Planning with DiSProD for the IPC 2023

Palash Chatterjee, Ashutosh Chapagain, Roni Khardon

Indiana University, Bloomington
{palchatt, aschap, rkhardon}@iu.edu

Abstract

DiSProD is an on-line planner suitable for hybrid domains. It builds a computation graph capturing an approximation of the distribution over future trajectories and rewards, conditioned on a probabilistic open-loop policy. At each decision step, DiSProD performs a parallel search over multiple policies, each optimizing the approximate cumulative reward by differentiating through the computation graph, and then uses the first action from the maximizing policy. For the competition, we introduce some variations on the original algorithms, including propagation of individual stochastic trajectories and a parallel search over gradient step sizes.

Introduction

Planning is one of the key problems in artificial intelligence enabling agents to make informed decisions in dynamic and complex environments. A key distinction is between *online planners* and *offline planners*. Online planners perform an optimization over a finite horizon at every decision step but execute only the first action from the solution. This process is repeated at each step of execution. On the other hand, offline planners compute a complete solution at once and then apply it at all future steps.

DiSProD (Chatterjee et al. 2023) is an online planner developed for environments with probabilistic transitions in hybrid state and action spaces. It generalizes the ideas from the SOGBOFA algorithm (Cui, Keller, and Khardon 2019) that was restricted to domains with binary variables. DiSProD builds a symbolic graph that captures the distribution of future trajectories and rewards, conditioned on a given policy. The distributions are approximated using a factored product representation, in which binary variables are captured by their marginals, as in SOGBOFA, and continuous variables are captured using their mean and variance. To facilitate this, all sources of randomness in the transition function are first externalized (referred to as *encapsulation*), and then rewritten using a Taylor approximation. This approximate representation enables calculation of propagation of distributions which constitute the symbolic graph. The construction is illustrated further in Figure 1. The symbolic graph provides a differentiable representation of the policy's value, enabling efficient gradient-based optimization for long-horizon search. The propagation of approximate distributions can be seen as an aggregation of many trajectories, making it well-suited for dealing with sparse rewards and stochastic environments.

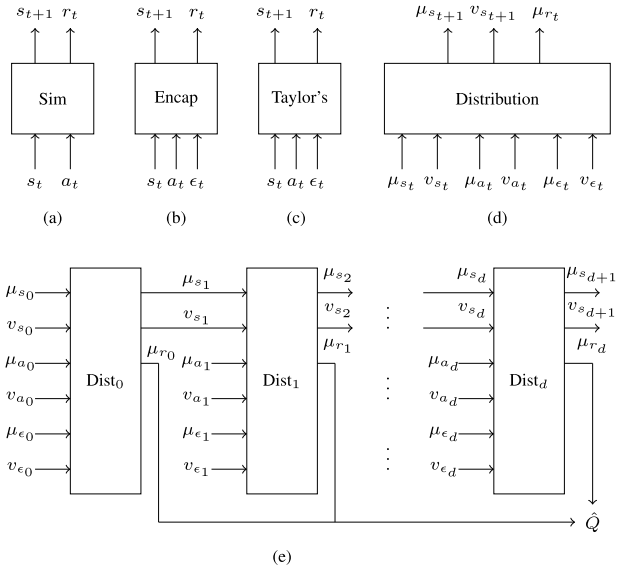

Figure 1: Schematic overview of the idea behind the construction of analytic computation graph. (a) and (b) show the original probabilistic simulator, and its *encapsulated* variant where noise variables are represented as inputs. Using Taylor's approximation of this variant, we generate a third representation (c) of the same transition function. All these take as input a concrete state, action (and noise) values and compute the next state. The key idea is to combine the approximation with propagation of distributions to yield (d) that takes distributions on states, actions, and noise as input and produces a distribution over next states. Stacking this model up to the desired search depth yields symbolic propagation of distributions (e).

Method

Here, we give an overview of the variants of the DiSProD system as entered for the probabilistic track of the Interna-

tional Planning Competition (IPC), 2023.

DiSProD was written in Python using the JAX (Bradbury et al. 2018) framework and expects an *encapsulated* and differentiable transition model. Hence, for the competition, we make use of the `JaxRDDLCompiler` which was provided to participants as part of pyRDDLGym (Taitler et al. 2022); we do not use the planner, and only use the compiled model to avoid the need to parse the RDDL code. The original RDDL transition model samples noise variables implicitly. To expose the noise variables to DiSProD, a copy of the RDDL transition function is altered, on the fly, such that the simulator is not affected and DiSProD has a compiled version of the transition function in the format it expects.

