Learning to Search in Task and Motion Planning with Streams

Mohamed Khodeir*† Ben Agro*† Florian Shkurti†

***Abstract*— Task and motion planning problems in robotics typically combine symbolic planning over discrete task variables with motion optimization over continuous state and action variables, resulting in trajectories that satisfy the logical constraints imposed on the task variables. Symbolic planning can scale exponentially with the number of task variables, so recent works such as PDDLStream [1] have focused on optimistic planning with an incrementally growing set of objects and facts until finding a feasible trajectory. However, this set is exhaustively and uniformly expanded in a breadth-first manner, regardless of the geometric structure of the problem at hand, which makes long-horizon reasoning with large numbers of objects prohibitively time-consuming. To address this issue, we propose a geometrically informed symbolic planner that expands the set of objects and facts in a best-first manner, prioritized by a Graph Neural Network based score that is learned from prior search computations. We evaluate our approach on a diverse set of problems and demonstrate an improved ability to plan in large or difficult scenarios. We also apply our algorithm on a 7DOF robotic arm in several block-stacking manipulation tasks.**

I. Introduction

Task and motion planning (TAMP) involves searching over both symbolic actions that determine a high-level task sequence and low-level motion controls that result in feasible trajectories [2]–[4]. The symbolic task planner operates on high-level abstractions of the environment, including object definitions, operations that can be applied to them, and pre- and post-conditions that these operations have to satisfy. On the other hand, the motion planner requires a non-abstracted description of its surrounding environment. The motion planner informs the symbolic planner about the kinematic and dynamic feasibility of a proposed coarse task plan, possibly leading to backtracking.

This interplay between high and low level planning has a significant effect on the total runtime of discovering a solution. Symbolic planning can scale exponentially with the number of objects and task variables, making an exhaustive search over task sequences prohibitive. In this paper, we propose a geometrically informed symbolic planner that learns to search over possible task sequences based on previous planning computations. We present an algorithm for solving TAMP problems expressed in *PDDLStream* [1], a recently proposed domain-independent planning framework.

The Planning Domain Definition Language (PDDL) requires an a priori discretization of the state space and does not allow for continuous variables [5]. PDDLStream extends PDDL by introducing the concept of *streams*, conditional generators that model black-box sampling procedures. Streams enable continuous quantities such as poses and configurations to be created dynamically, allowing planning without the need for pre-discretization. Existing PDDLStream algorithms use streams to grow the planning problem in a breadth-first manner until a feasible trajectory can be found. However, this can be prohibitively expensive in tasks which require long-horizon reasoning with large numbers of objects.

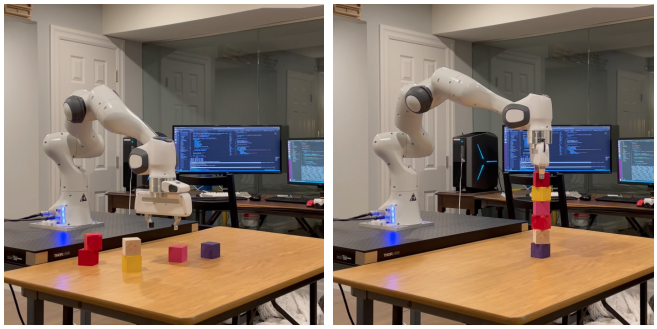


Fig. 1: Our planner solves the six-block stacking problem with a 7DOF robot arm[1]. Existing TAMP methods struggle to find a solution for more than four blocks (see Table II). By learning from easier intermediate planning problems, we show improved performance and runtime for larger scenes.

To address this shortcoming, we present a learning-based method to guide a best-first search in the space of streams. Our main contributions are (a) a Graph Neural Network (GNN) architecture that scores the relevance of streams, and (b) a queue-based algorithm that prioritizes their addition to the planning problem. The model is trained on prior planning experience from solving similar problems.

We compare `INFORMED` against the best performing existing PDDLStream algorithm across two simulated environments: Kitchen and Blocks World [6]. Our experiments show that `INFORMED` outperforms baselines on previously unseen problems, finding feasible plans faster, including on problems of larger size, where the baseline fails to find solution. Finally, we show this technique works on a real 7DOF robotic arm.

