The SOGBOFA system in IPC 2018: Lifted BP for Conformant Approximation of Stochastic Planning

Hao Cui and **Roni Khardon**
Department of Computer Science, Tufts University, Medford, Massachusetts, USA
Hao.Cui@tufts.edu, roni@cs.tufts.edu

Abstract

The SOGBOFA algorithm estimates the value of an action for the current state by building an explicit computation graph capturing an approximation of the value obtained when starting with this action and continuing with a random policy. This is combined with automatic differentiation over the graph to search for the best action. This approach was shown to be competitive in large scale planning problems with factored state and action spaces. Two extensions of the SOGBOFA system participated in the probabilistic track of the international Planning Competition (IPC). Both extensions use the recently observed connection between SOGBOFA and belief propagation to improve efficiency by lifting its computation graph, taking inspiration from lifted belief propagation. The second extension improves the rollout policy which is used in the approximate computation graph. Instead of rolling out a trajectory of the random policy, the trajectory actions are optimized at the same time as the initial action. In addition, due to changes in the specification language for the IPC, new facilities for handling action constraints were incorporated in the system.

Introduction

This paper gives an overview of variants of the SOGBOFA system that participated in the probabilistic track of the international Planning Competition (IPC), 2018. SOGBOFA (Cui and Khardon 2016) extends the well known rollout algorithm (Tesauro and Galperin 1996). The rollout algorithm uses a simulator to estimate the quality of each possible action for the first step by taking that action and then continuing the simulation with some fixed policy. Multiple simulations are required for each fixed action to get a reliable estimate. But once this is done one can perform policy improvement or just use the best action for the current state. SOGBOFA improves over this algorithm in two important ways. The first is that instead of using concrete simulation of trajectories the algorithm builds an explicit computation graph capturing an approximation of the corresponding value when rolling out the random policy. Therefore a single symbolic simulation suffices. The second is that because the simulation is given in an explicit computation graph one can use automatic differentiation and gradients to search for the best action, avoiding the action enumeration which is required by rollout. This approach was shown to be competitive in large scale planning problems with factored state and action spaces where such enumeration is not feasible.

Two improvements of the SOGBOFA system were added for the competition. In recent work we have shown that the computation graph of SOGBOFA calculates exactly the same solution as the one that would be computed by belief propagation (BP) on the corresponding inference problem (Cui and Khardon 2018). The first improvement uses a Lifted version of SOGBOFA, taking inspiration from lifted belief propagation. The idea in lifted BP is to avoid repeated messages during computation and calculate the result in aggregate. For SOGBOFA this turns out to be a simple modification of the construction of the computation graph.

The second improvement uses conformant approximation. The quality of the approximation of SOGBOFA is limited by the fact that it uses a random policy for rollout. For some domains this provides enough information to distinguish the best first action but for others this does not work. The conformant approximation learns a fixed sequence of actions to be used for rollout from the current state (this is similar to the plan used in conformant planning, hence the name for this approximation). The choice of these actions is optimized simultaneously with the optimization of the first action, using the same computation graph and gradient computation.

For the competition we used two entries. The first uses only Lifting, and the second uses Lifting and Conformant approximation where all action variables are optimized simultaneously at every step.

IPC 2018 has modified the RDDL (Sanner 2010) specification of domains by moving action preconditions into a separate constraints section and adding several other types of action constraints in that section that must be handled by the planner. The SOGBOFA entries for the IPC extend the system to handle these constructs.

The rest of the paper is structured as follows. The next section gives an overview of the original SOGBOFA algorithms. The following 3 sections describe lifting, the conformant approximation and constraints handling.

The Basic SOGBOFA Algorithm

Stochastic planning can be formalized using Markov decision processes (Puterman 1994) in factored state and action spaces. In factored spaces (Boutilier, Dean, and Hanks 1995) the state is specified by a set of variables and the number of states is exponential in the number of variables. Similarly in factored action spaces an action is specified by a set of variables. We assume that all state and action variables are binary. Finite horison planning can be captured using a dynamic Bayesian network (DBN) where state and action variables at each time step are represented explicitly and the CPTs of variables are given by the transition probabilities. In off-line planning the task is to compute a policy that optimizes the long term reward. In contrast, in on-line planning (which is the setup in the IPC) we are given a fixed limited time, t seconds, per step and cannot compute a policy in advance. Instead, given the current state, the algorithm must decide on the next action within t seconds. Then the action is performed, a transition and reward are observed and the algorithm is presented with the next state. This process repeats and the long term performance of the algorithm is evaluated.

