Combined Task and Motion Planning Via Sketch Decompositions -Supplementary Material-

Primary Keywords: *(3) Robotics*

This supplementary material contains a Background section and details of the extension of the available implementation of PDDLStream planners. Demonstration videos can be downloaded in https://anonymfile.com/bXOa/videos-ctmp-sketches.zip

Background

We review the notions of width and sketches developed in the setting of classical planning.

Classical Planning

A *classical planning* problem is a pair $P = (D, I)$ where D is a first-order *domain* with action schemas defined over predicates and I contains the objects in the instance and two sets of ground literals, the initial and goal situations Init and Goal respectively (Geffner and Bonet 2013; Ghallab, Nau, and Traverso 2016). An instance P defines a state model $S(P) = (S, s_0, G, \mathit{Act}, A(\cdot), f(\cdot,\cdot))$ where the states in S are the truth valuations over the ground atoms represented by the set of literals that they make `true`, the initial state s_0 is Init, the set of goal states G are those that make the goal literals in Goal `true`, and the actions Act are the ground actions obtained from the schemas and objects. The ground actions in $A(s)$ are the ones that are applicable in a state s; namely, those whose preconditions are `true` in s, and the state transition function f maps a state s and an action $a \in A(s)$ into the successor state $s' = f(a, s)$. A *plan* π for P is a sequence of actions $a_0, \ldots, a_n$ that is executable in s_0 and maps the initial state s_0 into a goal state; i.e., $a_i \in A(s_i)$, $s_{i+1} = f(a_i, s_i)$, and $s_{n+1} \in G$. A state s is *solvable* if a plan exists starting at s, otherwise it is *unsolvable* (also called *dead-end*). The plan *length* is the number of actions, and a plan is *optimal* if no shorter one exists.

Width

IW(1) is a simple planning algorithm that makes use of a set of atoms or Boolean features F to implement a breadth-first search where a newly generated state is pruned if it does not make `true` an atom or feature in F for the first time in the search (Lipovetzky and Geffner 2012). The procedure IW(k) is like IW(1) but prunes a state instead when it does not make `true` a collection of up to k atoms for the first time in the search. Unlike breadth-first search, IW(k) runs in time and space that are exponential in k and not in the number of problem variables. It has been found effective in classical planning because IW(k) for $k = 1$ or $k = 2$ is complete and optimal for many planning domains when goals are single atoms. Serialized Iterated Width (SIW) uses IW(k) with small values of k for problems with conjunctive goals, that are tackled in sequence. SIW starts at the initial state $s = s_0$ of P, and performs an IW(k) search from s to find a shortest path to a state s' such that $\#g(s') < \#g(s)$, where $\#g(s)$ counts the number of unsatisfied top-level goals of P in s. If s' is not a goal state, s is set to s' and the loop repeats.

These methods have been found effective in classical planning, because many classes of problems have a small *width*, over which polynomial IW(k) algorithms are very effective, and that many other problems can be easily split into problems of small width (Lipovetzky and Geffner 2017). This is the decomposition exploited by SIW.

The width $w(P)$ of a planning problem P is the minimum k for which there exists a sequence $t_0, t_1, \ldots, t_m$ of atom tuples t_i from P, each consisting of at most k atoms, such that: 1) t_0 is `true` in the initial state s_0 of P, 2) any optimal plan for t_i can be extended into an optimal plan for t_{i+1} by adding a single action, and 3) if π is an optimal plan for t_m, then π is an optimal plan for P. The algorithm IW runs IW(k) sequentially for $k = 1, 2, \ldots$ until the problem is solved or k is the number of problem variables. IW solves a problem P in time and space that are exponential in $w(P)$. For convenience, when a problem P is unsolvable, $w(P)$ is set to the number of variables in P, and when P is solvable in at most one step, $w(P) = 0$ (Lipovetzky and Geffner 2012; Bonet and Geffner 2021). The width of a conjunction of atoms T (or set of goal states S') is defined as the width of a problem P' that is like P but with the goal T (resp. S').

An important feature of width-based planning is that, unlike other classical planning methods, it does not require a declarative, PDDL-like, action model, and just requires a set of Boolean state features F. This allows width-based planning methods to be applied in domains for which a black-box simulator computes the next states as in video games (Lipovetzky, Ramirez, and Geffner 2015).

