If the model is specified with separate action preconditions, each occurrence of action variable $a_i$ is replaced with $a_i \wedge p_i$ where $p_i$ is the conjunction of conditions required by $a_i$. Action constraints (other than preconditions) are ignored in the construction of the graph and are handled separately. Finally, to support domains whose conditions include numerical equality tests as in $(\text{Exp} == k)$ with differential functions we replace such expressions with $\sigma(\text{Exp} - k + 0.5) - \sigma(\text{Exp} - k - 0.5)$ where $\sigma(a) = 1/(1 + e^{-a})$ is the well known sigmoid function.

Figure 1 (left) shows the basic form of the SOGBOFA graph with depth 3 for our running example. For visual clarity Boolean expressions have not been converted to their algebraic counterparts. The figure illustrates how the SOGBOFA graph is built from the transition functions for $v_1$, $v_2$ and $v_3$ and the reward formula (which are marked with different colors in the first layer of the graph). For instance, the value of $v_1$ in the second layer is represented by an addition node with two parents: a conjunction that encodes $(a_1 \wedge p_1)$ and a multiplication that encodes $\neg(a_1 \wedge p_1) \cdot v_1$. This translates to the disjoint sum form of $t_{v_1}$. Additional time steps are added to the SOGBOFA graph under the assumption that the random policy is rolled out, i.e., SOGBOFA uses a symbolic representation only for the first action and considers constant values for all further action variables. In our example, precondition $p_3$ can only hold if action variables are mutually exclusive, which gives marginals of $a_1 = a_2 = a_3 = \frac{1}{3}$ for rollout actions. All further layers are copies of layer two, with the exception of the last, where only nodes relevant for the action-value estimate are created.

The graph as depicted in Figure 1 (left) is meant to illustrate the compilation of the high-level MDP description into a graph. But this graph is not constructed by the system. SOGBOFA actually builds a graph for a specific state $s$ (whose state variables are binary valued) and simplifies any immediate computations such as $0 * x = 0$, $1 * x = x$, $0 + x = x$ or operations on two constants. Figure 1 (middle) shows the graph that is actually built by SOGBOFA for the initial state $s_I$ of our running example. Using concrete values for the state variables in the leaves simplifies the graph, but most of the size reduction appears close to the leaves. In our running example, almost all nodes from the first layer are simplified, but with the exception of the blue node at the top right which simplifies the multiplication of two constants, the two graphs are identical beyond the first layer.

## The Basic SOGBOFA Algorithm

The main idea in SOGBOFA is to use the computation graph to search over the action for the first step using gradient based search. This avoids enumeration of actions and tackles the second computational difficulty in the rollout algorithm. The SOGBOFA graph that is built for this purpose represents $Q^\pi(s, a)$ as a function of the action variables $a_1, \ldots, a_n$ that encode $a$. Once the graph is created, SOGBOFA initializes random values for $a_1, \ldots, a_n$. Then, at a high level, SOGBOFA can be seen to iteratively repeat the following steps:

1. propagate values from leaves to root to compute $Q^\pi(s, a)$
2. compute gradients for all nodes
3. take a gradient ascent step for $a_1, \ldots, a_n$

and when deliberation time runs out, it picks the action with the best $Q$ value. We refer the reader to (Cui and Khardon 2016) for complete details on the algorithm. In the following, we repeat details necessary to provide sufficient context for the paper.

**Gradient based search:** SOGBOFA calculates gradients using the method of automatic differentiation (Griewank and Walther 2008) and uses the gradients to update the values for action variables. In the SOGBOFA graph of our running example in Figure 1 (middle), the variables that are optimized are depicted in red. We use the reverse accumulation method that has linear complexity in the size of the DAG to calculate gradients for all nodes in the graph. The SOGBOFA implementation optimizes step size, and combines a stopping criterion for gradient search with random restarts to improve this process.

**Maintaining action constraints:** Gradient updates allow the values of marginal probabilities to violate the legal $[0, 1]$ range as well as violate explicit constraints on legal actions. SOGBOFA supports sum constraints of the form $\sum a_i \leq B$, as illustrated by $p_3$ in our running example. This is typical, for example, in high level representations that use 1-of-k representation for actions. For this we use the projected gradient ascent algorithm (Shalev-Shwartz 2012), where parameters are projected into the legal region after each update.

**Transforming aggregate actions to concrete actions:** Once the search is complete we have a values for $a_1, \ldots, a_n$. However, these values are numerical marginal probabilities for each action variable, whereas only a concrete action can be executed. SOGBOFA selects a concrete action from the marginal probabilities using a greedy iterative algorithm. In particular, we first sort action variables by their marginal probabilities. We then add active action bits as long as their marginal probability is not lower than marginal probability of random rollout and the constraints are not violated.