AROLLOUT and SOGBOFA perform on-line planning by estimating the value of initial actions where a fixed rollout policy, typically a random policy, is used in future steps. The AROLLOUT algorithm (Cui et al. 2015) introduced the idea of algebraic simulation to estimate values but optimized over actions by enumeration. Then Cui and Khardon (2016) showed how algebraic rollouts can be computed symbolically and that the optimization can be done using automatic differentiation. The high level structure of SOGBOFA is:

SOGBOFA(S)

1 $Qf \leftarrow BuildQf(S, timeAllowed)$
2 $As = \{\ \}$
3 **while** time remaining
4 **do** $A \leftarrow RandomRestart()$
5 **while** time remaining and not converged
6 **do** $D \leftarrow CalculateGradient(Qf)$
7 $A \leftarrow MakeUpdates(D)$
8 $A \leftarrow Projection(A)$
9 $As.add(SampleConcreteAct(A))$
10 $action \leftarrow Best(As)$

Overview of the Algorihm: In line 1, we build an expression graph that represents the approximation of the Q function. This step also explicitly optimizes a tradeoff between simulation depth and run time to ensure that enough updates can be made. Line 4 samples an initial action for the gradient search. Lines 6 to 8 calculate the gradient and make an update on the aggregate action. Line 9 makes the search more robust by finding a concrete action induced by the current aggregate action and evaluating it explicitly. Line 10 picks the action with the maximum estimate. Line 5 checks our stopping criterion which allows us to balance gradient and random exploration. In the following we describe these steps in more details.

Building a symbolic representation of the Q function: Finite horizon planning can be translated from a high level language (e.g., RDDL (Sanner 2010)) to a dynamic Bayesian network (DBN). AROLLOUT transforms the CPT of a node x into a disjoint sum form. In particular, the CPT is represented in the form $if(c_{11}|c_{12}...)\ then\ p_1\ ...\ if(c_{n1}|c_{n2}...)\ then\ p_n$, where p_i is $p(x{=}1)$ and the c_{ij} are conjunctions of parent values which are are mutually exclusive and exhaustive. It then performs a forward pass calculating $\hat{p}(x)$, an approximation of the true marginal $p(x)$, for any node x in the graph. $\hat{p}(x)$ is calculated as a function of $\hat{p}(c_{ij})$, an estimate of the probability that c_{ij} is true, which assumes the parents are independent. This is done using the following equations where nodes are processed in the topological order of the graph:

$$\hat{p}(x) = \sum_{ij} p(x|c_{ij})\hat{p}(c_{ij}) = \sum_{ij} p_i \hat{p}(c_{ij}) \qquad (1)$$

$$\hat{p}(c_{ij}) = \prod_{w_k \in c_{ij}} \hat{p}(w_k) \prod_{\bar{w}_k \in c_{ij}} (1 - \hat{p}(w_k)). \qquad (2)$$

The following example from (Cui and Khardon 2016) illustrates AROLLOUT and SOGBOFA . The problem has three state variables s(1), s(2) and s(3), and three action variables a(1), a(2), a(3) respectively. In addition we have two intermediate variables $cond1$ and $cond2$ which are not part of the state. The transitions and reward are given by the following RDDL (Sanner 2010) expressions where primed variants of variables represent the value of the variable after performing the action.

```
cond1 = Bernoulli(0.7)
cond2 = Bernoulli(0.5)
s'(1) = if (cond1)  then ~a(3)  else false
s'(2) = if (s(1))   then a(2)   else false
s'(3) = if (cond2)  then s(2)   else false
reward = s(1) + s(2) + s(3)
```

AROLLOUT translates the RDDL code into algebraic expressions using standard transformations from a logical to a numerical representation. In our example this yields:

```
s'(1) = (1-a(3))*0.7
s'(2) = s(1)*a(2)
s'(3) = s(2) * 0.5
r = s(1) + s(2) + s(3)
```