Sketches

The SIW algorithm decomposes problems into subproblems which are solved by running IW. The decomposition is done

by exploiting the structure of conjunctive goals and requiring to achieve one more top goal in each iteration. There are problems however that require more subtle forms of decompositions. The language of sketches has been introduced recently for expressing such decompositions in a convenient, compact way by means of simple rules over features.

A sketch R is a collection of rules of the form $C \mapsto E$ where the condition C and the effect expression E are defined over a set of state features F that can be Boolean or numerical, taking values over the non-negative integers (Bonet and Geffner 2021; Drexler, Seipp, and Geffner 2021). The condition C may contain expressions of the form p or $\neg p$, for a Boolean feature p in F or expressions $n = 0$ or $n > 0$, for a numerical feature n in F. Likewise, the effects E may contain expressions of the form p, $\neg p$, and $p?$ for for a Boolean feature p in F, and expressions of the form $n\downarrow$, $n\uparrow$, or $n?$, for a numerical feature n in F. If $F = \{p, q, n, m\}$, the meaning of a rule $\{\neg p, n > 0\} \mapsto \{p, n\downarrow, m?\}$ is that in a state s where p is false and n is positive, a *subgoal state* s' needs to be found from s where p is `true`, n decreases in value relative to its value in s, and m can change arbitrarily. Features like $q \in F$ that are not mentioned in the effects cannot change, and its value in s and s' must be the same.

More precisely, a sketch R made of rules $r : C \mapsto E$ over a set of features F, defines for each state s of a problem P, a subproblem $P[s, G_R(s)]$ that is like P but with initial state s and a set of subgoal states $G_R(s)$. This set is given by the states s' such that the pairs $[s, s']$ satisfies a rule $C \mapsto E$ in R; namely s makes C `true`, and the pair $[s, s']$ makes the effect expression E `true`[1]. For instance, if $E = \{p, n\downarrow, m?\}$ and the set of features is $F = \{p, q, n, m\}$, then $[s, s']$ makes E `true` if p is `true` in s', the value of n in s' is lower than the value of n in s, and the value of the feature q, not mentioned in E, is the same in s and s'.

The goal decomposition that underlies the SIW$_R$ algorithm is given by a sketch R with a single rule $\{\#g > 0\} \mapsto \{\#g\downarrow\}$, where $\#g$ is a feature counting the number of unachieved top goals, and which defines the set of subgoals $G_R(s)$ from a state s as the states s' when the number of unachieved top goals has been decreased.

Sketch Width

By decomposing problems, sketches can be used more flesibly with IW search methods than SIW. In particular, SIW$_R$ uses a given sketch R for solving a class $\mathcal{Q}$ of problems P in the following way. Starting at the initial state $s = s_0$ of P, an IW search is run to find a closest state s' in $G_R(s)$. If s' is not a goal state of P, s is set to s', and the process repeats until a goal state is reached. The width of a sketch R over a class of problems $\mathcal{Q}$ is given by the maximum width over the subproblems $P[s, G_R(s)]$ that may be encountered in the SIW$_R$ search. The SIW$_R$ algorithms thus solves problems P in time and space exponential in the sketch width. The use of hand-made sketches in planning and ways for learning sketches of bounded width are considered in (Drexler, Seipp, and Geffner 2021, 2022). For these complexity bounds, the features are assumed to be linear in N; namely, they must have a linear number of values at most, computable in linear time (Bonet and Geffner 2021).

[1]Usually $G_R(s)$ also contains the goal states of the problem P, but we do not need this extension in this work.

PDDLStream Comparison Details

We compared our approach with PDDLStream (Garrett, Lozano-Pérez, and Kaelbling 2020), and it is important to remark that there exists a lack of work on cross-evaluation within the TAMP community. Although the selected benchmark (Lagriffoul et al. 2018) aims to be a standard for the community is not yet widely adopted. Thus, we decided to contribute also by evaluating an important work as PDDLStream in this benchmark.

PDDLStream is an AI planning framework with an action language and algorithms tailored for planning in scenarios involving sampling procedures. It enhances PDDL by introducing declarative stream specifications and solves problems without requiring detailed sampler descriptions. Its primary use case was in general-purpose robot TAMP. This planning framework comes with four different algorithms: *Incremental*, *Focused*, *Binding* and *Adaptive*.