II. Related Work

Integrated task and motion planning. There is a vast literature on the problem of integrating the geometric reasoning required by motion planning with the symbolic reasoning that is necessary for planning to achieve abstract

*Authors contributed equally.

†Robot Vision and Learning Laboratory, University of Toronto Robotics Institute. {m.khodeir, ben.agro}@mail.utoronto.ca, florian@cs.toronto.edu

[1]Code, additional results, and videos of the experiments can be found here https://rvl.cs.toronto.edu/learning-based-tamp/

goals; see [2] for a detailed taxonomy. Our work builds on PDDLStream [1], which we introduce in more detail in the background section, but there is a long history of prior research, including [7] [8], which combine symbolic planners with complete geometric planner, and even older ideas on domain-specific hierarchical reasoners and planners for machining [9]. The need for selecting the right hierarchical abstractions for symbolic planning and favoring feasibility and real-time results over optimality was emphasized in [3]. Logic Geometric Programming combined symbolic planning and trajectory optimization [4], [10], [11], even for dynamic physical motions involving tool use, while [12] integrated sampling procedures with SAT solvers as symbolic planners.

Learning for TAMP. Motivated by the success of learning in the context of robotics, recent work has sought to combine the ability of TAMP systems to plan for novel temporally extended goals with learning methods. There are several orthogonal avenues of research under this umbrella: methods which learn capabilities that may be difficult to engineer (e.g. a pouring action) [14], those which learn the symbolic representations with which to plan [15] [16], those that integrate perception learning and scene understanding into TAMP [17], [18], and those which attempt to learn search guidance from experience [19] [20] [21]. Similar in spirit to our work, [19] tries to guide the search for action skeletons by learning to predict the credibility of a sequence of discrete actions directly from visual observations of the scene, using these predictions as a heuristic in a best-first search for action sequences.

In contrast, we seek to learn an efficient heuristic to guide the construction of an optimistic planning problem with a minimal set of irrelevant facts. We leverage an off-the-shelf domain-independent search sub-routine, thus our work can be synergistically combined with methods like [19] [20] [21] that learn a domain-specific heuristic.

Planning with many objects. The motivation of our work shares similarities with [22] and [23], both of which attempt to speed up task planning by learning models which exclude irrelevant objects or actions (respectively) from the initial problem representation. Our work can be seen as an extension of such methods to the case when the full planning problem is not given a priori, and instead needs to be constructed from a set of initial facts and a set of streams that can generate additional facts.

III. Background

A PDDLStream [1] problem $(\mathcal{P}, \mathcal{A}, \mathcal{S}, \mathcal{O}, \mathcal{I}, \mathcal{G})$ is given by a set of predicates $\mathcal{P}$, actions $\mathcal{A}$, streams $\mathcal{S}$, initial objects $\mathcal{O}$, an initial state $\mathcal{I}$, and a goal state $\mathcal{G}$. Similarly to PDDL, a *predicate*, p is a boolean function. An instance of a predicate $p(\bar{x})$ applied on a tuple of objects $\bar{x} = \langle x_1, \ldots, x_n \rangle$ is called a *fact*. PDDL *objects* are references to entities in the planning problem. These can refer to anything from the identities of physical objects, to continuous quantities or even complex data structures such as those representing trajectories. A *state* is a set of facts. Both $\mathcal{I}$ and $\mathcal{G}$ are examples of states. An action a is a tuple of three components: parameters $\bar{X}$, a set of preconditions $\text{pre}(a)$, and a set of effects $\text{eff}(a)$. An action instance $a(\bar{x})$, whose parameters $\bar{X}$ have been assigned to particular objects $\bar{x}$, is only applicable/available in a given state if it satisfies the set of facts in the action's preconditions $\text{pre}(a(\bar{x}))$. The set of facts $\text{eff}(a(\bar{x}))$ specify how a state will change after $a(\bar{x})$ is applied. A solution to the planning problem is a sequence of valid action instances $\pi = [a_1(\bar{x}_1), \ldots a_N(\bar{x}_N)]$ that transform $\mathcal{I}$ into $\mathcal{G}$. A helpful concept is the *preimage* of a plan; the set of facts that must hold in order for the entire plan to be applicable in a given initial state. This is defined as follows:

$$\text{PREIMAGE}(\pi) = \cup_{i=1}^{N} \big(\text{pre}(a_i(\bar{x}_i)) - \cup_{j<i} \text{eff}(a_j(\bar{x}_j))\big).$$