These expressions are used to calculate an approximation of marginal distributions over state and reward variables. The distribution at each time step is approximated using a product distribution over the state variables. To illustrate, assume that the state is $s_0 = \{s(1){=}0, s(2){=}1, s(3){=}0\}$ which we take to be a product of marginals. At the first step AROLLOUT uses a concrete action, for example $a_0 = \{a(1){=}1, a(2){=}0, a(3){=}0\}$. This gives values for the reward $r_0 = 0 + 1 + 0 = 1$ and state variables $s_1 = \{s(1){=}(1-0)*0.7{=}0.7, s(2){=}0*0{=}0, s(3){=}1{*}0.5 = 0.5\}$. In future steps it calculates marginals for the action variables and uses them in a similar manner. For example if $a_1 = \{a(1){=}0.33, a(2){=}0.33, a(3){=}0.33\}$ we get $r_1 = 0.7 + 0 + 0.5 = 1.2$ and $s_2 = \{s(1){=}(1-0.33)*0.7$, s(2)= 0.7 * 0.33, $s(3){=}0*0.5\}$. Summing the rewards from all steps gives an estimate of the Q value for a_0. AROLLOUT

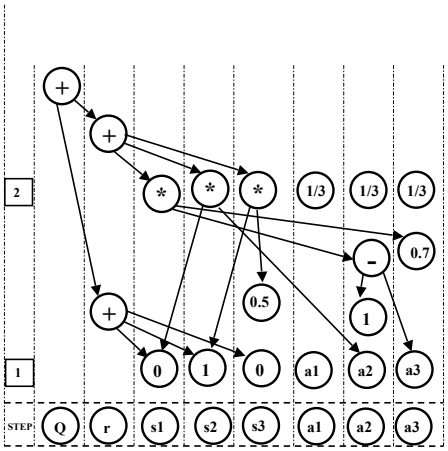

Figure 1: Example of SOGBOFA graph construction.

randomly enumerates values for a_0 and then selects the one with the highest estimate.

The main observation in SOGBOFA is that instead of calculating concrete values, we can use the expressions to construct an explicit directed acyclic graph representing the computation steps, where the last node represents the expectation of the cumulative reward. SOGBOFA uses a symbolic representation for the first action and assumes that the rollout uses the random policy. In our example if the action variables are mutually exclusive (such constraints are used imposed in high level domain descriptions) this gives marginals of $a_1 = \{a(1)$=0.33, $a(2)$=0.33, $a(3)$=0.33$\}$ over these variables. The SOGBOFA graph for our example expanded to depth 1 is shown In Figure 1. The bottom layer represents the current state and action variables. Each node at the next level represents the expression that AROLLOUT would have calculated for that marginal. To expand the planning horizon we simply duplicate the second layer construction multiple times.

Now, given concrete marginal for the action variables at the first step, i.e., a_0, an evaluation of the graph captures the same calculation as AROLLOUT.

Dynamic control over simulation depth: In principle we should build the DAG to the horizon depth. However, if the DAG is large then evaluation of $Q^\pi(s, a)$ and gradient computation are expensive so that the number of actions explored in the search might be too small. SOGBOFA first estimate the time cost for calculating gradients and performing updates *per node in the graph*. It then estimates the potential graph size as a function of simulation depth. The rollout depth is chosen to guarantee at least 200 updates on action marginals.

Random Restarts: A random restart generates a concrete (binary) legal action in the state.

Calculating the gradient: The main observation is that once the graph is built, representing Q as a function of action variables, we can calculate gradients using the method of automatic differentiation (Griewank and Walther 2008). Our implementation uses the reverse accumulation method since it is more efficient having linear complexity in the size of the DAG for calculation of all gradients.

Maintaining action constraints: Gradient updates allow the values of marginal probabilities to violate the $[0, 1]$ range constraint on marginals as well as explicit constraints on legal actions. The original version of SOGBOFA only allowed for sum constraints of the form $\sum a_i \leq B$. This is typical, for example, in high level representations that use 1-of-k representation for actions where at most one bit among a group of k should be set to 1 or in cases of limited concurrency.

To handle this issue we use projected gradient ascent (Shalev-Shwartz 2012), where parameters are projected into the legal region after each update. The projection step uses an iterative procedure from (Wang and Carreira-Perpiñán 2013; Duchi et al. 2008) that supports constraints of the form $\sum a_i \leq B$. The algorithm repeatedly subtracts the surplus amount ($(\sum a_i - B)/k$ for k action variables) from all nonzero entries, clipping at 0, until the surplus is 0.

Optimizing step size for gradient update: Gradient ascent often works well with a constant step size. However, in some problems, when the rollout policy is random, the effect of the first action on the Q value is very small implying that the gradient is very small and a fixed step size is not suitable. This also implies that we need to search over a large range for an appropriate step size. SOGBOFA performs this search hierarchically. We start with a large fixed range and search over a grid set of values. Then if the value chosen is the smallest one tested we recurse to a smaller range.