**Action Evaluation Step:** In addition to the above, we use action selection to improve quality estimates. In particular the $Q^\pi(s, a)$ values computed above are associated with the aggregate action $a$, but the aggregate action induces a concrete action $concrete(a)$ selected by the greedy algorithm. Therefore, we associate each aggregate action that is visited in the gradient search with $concrete(a)$ and record the more reliable value of $Q^\pi(s, concrete(a))$. These scores do not affect the gradient search, but final action selection is based on the values of the induced concrete actions.

**Dynamic simulation depth:** In principle, the creation of the SOGBOFA graph should be such that the number of graph layers matches the number of remaining steps to the horizon. However, if the horizon is large and/or each layer contains a large number of nodes, the evaluation of $Q^\pi(s, a)$ and the gradient computation are expensive so that the number of actions explored in the search might be too small. SOGBOFA therefore estimates the required time to calculate gradients and perform updates for one node. With this, the number

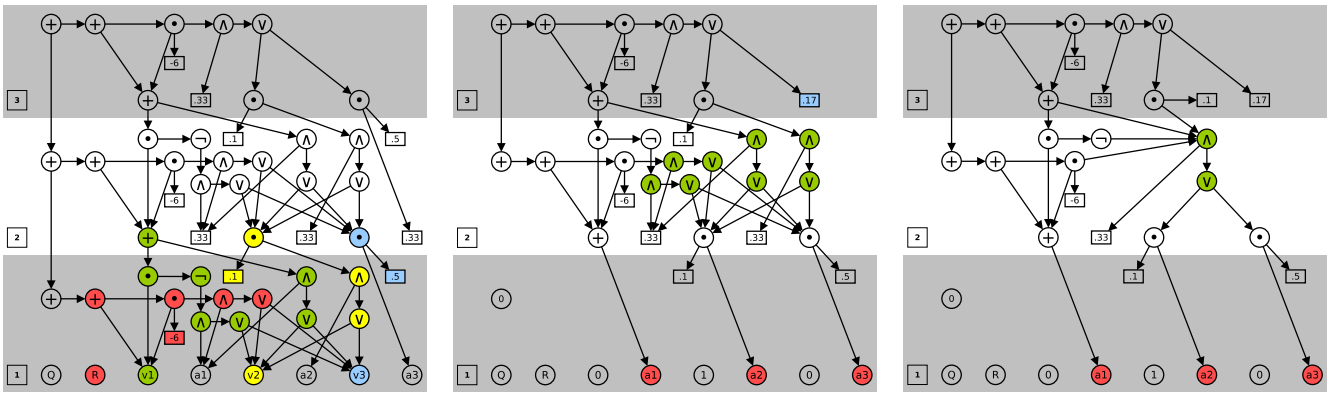

Figure 1: Three layers of the SOGBOFA graph of our running example. Left: basic construction. Middle: simplified graph for input state $\{v1 \to 0, v2 \to 1, v3 \to 0\}$ as built by SOGBOFA . Right: lifted graph. Colored nodes are referred to in the text.

of layers in the DAG is selected such that at least 500 updates on action marginals are possible with the time limit $t$. This fact is important to understanding the experimental evaluation in this paper because a compression of the SOGBOFA graph can lead to a deeper search or to an increase in search steps and both can affect planning performance.

## Lifted SOGBOFA

Although originally developed as a heuristic, recent work showed that the approximate value computed by the SOGBOFA graph is closely related to belief propagation. To explain this, we consider a finite-horizon MDP, and the corresponding DBN with evidence specifying current state and action, uniform priors for future actions and no further evidence. The connection is given by:

**Proposition 1. (Cui, Marinescu, and Khardon 2018)** *The marginals for state and reward variables computed by* AROLLOUT *and embedded in the* SOGBOFA *graph are exactly the same as the messages produced in a forward pass of belief propagation on the corresponding DBN. Moreover, because the DBN has no downstream evidence, BP has no backward messages and it converges in one pass. Therefore, the result of* AROLLOUT *is identical to the result of BP.*

Cui, Marinescu, and Khardon (2018) used this observation to show how planning algorithms can be used for solving marginal MAP inference problems. In this paper, we exploit the relationship to show how improved versions of BP can improve planning performance. In particular, we consider Lifted BP which aims to calculate the same solution as BP but do so more quickly by avoiding identical messages. The idea was first introduced by Singla and Domingos (2008) who showed how to extract the equivalence classes of messages from a structured model, specifically a Markov logic network. Each type of message is then computed only once and propagated as a power of the original individual message, replacing the product of identical messages. Kersting, Ahmadi, and Natarajan (2009) showed how repeated structure can be extracted from any ground Markov network by simulating only the process of BP without calculating the messages. The messages are then grouped and propagated similarly. Although this was developed for undirected models, the same ideas apply to directed models.