Example. To describe the environment depicted in Fig. 2, we might define predicates such as `on-block(?b1 ?b2)` and `at(?b ?`X_{WB}`)`. We could then define an action that stacks one block onto another as `stack(?q ?b ?lb ?`X_{WL}`)` with preconditions that include `clear(?lb)` and `at(?lb ?`X_{WL}`)`, requiring that the lower block is clear and at the given world pose X_{WL}. One effect of this action might be `on-block(?b ?lb)`, but there may be other side-effects such as a new configuration of the robot arm. To describe a particular stacking problem in this environment, we simply need to specify objects $\mathcal{O} = \{b_1, b_0, b_2, a, s_0, s_1, q_0, \ldots\}$, initial state $\mathcal{I} = \{$ `on-block(`$b_1\, b_0$`), at(`$b_0\, X_{Wb_0}$`)`$, \ldots\}$, and goal state $\mathcal{G} =$ {`on-surface(`$b_0\, s_1$`), on-surface(`$b_1\, s_1$`)`}.

Streams. The set of streams, $\mathcal{S}$, distinguishes a PDDLStream problem from traditional PDDL. Streams are conditional generators which yield objects that satisfy specific constraints conditioned on their inputs. Formally, a stream, s, consists of input parameters $s.input$, a set of domain predicates $s.domain$, output parameters $s.output = \bar{o}$, and a set of certified predicates $s.certified$. $s.domain$ is the set of predicates that must evaluate to true for an input tuple $s.input = \bar{x}$ to be valid. This ensures the correct types of objects (e.g., configurations, poses etc.) are provided to the generators. $s.certified$ are predicates on $s.input$ and $s.output$ that assert facts that $\langle \bar{x}, \bar{o} \rangle$ always satisfy.

Example. In our block stacking environment (Fig. 2), we can define a stream `find-place(?b, ?s)` $\rightarrow$ `?`X_{WB} which takes as input a block `?b` and a surface `?s`, and produces a world pose `?`X_{WB}, at which `?b` would be on `?s`. The `domain` facts of this stream would restrict the types of its input objects (i.e. `find-place.domain = {block(?b), surface(?s)}`), and the set of certified facts might contain the single fact `block-support(?block ?surface ?`X_{WB}`)`. Given such a stream, a planner could dynamically request placement poses for any object/surface pair.

Streams can be applied recursively, to generate a potentially infinite set of objects and their associated facts, starting from those in $\mathcal{I}$ and $\mathcal{O}$, respectively. A crucial aspect of PDDLStream problems is that there usually does not exist a plan whose *preimage* is a subset of $\mathcal{I}$. Instead, solvers of PDDLStream problems must use the streams in $\mathcal{S}$ to recursively expand $\mathcal{I}$ to I^* such that there exists a plan π

whose *preimage* is a subset of I^*. This is reminiscent of theorem proving problems that start from a set of axioms and use rules of inference to expand the set of known theorems.

We use the term stream *instantiation* to refer to the act of creating a new instance of a stream on a specific object tuple. We distinguish this from *evaluation* of a stream instance, which refers to the act of invoking the associated sampler on the given inputs. When a stream instance is evaluated, it produces *grounded* objects (i.e. with a concrete value). A stream instance can only be evaluated if its inputs are themselves grounded. Prior to evaluation, a stream instance's output objects (and associated certified facts) are said to be *optimistic*.

Existing PDDLStream algorithms expand the set of stream *results* (objects and associated certified facts) in a breadth-first manner. On a given iteration, all streams are instantiated up to a certain depth l, and a PDDL planner is invoked on the resulting set. If no plan is found, the next iteration starts with $l \leftarrow l + 1$. If a plan π^*, is found, the sequence of stream instances that support the plan, ψ (the *stream plan*), is extracted, and the stream instances are evaluated. If all stream evaluations succeed, then the plan π is returned. Otherwise, grounded stream results are added to the set of facts used for planning, and the next iteration begins.