Sampling concrete Actions: The search assigns numerical marginal probabilities for each action variable. We need to select a concrete action from this numerical representation. In addition the selected action must satisfy the action constraints as mentioned above. SOGBOFA uses a greedy heuristic as follows. We first sort action variables by their marginal probabilities. We then add active action bits as long as the marginal probability is not lower than marginal probability of random rollout and the constraints are not violated. For example, suppose the marginals are $\{0.8, 0.6, 0.5, 0.1, 0\}$, $B = 3$, and we use a threshold of 0.55. Then we have $\{a_1, a_2\}$ as the final action. This procedure is used for selecting the final action to use during online planning.

In addition to the above, we also use action selection during the search. The gradient optimization performs a continuous search over the discrete space. This means that the values given by the graph are not always reliable on the fractional aggregate actions. To add robustness we associate each aggregate action encountered in the search with a concrete action chosen as in the previous paragraph and record the more reliable value of the concrete action. Search proceeds as usual with the aggregate actions but final action selection is done using these more reliable scores for concrete actions.

Stopping Criterion: Our results show that gradient information is useful. However, getting precise values for the marginals at local optima is not needed because small changes are not likely to affect the choice of concrete action.

We thus use a loose criterion aiming to allow for a few gradient steps but to quit relatively early so as to allow for multiple restarts. Our implementation stops the gradient search if the max change in probability is less than $S = 0.1$, that is, $\|A_{new} - A_{old}\|_1 \leq 0.1$. The result is an algorithm that combines Monte Carlo search and gradient based search.

Lifted SOGBOFA

In recent work (Cui and Khardon 2018) we showed that the approximate value computed by SOGBOFA is identical to the value that would be computed by belief propagation. Motivated by that we proposed a lifted planning algorithm that takes inspiration from lifted BP (Singla and Domingos 2008; Kersting, Ahmadi, and Natarajan 2009). In Lifted BP, two nodes or factors send identical messages when all their input messages are identical and in addition the local factor is identical. With the computational structure of SOGBOFA this has a clear analogy. Two nodes in SOGBOFA's graph are identical when their parents are identical and the local algebraic expressions capturing the local factors are identical. This suggests a straightforward implementation for Lifted SOGBOFA using dynamic programming.

The lifted algorithm is as follows. We run the algorithm in exactly the same manner as SOGBOFA except for the construction of the computation graph. (1) When constructing a node in the explicit computation graph we check if an identical node, with same parents and same operation, has been constructed before. If so we return that node. Otherwise we create a new node in the graph. (2) If we only use the idea above we may end up with multiple identical edges between the same two nodes. Such structures are simplified during construction. In particular, every `plus` node with k incoming identical edges is turned into a `multiply node` with constant multiplier k. Similarly, every `multiply` node with k incoming identical edges is turned into a `power node` with constant multiplier k. This automatically generates the counting expressions that are often seen in lifted inference.

Conformant SOGBOFA

SOGBOFA uses a random policy for trajectory actions, that is, for actions taken after the first step. In some domains the value achieved by the random policy is already indicative of the value of the state it is started from. In these cases SOGBOFA is successful. In some domains a random policy masks any information and one must use a more informed type of lookahead. For SOGBOFA there is a natural way to form this idea: one can try to improve the rollout policy. It turns out, however, that optimizing a policy within the time to select an action for online planning is not realistic. In addition, because we only maintain aggregate marginals for states after the first step, the simulation cannot condition the policy on a concrete state. We therefore optimize a choice of a concrete action (not conditioned on the state) for each time step. That is, the process of evaluating the first action also chooses a linear plan that best supports that first action. The linear plan is only used for the purpose of action evaluation. It is not used for controlling the MDP. Instead, as always done in online planning, the process restarts its computation after the first action has been taken.

The SOGBOFA graph immediately support this type of optimization because trajectory action variables are represented as nodes in the graph. 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. Note that reverse mode automatic differentiation supports calculation of gradients w.r.t. all nodes with the same time complexity so that there is no significant change to run time. However, in preliminary experiments we have found that optimizing a large number of action variables from all time steps requires significantly more gradient steps to reach a good solution.

Conformant SOGBOFA adjusts for this by modifying the dynamic depth selection heuristic. As before, we estimate the cost of gradient computation *per node* in the graph and the expected graph size as a function of depth. Our heuristic here is to select the depth so as to guarantee $200 * 2^i$ updates, where i is the conformant search depth.