The crucial aspect in our context is the fact that *there are no backward messages*. Thanks to this we can focus on identical messages in a forward pass of BP for directed models, or equivalently in the computation path of AROLLOUT. Our main observation is that the pre-computation of Kersting, Ahmadi, and Natarajan (2009) can be considerably simplified in this case. In lifted BP, messages are identical if and only if their parents are identical and the local CPT structure is identical. For the computation graph of SOGBOFA this is easily tested. The computation of marginals for two nodes in the graph is identical if and only if their parents are identical and the local expression is identical and *this can be tested dynamically at construction time at the syntax level*. This test is just the standard notion of dynamic programming. The discussion gives the following observation and algorithm.

**Observation 2.** *One can detect identical BP messages for the computation of* SOGBOFA *by implementing the graph construction using dynamic programming.*

**Lifted SOGBOFA algorithm:**
We run the algorithm in exactly the same manner as SOGBOFA except for the construction of the SOGBOFA graph, which is modified as follows:

1. If an identical node with same parents and same operation has been constructed before return that node.
2. Replace a node $y = \underbrace{x \circ x \circ \ldots \circ x}_{k \text{ times}}$ with $y = x^k$ for multiplication and $y = k \cdot x$ for addition.
3. Otherwise, construct the node as in SOGBOFA.

Step (2) in the modified algorithm generates the counting expressions that are at the core of lifted inference algorithms. We emphasize that the lifting construction reorders and groups the computations but computes exactly the same function. Similarly, the gradients on the lifted graph w.r.t. action variables are identical.

For our running example, the differences between SOGBOFA and lifted SOGBOFA graphs are highlighted with the green nodes in the middle and right columns of Figure 1 which encode the same part of the MDP. We observe a significant reduction even in this tiny example. Saving is due to repeated sub-expressions and due to the same constant for different action variables feeding into similar expressions. We emphasize that constant nodes with the same value are not duplicated in the implementation. They are separated in the figure for visual clarity.

We note three differences between this algorithm and lifted BP. First, similar to the work of Kersting, Ahmadi, and Natarajan (2009) and unlike the work of Singla and Domingos (2008), our algorithm must first process the ground network, i.e., its complexity is linear in the size of the original ground computation graph. It is interesting to explore constructions that go directly from the relational planning problem description to the lifted SOGBOFA graph, but we do not pursue this here. One might worry that spending linear time on the construction is too costly if we have only one pass of messages on the graph. However, because we evaluate and calculate gradients using the same lifted SOGBOFA graph *many times* during optimization, the effect of a compressed computation graph can be significant. The second difference is due to the inputs to the two algorithms. SOGBOFA uses algebraic expressions whereas lifted BP uses tabular CPTs. SOGBOFA's computation is potentially more structured than lifted BP because lifting can occur at sub-CPT level. The third difference is that because SOGBOFA treats the action variables as symbols for the purpose of optimization, they are treated as having a different CPTs from the perspective of lifting. Summarizing this discussion we have:

**Proposition 3.** *The* SOGBOFA *graph of Lifted* SOGBOFA *is at least as compressed as the message structure in Lifted BP when run on the DBN generated by the planning problem, conditioned on the initial state and rollout policy, with no evidence, and where action variables have distinct CPTs.*

## Conformant SOGBOFA

The basic rollout algorithm evaluates an action by first applying it and then estimating the value of the next state by a rollout of a fixed policy $\pi$ (and averaging this process over multiple runs). The success of this approach relies on getting informative values from the rollout simulation using $\pi$. Such informative values need to distinguish high quality states from low quality states, or at least correctly rank such pairs of states. This obviously holds if the rollout policy is an optimal policy. However, failing that, the hope is that any reasonable policy will give some useful signal to distinguish the quality of states. SOGBOFA uses a random policy for rollout actions and in many domains even this seems to provide enough information to guide action selection.

However, it was already pointed out for ARollout (Cui et al. 2015) that in some problems rolling out the random policy masks crucial differences between values of states. They illustrate this point by the ELEVATORS domain from IPC 2011 where a reward is given for delivering passengers to their destination floor, but a large penalty is given if a passenger is transported in the wrong direction. In this case, rolling out the random policy (whether aggregate or not) incurs a large penalty when a passenger is in the elevator and states with no passengers in elevators appear better than ones with passengers. As a result SOGBOFA never chooses to allow passengers into elevators. The same phenomenon can be illustrated with our running example. Here, an optimal policy aims to make $v_1$ true by executing $a_1$, and then never executes $a_1$ again. The policy collects a reward of 1 in all future states. However, if the random policy is rolled out, $a_1$ is simulated with probability $\frac{1}{3}$, and the reward on a layer where $v_1$ is true is $1-(6\cdot\frac{1}{3}\cdot 1)=-1$ and hence worse than the reward of 0 for a state where $v_1$ is not true.