IV. Our Approach

Our approach aims to use a learned model to guide the expansion of optimistic facts in a PDDLStream problem, leading to smaller intermediate planning problems. We first present a PDDLStream algorithm which uses a priority queue to order the instantiation of streams. Second, we formulate the problem of assigning a priority to optimistic stream instantiations as a learning problem. Finally, we propose a model architecture which makes use of the structure of streams to recursively build up representations of optimistic objects, and assign priorities to them simultaneously.

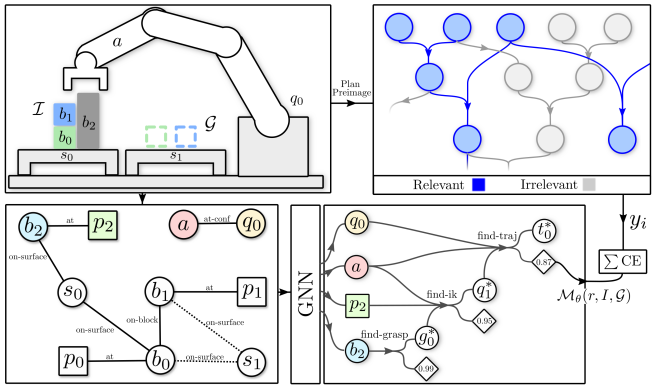


Fig. 2: Top row: A block stacking task is solved and a set of labeled stream results is extracted from the plan. Bottom left: the Problem Graph representation of the problem. Bottom right: GNN-Stream is trained to predict relevance of optimistic stream results.

A. Our Algorithm

Our `INFORMED` algorithm uses a priority queue, Q, to order the addition of optimistic results to the planning problem based on a *relevance score* given by a black-box model, $\mathcal{M}_\theta$. The algorithm is composed of four distinct phases: Expansion, Planning, Sampling, and Feedback.

Expansion. In each iteration of the algorithm, the stream result with the highest score is popped off of the queue and if the result is optimistic, its certified facts are added to I^*. In doing so, we add new objects and facts to the planning problem, which can be used as inputs to yet more stream instances. The sub-routine `EXPAND` creates all the child stream instances which use these newly added objects. These are all scored and pushed onto the queue for later expansion (line 11 in Alg. 1).

Planning. In the planning phase, we invoke a PDDL planner on the current set of optimistic and grounded facts (line 17 in Alg. 1). If an optimistic plan π^* is found, it is returned along with the *stream plan* ψ, which is the sequence of stream instances that need to be evaluated to ground π^*. This subroutine, which is a light wrapper around the underlying FastDownward planner, is the same as that presented in [1]. A key issue for our algorithm is how to decide when the set of included optimistic facts is sufficient for planning to succeed. This `SHOULD-PLAN` classifier may itself be a good candidate for learning from experience. However, in this work, we simply choose to plan every time $K = 100$ optimistic and grounded facts have been added to the problem, so in our implementation `SHOULD-PLAN` returns true every K iterations.

Sampling. If a plan has just been found, or if we have found a plan in a previous iteration without yet grounding all of its parameters, then we have the option to devote time to sampling. This means running the underlying generators for each of the streams in the plan ψ. This is facilitated by the `PROCESS-STREAMS` sub-routine (line 22), of which there are several implementations provided by [1]. In this work, we employ the version used by the `ADAPTIVE` algorithm for consistency. This implementation maintains a search tree, where each node corresponds to a partial grounding of the parameters in a plan. This allows for backtracking if, for example, a grasp configuration found early in the plan causes a collision in a later step. Moreover, the algorithm will continue trying to satisfy a plan as long as time is allocated by `SHOULD-SAMPLE`, regardless of whether it is feasible.

Feedback. If in the course of sampling, the algorithm has managed to produce groundings for all the parameters in an optimistic plan, then our algorithm halts and the plan is returned (line 24).

Otherwise, we would like to update our planning problem to account for information gleaned from sampling. By running the streams that support an optimistic plan, `PROCESS-STREAMS` will produce many grounded objects that we already suspect to be relevant (because they were part of a plan). These new values are all immediately added to I_{ground} so that they can be used by the planner in the next planning phase (line 26). However, it would be prohibitively expensive to immediately expand all of these new results. Therefore, we simply score and push all them on Q for later expansion (line 28).

Finally, for each optimistic result that was processed during the course of sampling, we now either have a grounding or we know it is infeasible (because the underlying sampler failed to produce a grounding). In both cases, the corresponding optimistic results are removed from I^* so that they can no longer be used in any new plan (line 30).