Note that in principle we could separate search depth from conformant depth, for example, optimize actions for the first few steps and thereafter use the random policy. This can allow for a more refined control of the time tradeoff for the search. However, for the IPC we did not implement such a scheme. We use two variants, the first without the conformant improvement and the other always optimizing all levels of the search.

Handling Constraints

Previous versions of the probabilistic IPC integrated action preconditions into the transition function. With this, if an action is applied in a state where it is not legal it is simply a no-op and therefore not useful. Action constraints were limited to the sum constraints discussed above. The current IPC represents action preconditions as constraints, and if an illegal action is attempted the planner fails. In addition, the types of constraints were expanded to include more constraints on actions.

Our implementation parses the action constraints and handles each type in a separate manner, following naturally the previous treatment in SOGBOFA.

First, action preconditions have the form: "for all arguments x, if $a(x)$ is used, then $c(x)$ must be true", where $c(x)$ is the precondition. When we identify this form, we embed it into the transition function to mimic the previous IPC encoding. In particular, consider a concrete action bit $a(o)$ for some concrete objects o. We replace each occurrence of $a(o)$ in the transition function by $a(o) \wedge c(o)$. In other words, during simulation we assume that illegal actions are no-op.

Second, constraints of the form $\sum a_i \leq B$ are treated exactly as before.

The IPC used additional types of constraints. The first includes conditions of the form: "some quantifiers x, condition(x) $\rightarrow$ some quantifiers y, actions(x, y)" where actions(x, y) is a subset of the ground action variables, and the quantifiers over action arguments and other objects can be existential or universal as appropriate. When

actions(x, y) include just one action then we have a forced action. When actions(x, y) represents a disjunction of action variables, this means that at least one of these actions must be chosen. A similar situation arises with $\sum a_i \geq 1$ which captures a disjunction and with $\sum a_i = 1$ which has both the upper bound and a disjunction. It should be clear the SOGBOFA is not well matched for handling general action constraints. Our implementation uses the previous methodology to support the new constraints as follows.

For constraints with forced actions we can evaluate the condition to a value v. In a concrete state it evaluates to 0 or 1 and in an aggregate state it evaluates to (an approximation of) the probability that the condition holds. We then replace the marginal probability p for for the corresponding a_i 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 will pick this action variable first, so we comply with the forced action constraint. In an aggregate state we potentially increase p to be as high as the probability that the condition holds v. Note that this implementation supports the conformant algorithm in the same manner as the action of the first step so no distinction is needed.

Our implementation for an implied disjunction of action variables, and for an implication with $\sum a_i \geq 1$ on the right hand side is similar. 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.

Finally we give a separate treatment to constraints where the condition is always true and the outcome is a disjunction. For example this includes $\sum a_i = 1$ without an associated condition. If we used 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 marginal probabilities and gradients which is 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.

Conclusions

This paper describes variants of the SOGBOFA system that participated in the IPC 2018. Two algorithmic extensions were developed including lifting the computation for speedup and the use of the conformant heuristic to improve the quality of the search. The new form of action preconditions and action constraints included in the IPC required extensions to handle forced actions and an implied disjunction of possible actions. These are naturally implemented within the current system by modifying calculated marginal probabilities to enforce a logical or probabilistic form of the constraints.

Acknowledgments

This work was partly supported by NSF under grant IIS-1616280.

References

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

Cui, H., and Khardon, R. 2016. Online symbolic gradient-based optimization for factored action MDPs. In *Proc. of the International Joint Conference on Artificial Intelligence*.

Cui, H., and Khardon, R. 2018. Stochastic planning, lifted inference, and marginal MAP. In *Workshop on Planning and Inference help with AAAI*.

Cui, H.; Khardon, R.; Fern, A.; and Tadepalli, P. 2015. Factored MCTS for large scale stochastic planning. In *Proc. of the AAAI Conference on Artificial Intelligence*.

Duchi, J. C.; Shalev-Shwartz, S.; Singer, Y.; and Chandra, T. 2008. Efficient projections onto the l_1-ball for learning in high dimensions. In *Proceedings of the International Conference on Machine Learning*, 272–279.

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

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

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

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 *AAAI*.

Tesauro, G., and Galperin, G. R. 1996. On-line policy improvement using monte-carlo search. In *Advances in Neural Information Processing Systems*, 1068–1074.

Wang, W., and Carreira-Perpiñán, M. Á. 2013. Projection onto the probability simplex: An efficient algorithm with a simple proof, and an application. *CoRR/arXiv* abs/1309.1541.