Two variants were explored with the original DiSProD system (Chatterjee et al. 2023). DiSProD-C uses a second Taylor expansion when propagating distributions over trajectories, while DiSProD-NV zeroes out the variance terms in the second order Taylor expansion, effectively propagating the means of the distributions. DiSProD-NV yields smaller computation graphs and is significantly faster, whereas DiSProD-C has a tighter approximation which is supposed to yield more accurate value estimates. The competition setup allows a relatively short time per step. We therefore developed a third variant, DiSProD-S, which can be seen as an intermediate variant. The idea is to modify DiSProD-NV to stack up *encapsulated* versions of the transition function to construct the computation graph, so that instead of propagating the mean, we propagate a stochastic trajectory. This should have a similar time cost to DiSProD-NV. It has the advantage of sampling from the distribution in the search, but runs the risk of high variance in search results. In this light, DiSProD-C can be seen as an approximate aggregation of all possible trajectories which is stable, albeit possibly biased.

Embedding prior knowledge and Tuning the planner

The main parameters in the system are search depth, number of restarts, gradient step size and the search variant (-C, -NV, -S as above). Another parameter that impacts the performance, is the weight multiplier (w) used in the approximations of non-differentiable functions.[1] As the competition restricts the time allowed per episode, a tradeoff is required between these parameters.

IPC 2023 allowed for early tuning of parameters before the evaluation phase. The search depth and the number of restarts was fixed for each domain based on explorations of the released instances. Note that DiSProD optimizes multiple policies in parallel using the idea of restarts. For the IPC, we introduced a new heuristic, exploiting the idea of parallel search to perform the search over multiple gradient step sizes as well. To decide what gradient step sizes to search over, we compute the mean value of gradients of the action variables $(\nabla \boldsymbol{a})$ in the computation graph using a random observation from the environment and a random action initialization. We use this as a heuristic and fix three gradient step sizes - $1/(10\nabla \boldsymbol{a}), 1/\nabla \boldsymbol{a}, 10/\nabla \boldsymbol{a}$. Notice that this gives us a matrix of step-sizes where each row represents the step size for the corresponding restart and each column represents the step-size to be used for that particular action variable. In this way, unlike the original system, each optimization step also chooses among the step sizes based on the improvement in the approximate Q value. We observed that while this heuristic very loose, it works well in most of the domains.

IPC 2023 had an exploration phase during where 3 instances from each of the domains were released. Experimenting with these instances, we realized that the performance of the planner was better when $w = 3$ or 5. Further, we observed that the performance of the -S and -NV variants were similar while -C variant mostly timed out. As discussed in (Chatterjee et al. 2023), the main bottleneck of the -C variant is the size of the transition function which is typically related to the number of state and action variables. In the competition, all the domains either had a lot of state variables or a lot of intermediate variables leading to large transition function. We hypothesize this to be the reason for the timeouts for the -C variant. Prior to the evaluation of an instance, all planners were provided an hour of time according to the competition rules. To decide on the best variant (among -NV and -S) and the best w (3 or 5) for a particular instance, we evaluate the planner 5 times using these combinations and pick the one with the best average reward.

For some instances of RecSim with more than 2500 state variables and more than 250 action variables, we include a fallback method which reduces the search depth and the number of restarts to 2 since otherwise even the -S and -NV variant were timing out. By default the action space is boolean of size $n_c \times n_i$ where n_c and n_i are number of customers and number of items respectively. To reduce the dimensionality of the action space for the planner, we include a translation layer that transforms the boolean action space to a continuous one of size n_c with range $[0, n_i]$.

External libraries

At present, the only major external libraries DiSProD uses are JAX and OmegaConf (for parsing configuration files). The competition specific solution also makes use of pyRDDLGym (Taitler et al. 2022) for parsing RDDL files.

Acknowledgements

This work was partly supported by NSF under grants 2002393 and 2246261. Some of the experiments in this paper were run on the Big Red computing system at Indiana University, supported in part by Lilly Endowment, Inc., through its support for the Indiana University Pervasive Technology Institute.

[1] As discussed in (Chatterjee et al. 2023), changing the multiplier in the reward function can lead to a dense reward signal that enables faster planning in some domains. In the compiler used for IPC, a single scalar, whose value we control, impacts the multipliers in all the non-differentiable functions, and not just the reward.

References

Bradbury, J.; Frostig, R.; Hawkins, P.; Johnson, M. J.; Leary, C.; Maclaurin, D.; Necula, G.; Paszke, A.; VanderPlas, J.; Wanderman-Milne, S.; and Zhang, Q. 2018. JAX: composable transformations of Python+NumPy programs.

Chatterjee, P.; Chapagain, A.; Chen, W.; and Khardon, R. 2023. DiSProD: Differentiable Symbolic Propagation of Distributions for Planning. In *Proceedings of the International Joint Conference on Artificial Intelligence*.

Cui, H.; Keller, T.; and Khardon, R. 2019. Stochastic planning with lifted symbolic trajectory optimization. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 29, 119–127.

Taitler, A.; Gimelfarb, M.; Jeong, J.; Gopalakrishnan, S.; Mladenov, M.; Liu, X.; and Sanner, S. 2022. pyRDDLGym: From RDDL to Gym Environments. arXiv:2211.05939.