To avoid such pitfalls, the natural approach would be to use an informed rollout policy. Since we deal with domain-independent planning and do not have such a policy, we aim to learn the rollout policy during the planning process. However, this is difficult for several reasons and our prior work in this direction has not shown empirical success (Cui and Khardon 2016). In the following, we explain why using a rollout policy is not useful for aggregate states, and motivate the conformant solution.

We first note that unless the reward function (or goal) is put explicitly into the description of the state, an optimal rollout policy depends on the reward and therefore it is not transferable across different planning instances.[2] This means that the learned rollout policy will be used for a single problem instance. On the other hand, the setting of online planning typically does not allow for sufficient training trials to learn a full policy. Therefore, this approach is not useful unless one solves the same instance multiple times possibly with different start states. The latter is the typical setting in reinforcement learning but not in planning.

More importantly, as we argue next, a rollout policy is not useful with aggregate states. In this case, even if the rollout policy is a map from states to actions, it cannot see the concrete state in the simulation. Therefore, the rollout policy maps an aggregate state to an action. Moreover, the aggregate state which is observed by the policy is a deterministic function of three arguments: the current state, the rollout policy and the number of remaining steps. In other words *if the start state is fixed* then the policy can be seen to map the step number to an action, or simply to produce a sequential plan. We therefore have:

**Observation 4.** *When using simple aggregate simulation (i.e., if the rollout policy can only observe a product distribution over state variables) from a single start state, each optimal non-stationary rollout policy is a linear plan.*

This motivates the use of conformant solutions that use a linear plan for rollout actions. That is, the process of evaluating the first action also chooses a linear rollout plan that best supports that first action.[3] This can be supported with

[2]The work of Khardon (1999) shows how embedding the goal into the state allows for generalized policies.

[3]The rollout policy is not used to control the MDP, but only to improve the search, so this is technically different from conformant planning. But we borrow the name due to the use of the linear plan.

SOGBOFA with minor modifications. We build the SOGBOFA graph exactly as before (either lifted or non-lifted) except that trajectory action variables, and the corresponding nodes in the graph, are treated differently. Instead of assigning these nodes numerical values imposed by the random policy we retain them as symbolic variables and optimize them in the same manner as the first action. When using a sequential rollout plan, trajectory action nodes are leaves of the SOGBOFA graph and they can be treated in the same manner as actions for the first step. Note that reverse mode automatic differentiation supports calculation of gradients w.r.t. all nodes with the same time complexity so all that is needed is to identify the variables that are optimized and there are no significant changes in the algorithm or run time.

For our running example the SOGBOFA graph for the lifted conformant construction (not shown) is almost identical to the lifted graph. The only difference is that one of the nodes that was unified before is now separated out. Specifically, this happens because $a_1 \wedge p_1$ and $a_2 \wedge p_1$ which were identical when $a_1$ and $a_2$ had the value 0.33 are now distinct.

The use of conformant actions raises another subtle aspect. Recall that SOGBOFA searches over aggregate actions and uses gradients over this space but in the *action evaluation step* an aggregate action is scored by selecting a concrete binary action and computing its value. This was argued to add robustness to the search, because the actions that are actually used are binary. This argument does not hold here – rollout actions are not used for control, but only for evaluating the quality of the first action. In addition, an aggregate action captures a distribution over actions which is better suited for a distribution over states represented by the aggregate state. We therefore keep rollout actions in their aggregate form in the evaluation step. In summary, this yields the following algorithm:

**Conformant** SOGBOFA **algorithm:** We run the algorithm in exactly the same manner as (Lifted) SOGBOFA except for the following 3 changes:

1. The SOGBOFA graph is constructed as before except that all action variables are symbolic variables.
2. The gradient step is performed over all action variables.
3. In the action evaluation step, the aggregate action in the lowest layer is converted to a concrete action, but all other actions remain aggregated.

## Handling Constraints

The SOGBOFA algorithm supports basic sum constraints $\sum a_i \leq B$ that enable simple specification of mutually exclusive actions. The basic scheme, projected gradient ascent, is to first take a gradient step and then fix the outcome to satisfy the action constraints. We next introduce techniques for handling a much larger set of constraints in a manner which agrees with aggregate simulation and the action selection mechanism described above. This increases the applicability of the algorithm and specifically enables the experiments on domains from the International Planning Competition (IPC) 2018. These constraints are read from a high level specification in RDDL and parsed into the cases below.

The first type of constraints is an artifact of a change in the specification language of the IPC. Earlier RDDL models integrated action preconditions into the transition function. In this case, if an action is applied when its preconditions are violated it is effectively a no-op. IPC 2018 moved preconditions to a separate section with constraints of the form [action $\Rightarrow$ condition] and as a result when an action is applied when its preconditions are violated the simulator crashes and the algorithm fails. To support this we treat preconditions differently during action optimization and execution. Consider a ground precondition $[a_i \Rightarrow C]$ where $a_i$ is an action variable and $C$ is a formula. For optimization we modify the transition model by replacing every occurrence of $a_i$ with $(a_i \wedge C)$. This effectively treats actions with violated preconditions as no-op. This is applied at all levels of the SOGBOFA graph. During execution, action bits that violate preconditions for first action are fixed at 0 during search and therefore not selected.