The *Incremental* algorithm, iteratively increases the number of stream evaluations that are required to certify a fact. Furthermore, it eagerly and blindly evaluates all stream instances (required in a certain iteration) producing many facts that are irrelevant to the task. The other three algorithms belong to the category of Optimistic Algorithms, which means that they lazily explore candidate plans before checking their validity. Note that, in our work, we also propose lazy-evaluation of the geometric constrains until a plan candidate is found (refer to the main part of the paper for a complete description of this procedure). *Focused* is the starting point of the other two optimistic algorithms. It evaluates streams instances that have satisfied domain facts and adds new certified facts until a all plan actions are certified and a solution is found. *Binding* algorithm propagates stream outputs that are inputs to subsequent streams to evaluate more of the stream plan at once. Finally, *Adaptive* balances the time spent in searching versus processing the streams, often reducing the number of search calls required to find a solution.

We extended the available PDDLStream software[2] to run the different algorithms in the multiple problem instances and variations of the benchmark (Lagriffoul et al. 2018). In particular, we modeled the different scenarios in the simulator Pybullet (Coumans and Bai 2016–2021). It is important to mention that we have not been able to incorporate the fence that surrounds the scenario in the defined benchmark in these experiments. Nevertheless, this benefits the PDDLStreams methods since with this modification the robot is less restricted and have more opportunities to interact with the environment objects. Thus, it has a little advantage particularly in cluttered environments. Furthermore we defined and implemented the goal regions and conditions and solved multiple integration issues.

In the Experimentation section, the evaluation results are reported. The more complex benchmark instances are not solved by any of the algorithms given the 30 min and 16 GB

[2]Find the repository in https://github.com/caelan/pddlstream.

RAM budgets. In the case of the *Incremental* algorithm, the timeout was exceeded and on the other cases the memory capacity was exceeded approximately when the 60% of the time budget was reached.

References

Bonet, B.; and Geffner, H. 2021. General Policies, Representations, and Planning Width. In *Proc. AAAI Conf. Artif. Intell.*, 11764–11773.

Coumans, E.; and Bai, Y. 2016–2021. PyBullet, a Python module for physics simulation for games, robotics and machine learning. http://pybullet.org.

Drexler, D.; Seipp, J.; and Geffner, H. 2021. Expressing and Exploiting the Common Subgoal Structure of Classical Planning Domains Using Sketches. In *Proc. Int. Conf. Principles of Knowledge Representation and Reasoning*, 258–268.

Drexler, D.; Seipp, J.; and Geffner, H. 2022. Learning Sketches for Decomposing Planning Problems into Subproblems of Bounded Width. In *Proc. Int. Conf. Automated Planning and Scheduling*, 62–70.

Garrett, C. R.; Lozano-Pérez, T.; and Kaelbling, L. P. 2020. PDDLStream: Integrating Symbolic Planners and Blackbox Samplers via Optimistic Adaptive Planning. In *Proc. Int. Conf. Automat. Plan. and Scheduling*, volume 30, 440–448.

Geffner, H.; and Bonet, B. 2013. *A Concise Introduction to Models and Methods for Automated Planning*. Springer.

Ghallab, M.; Nau, D.; and Traverso, P. 2016. *Automated planning and acting*. Cambridge University Press.

Lagriffoul, F.; Dantam, N. T.; Garrett, C. R.; Akbari, A. A.; Srivastava, S.; and Kavraki, L. E. 2018. Platform-Independent Benchmarks for Task and Motion Planning. *IEEE Robotics and Automation Letters*, 3: 3765–3772.

Lipovetzky, N.; and Geffner, H. 2012. Width and Serialization of Classical Planning Problems. In *Frontiers in Artif. Intell. and Applications*, volume 242, 540–545.

Lipovetzky, N.; and Geffner, H. 2017. Best-First Width Search: Exploration and Exploitation in Classical Planning. In *Proc. AAAI Conf. Artif. Intell.*

Lipovetzky, N.; Ramirez, M.; and Geffner, H. 2015. Classical Planning with Simulators: Results on the Atari Video Games. In *Proc. Int. Joint Conf. Artif. Intell.*