While duplicating the precondition in the transition is expensive this is necessary for correctness and it is not clear how to automatically integrate preconditions into an algebraic form of the model otherwise. However, note that this is handled naturally by the lifted algorithm because nodes for the precondition are generated only once in the computation graph. The compression of lifting is not restricted to the use of explicit preconditions. This can be seen, for example, in the experimental results for the ELEVATORS domain whose specification uses the standard encoding.

The second type includes sum constraints $\sum a_i \leq B$ which are supported by projection as described above.

The third type are *action forcing* constraints. These constraints include an implication, [condition $\Rightarrow$ RHS], whose right hand side is a single action bit $a_j$ or a disjunction of such bits $\vee a_i$, or similarly constraints of the form $\sum a_i \geq 1$ and $\sum a_i = 1$. All these require the selection of (at least one) action bit to be true. These constraints are handled as follows. We first evaluate the condition to a value $v$. In a concrete state this evaluates to 0 or 1 and in an aggregate state it evaluates to (an approximation of) the probability that the condition holds. Now, for the case with a single forced action bit, we replace the marginal probability $p$ for for the corresponding $a_j$ by $p \leftarrow \max\{p, v\}$. Note that if $v = 0$ then $p$ does not change and if $v = 1$ then $p = 1$ which means that action selection mechanism will pick this action variable first, so we comply with the forced action constraint. In an aggregate state the effect is to increase $p$ to be as high as the probability $v$ that the condition holds. This implementation supports the conformant algorithm in the same manner as the action of the first step so no distinction is needed.

For the case of an implied disjunction of action variables we first evaluate the condition to a value $v$. We then pick the $a_i$ with the highest marginal probability $p$ among the ones in the disjunction and replace it with $\max\{p, v\}$. As with forced actions, if the condition is true then we force at least one of the relevant action variables to be true as well. With aggregate states we get a correction to the marginal probabilities on action variables.

The fourth type includes constraints of the form [$\Rightarrow$ RHS], that is, type 3 constraints with an empty condition. If we use the implementation of the previous paragraph in aggregate states this will force at least one action variable to be 1 and if the constraint is $\sum a_i = 1$ the trajectory actions in conformant SOGBOFA will always use discrete 0,1 values. This will hinder the search that uses aggregate actions which are the main advantage of our method. Instead, for this type of constraint, we first calculate $\sum p_i$ where $p_i$ is the current marginal probability of $a_i$. If $\sum p_i > 1$ we use projection as explained above. If $\sum p_i < 1$ we add $1 - \sum p_i$ to the largest among the $p_i$. In this way the constraint is satisfied on the fractional values but we do not force any specific action variable among trajectory actions to 1 during the search.

## Related Work

The paper makes use of the connection between planning and inference to improve the SOGBOFA algorithm, specifically introducing lifting and conformant solutions. The connection between planning and inference is not new and is rooted in work on influence diagrams (e.g., Cooper 1988; Mauá 2016). In probabilistic planning, Domshlak and Hoffmann (2006) showed how the conformant planning problem can be solved using weighted model counting for CNF, and Lee, Marinescau, and Dechter (2016) showed how the problem can be solved using MMAP inference. Our conformant solution is different because it optimizes over an approximate aggregate simulation and not the exact inference graph.

A different approach uses variational inference to solve MDPs. Starting with the work of Dayan and Hinton (1997), several formulations define a reward weighted posterior distribution over trajectories, where identifying the MAP over actions gives an optimal policy. Solutions include the expectation maximization (EM) or variational EM algorithms (Toussaint and Storsky 2006; Furmston and Barber 2010; Neumann 2011; Kober and Peters 2011), policy gradients (Kober and Peters 2011; van de Meent et al. 2016) and BP (Liu and Ihler 2012; Cheng et al. 2013). One of the main differences between these approaches and ours is that they condition on the reward whereas our approach evaluates the marginal of the reward given an action and can hence focus on forward inference. While the representations are equivalent when performing exact inference, the computational properties of different approximation algorithms on different Bayesian networks of the same distribution can vary dramatically. The choice of representation is hence important. We are not aware of approximate inference methods applied to planning problems of the same scale as done here.

Another major direction developed symbolic versions of dynamic programming algorithms (Boutilier, Dearden, and Goldszmidt 1995; Hoey et al. 1999; Raghavan et al. 2012; 2013). It is easy to see that symbolic value iteration corresponds to alternating variable elimination where we alternate between eliminating state variables of a time step and action variables of the same step. The main difficulty with this approach is the space requirements of representing value functions or policies. The SOGBOFA solutions avoid such difficulties by not representing a full policy and by integrating the approximation into the computation graph.

## Experimental Evaluation

In this section we evaluate the performance of the proposed algorithms. For uniformity and to compare to state of the art we use the problems and experimental setup from the IPC 2018[4]. The competition included 8 domains with 20 problems each. We additionally show results for the ELEVATORS domain from IPC 2011 which was discussed above. For this domain we use the 10 instances from the competition and add 10 larger instances with up to 10 elevators and 43 floors. We compare our algorithms to PROST 2011 (Keller and Eyerich 2012) and PROST 2014 (Keller and Helmert 2013), the baselines of IPC 2018 that also achieved the highest IPC scores in the competition.

We run each algorithm 75 times on each problem and collect average rewards for each problem. As in the competition, each algorithm is given a total time for each instance which is equal to 2.5 seconds times the problem horizon times 75 and it must select actions for all these steps within that time. To compare algorithms we use the IPC score which normalizes the average cumulative reward of each planner based on the score of a minimal policy (the better of random and no-op) and best score obtained by all planners on the same instance. This score is not ideal when (near-)optimal values are unknown which leads to some variance in normalized scores. But it has become the standard way to compare systems because it provides a single normalized score with the same scale across problems and instances.

Recall that SOGBOFA uses a dynamic scheme to balance graph size and search depth. Therefore, the effect of lifting can be a smaller graph and larger number of search steps or a deeper search with similar number of search steps. This complicates the evaluation of lifting. Therefore, we first explore size compression *for fixed depth*, independently of planning performance. Figure 2 shows scatter plots with pairwise comparisons of graph size (at fixed depth) between different algorithms. As can be seen, lifting leads to compression by a factor of at least 2 on 6 of the 9 domains, with up to 8x compression in PUSHYOURLUCK, CHROMATICDICE and COOPERATIVERECON. Conformant variants increase the size because they prevent reusing constant value nodes for action variables of rollout actions.

We next turn to evaluate planning performance with dynamic depth. We first show the planning results for the ELEVATORS domain. Figure 3 shows search depth, and the number of search steps (restarts and gradient steps together) and normalized planning performance scores. We see that lifting leads to a deeper search and that (due to this) all 3 variants decrease the number of updates in this domain. Considering planning performance, lifting on its own does not improve, whereas conformant variants do and their performance is close. Therefore, in this problem the increased depth does not lead to improved performance and the key to success is the use of the conformant rollout policy.

We next turn to planning results on IPC 2018 domains. Table 1 shows the overall IPC scores for each planner on each domain, as well as the sum over all domains. Considering first the comparison among SOGBOFA variants. We see that

[4] https://ipc2018-probabilistic.bitbucket.io

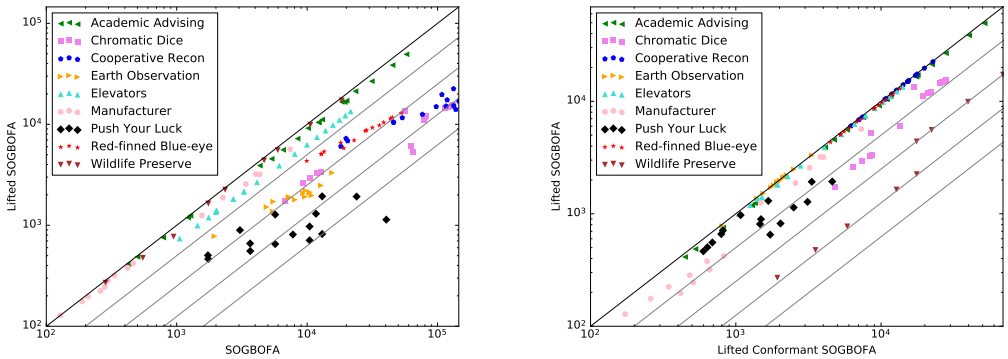


Figure 2: Pairwise comparisons of graph sizes between algorithms. Diagonal lines correspond to doubling the size.

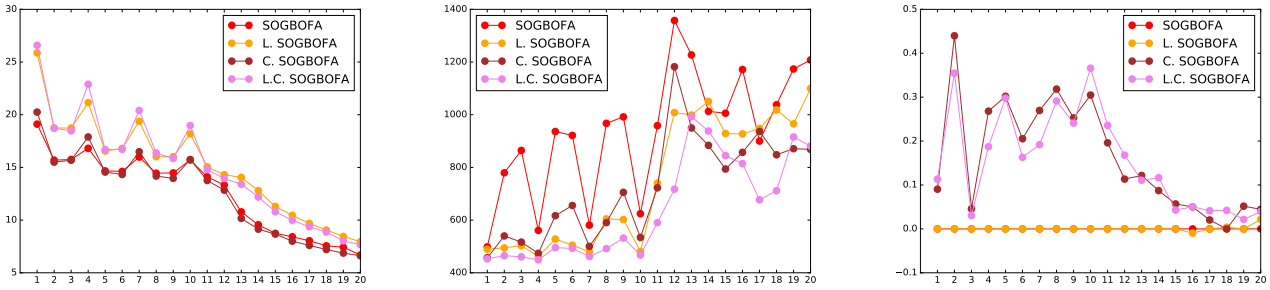


Figure 3: Results for the ELEVATORS domain: columns represent depth, search steps, and normalized average cumulative reward. Rewards are normalized as (avg_reward(algorithm) − avg_reward(SOGBOFA))/ |avg_reward(SOGBOFA)|.

lifted and conformant variants improve performance in 5 of the 8 domains and (except for conformant variants on COOPERATIVERECON) they do not harm performance in other domains. We also see that SOGBOFA variants perform significantly better than PROST variants on 4 of the 8 domains. Considering the total score over all domains, all SOGBOFA variants perform equal to or better than the PROST variants and Lifted Conformant SOGBOFA is ahead by a wide margin.

We next consider performance in individual domains, starting with a comparison to PROST. For ACADEMIC and WILDLIFE the differences between SOGBOFA and PROST variants are relatively small. Many of the test instances in these domains are hard for all all planners in the experiments and this is reflected in the scores. PROST variants perform better on EARTHOBSERVATION and PUSHYOURLUCK. These are the two domains with the smallest number of actions and with limited stochasticity. This agrees with our expectation. PROST uses Monte-Carlo Tree Search with concrete states and actions which leads to high quality estimates when the action space is small and stochasticity is limited. On the other hand SOGBOFA's strength is in large action spaces and high stochasticity. SOGBOFA variants perform significantly better in the other 4 domains.

Finally note that both lifting and conformant extensions are important for the performance of the system as a whole. For the MANUFACTURER domain we see that variants not using conformant search have a score of zero. The reason is similar to the one for ELEVATORS. Here random rollouts do not show the advantage of makings goods or investing in factories (in order to sell products in future) and as a result SOGBOFA decides not to do anything. On the other hand, the conformant variants are successful. The results for REDFINNEDBLUEEYE show the advantage of lifting (as well as the potential disadvantage of conformant which makes the search more expensive). In EARTHOBSERVATION, we see that both lifting and conformant improve individually but that their combination is significantly better.

## Conclusions

The SOGBOFA system has been previously shown to have excellent performance in many domains. However, it works with a ground computation graph and does not use problem structure. In addition, the approach fails for problems where rolling out the random policy does not provide useful information on the quality of a state. The paper introduces new techniques to address these deficiencies. The first uses lifted BP to compress the SOGBOFA graph. The second introduces conformant optimization into the aggregate search of SOGBOFA. The experiments demonstrate that lifting improves search through size compression and deeper search and that in some domains conformant search leads to better decisions. Overall, Lifted Conformant SOGBOFA provides

| | CoopRecon | Academic | Earth | Push | Wildlife | Red | Manufacturer | Chromatic | Total |
|---|---|---|---|---|---|---|---|---|---|
| Prost 2014 | 6.82 | 3.7 | 19.87 | 15.8 | 8.71 | 6.5 | 2.14 | 10.17 | 73.71 |
| Prost 2011 | 5.3 | 3.77 | 17.77 | 7.61 | 3.99 | 5.06 | 5.1 | 12.9 | 61.5 |
| sogbofa | 14.64 | 4.83 | 4.01 | 5.17 | 9.25 | 17.43 | 0 | 18.26 | 73.59 |
| L. sogbofa | 16.21 | 4.95 | 6.69 | 5.46 | 8.42 | 18.64 | 0 | 18.52 | 78.89 |
| C. sogbofa | 12.96 | 5.36 | 7.1 | 4.37 | 8.61 | 16.54 | 10.24 | 17.94 | 83.12 |
| L.C. sogbofa | 12.53 | 5.1 | 11.46 | 4.78 | 9.04 | 17.09 | 10.04 | 19.48 | 89.52 |

Table 1: IPC scores of the 4 variants of SOGBOFA and the two IPC baselines on the IPC 2018 benchmarks.

state of the art performance.

There are many interesting directions for future work. The paper argued that due to the limitations of aggregate search the best rollout policy is a sequential plan. Maintaining the advantage of aggregate search but allowing for more complex rollout policies is an important challenge. Another interesting direction includes incorporating the SOGBOFA graph and its $Q$ estimates within traditional search based planning algorithms. More generally, as demonstrated in this paper, the synergy between planning and inference has the potential to improve algorithms in both fields.

**Acknowledgments** This work was partly supported by NSF under grant IIS-1616280. Thomas Keller received funding for this work from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement no. 817639).

## References

Boutilier, C.; Dean, T.; and Hanks, S. 1995. Planning under uncertainty: Structural assumptions and computational leverage. In *Proc. of the Second European Workshop on Planning*, 157–171.

Boutilier, C.; Dearden, R.; and Goldszmidt, M. 1995. Exploiting structure in policy construction. In *Proc. IJCAI*, volume 14, 1104–1113.

Cheng, Q.; Liu, Q.; Chen, F.; and Ihler, A. T. 2013. Variational planning for graph-based MDPs. In *Proc. NIPS*, 2976–2984.

Cooper, G. F. 1988. A method for using belief networks as influence diagrams. In *Proc. UAI*, 55–63.

Cui, H., and Khardon, R. 2016. Online symbolic gradient-based optimization for factored action MDPs. In *Proc. IJCAI*, 3075–3081.

Cui, H.; Khardon, R.; Fern, A.; and Tadepalli, P. 2015. Factored MCTS for large scale stochastic planning. In *Proc. AAAI*, 3261–3267.

Cui, H.; Marinescu, R.; and Khardon, R. 2018. From stochastic planning to marginal MAP. In *Proc. NIPS*, 3085–3095.

Dayan, P., and Hinton, G. E. 1997. Using expectation-maximization for reinforcement learning. *Neural Computation* 9(2):271–278.

Domshlak, C., and Hoffmann, J. 2006. Fast probabilistic planning through weighted model counting. In *Proc. ICAPS*, 243–252.

Furmston, T., and Barber, D. 2010. Variational methods for reinforcement learning. In *Proc. AISTATS*, 241–248.

Griewank, A., and Walther, A. 2008. *Evaluating derivatives - principles and techniques of algorithmic differentiation (2. ed.)*. SIAM.

Hoey, J.; St-Aubin, R.; Hu, A.; and Boutilier, C. 1999. SPUDD: Stochastic planning using decision diagrams. In *Proc. UAI*, 279–288.

Keller, T., and Eyerich, P. 2012. PROST: probabilistic planning based on UCT. In *Proc. ICAPS*, 119–127.

Keller, T., and Helmert, M. 2013. Trial-based heuristic tree search for finite horizon MDPs. In *Proc. ICAPS*, 135–143.

Kersting, K.; Ahmadi, B.; and Natarajan, S. 2009. Counting belief propagation. In *Proc. UAI*, 277–284.

Khardon, R. 1999. Learning action strategies for planning domains. *Artificial Intelligence* 113(1-2):125–148.

Kober, J., and Peters, J. 2011. Policy search for motor primitives in robotics. *Machine Learning* 84(1-2):171–203.

Kolobov, A.; Dai, P.; Mausam, M.; and Weld, D. S. 2012. Reverse iterative deepening for finite-horizon MDPs with large branching factors. In *Proc. ICAPS*, 146–154.

Lee, J.; Marinescau, R.; and Dechter, R. 2016. Applying search based probabilistic inference algorithms to probabilistic conformant planning: Preliminary results. In *Proc. ISAIM*.

Liu, Q., and Ihler, A. T. 2012. Belief propagation for structured decision making. In *Proc. UAI*, 523–532.

Mauá, D. D. 2016. Equivalences between maximum a posteriori inference in bayesian networks and maximum expected utility computation in influence diagrams. *International Journal Approximate Reasoning* 68:211–229.

Neumann, G. 2011. Variational inference for policy search in changing situations. In *Proc. ICML*, 817–824.

Puterman, M. L. 1994. *Markov decision processes: Discrete stochastic dynamic programming*. Wiley.

Raghavan, A.; Joshi, S.; Fern, A.; Tadepalli, P.; and Khardon, R. 2012. Planning in factored action spaces with symbolic dynamic programming. In *Proc. AAAI*, 1802–1808.

Raghavan, A.; Khardon, R.; Fern, A.; and Tadepalli, P. 2013. Symbolic opportunistic policy iteration for factored-action MDPs. In *Proc. NIPS*, 2499–2507.

Sanner, S. 2010. Relational dynamic influence diagram language (RDDL): Language description. *Unpublished Manuscript. Australian National University*.

Shalev-Shwartz, S. 2012. Online learning and online convex optimization. *Foundations and Trends in Machine Learning* 4(2):107–194.

Singla, P., and Domingos, P. M. 2008. Lifted first-order belief propagation. In *Proc. AAAI*, 1094–1099.

Tesauro, G., and Galperin, G. 1996. On-line policy improvement using Monte-Carlo search. In *Proc. NIPS*, 1068–1074.

Toussaint, M., and Storsky, A. 2006. Probabilistic inference for solving discrete and continuous state Markov decision processes. In *Proc. ICML*, 945–952.

van de Meent, J.; Paige, B.; Tolpin, D.; and Wood, F. 2016. Black-box policy search with probabilistic programs. In *Proc. AISTATS*, 1195–1204.