Algorithm 1 INFORMED($\mathcal{A}, \mathcal{S}, \mathcal{I}, \mathcal{G}, \mathcal{M}_\theta$, PROCESS-STREAMS):

1 $I_{ground} = \mathcal{I}$.copy(), $I^* = \{\}$, $Q = []$
2 **for** $s(\bar{x}) \in$ INSTANTIATE($\mathcal{S}, \mathcal{I}$) **do**
3 r = $s(\bar{x})$.result
4 score = $\mathcal{M}_\theta(r, I_{ground} \cup I^*, \mathcal{G})$
5 Q.push($\langle score, r \rangle$)
6 **while** True **do**
7 **if** Q.len() > 0 **then**
8 $\langle score, r \rangle = Q.pop()$
9 **if** r.optimistic() **then**
10 I^*.add($r.certified$)
11 **for** $s(\bar{x}) \in$ EXPAND($r, I_{ground} \cup I^*, \mathcal{S}$) **do**
12 r = $s(\bar{x})$.result
13 score = $\mathcal{M}_\theta(r, I_{ground} \cup I^*, \mathcal{G})$
14 Q.push($\langle score, r \rangle$)
15 π^* = None
16 **if** SHOULD-PLAN() **then**
17 $\pi^*, \psi =$ OPTIMISTIC-SOLVE($\mathcal{A}, I_{ground} \cup I^*, \mathcal{G}$)
18 $t =$ SHOULD-SAMPLE()
19 **if** $t \leq 0$ and Q.len() $= 0$ **then**
20 **return** None
21 **if** $t > 0$ **then**
22 $\pi, new, processed =$ PROCESS-STREAMS($I_{ground} \cup I^*, \psi, \pi^*; t$)
23 **if** $\pi \neq$ None **then**
24 **return** π
25 **for** $r \in new$ **do**
26 $I_{ground}.add(new.certified)$
27 score = $\mathcal{M}_\theta(r, I_{ground} \cup I^*, \mathcal{G})$
28 Q.push($\langle score, r \rangle$)
29 **for** $r \in processed$ **do**
30 I^*.remove($r.certified$)

Algorithm 2 INSTANTIATE ($\mathcal{S}, \mathcal{I}$):

1 $\{s(\bar{x}) | \forall s \in \mathcal{S}, \forall p \in s.domain, |\bar{x}| = |s.input|, p(\bar{x}) \in \mathcal{I}\}$

Algorithm 3 EXPAND (r, $\mathcal{S}, \mathcal{I}$):

1 $\{s_j(\bar{x}) | \forall s_j(\bar{x}) \in$ INSTANTIATE$(\mathcal{S}, r.certified \cup \mathcal{I}), r.output \subset \bar{x}\}$

B. Relevance of Streams as Binary Classification

The question of whether a stream instance is relevant to the planning problem can be boiled down to whether any of the objects that it produces are relevant. In this work we define an object to be relevant to a PDDLStream problem if:

1) it is part of the preimage of any optimal plan
2) or, it is used as input to a stream which produces a relevant object

We note, however, that applying this definition directly would be impractical. It is often prohibitively expensive to compute even one optimal plan for problems of interest. In order to make it tractable, we apply this definition using satisficing plans found using existing PDDLStream algorithms.

Relevance of a PDDL object. In a TAMP setting, the objects in the preimage of a given plan will correspond to real valued quantities, such as the 7DOF configuration of a robotic arm. However, these values are rarely ever unique; that a specific grasp configuration of an object was used in a plan should imply that a different grasp of the same object is also important to consider. Therefore, when deciding on the relevance of an object, we concern ourselves not with the actual numeric value of the object, but with the set of constraints that act on it. For instance, we want it to be a kinematically feasible grasp configuration for a given object placed in a given location. The sequence of stream instances leading up to a given object define the complete set of constraints acting it. To determine whether an object is relevant, we simply need to check whether there exists a counterpart in a given preimage which has the same set of constraints (i.e. the same ancestor streams).

Training data. This approach gives allows us to cast the relevance of stream instances as a supervised binary classification problem. Given a set of training problems $T = \{\langle \mathcal{O}_i, \mathcal{I}_i, \mathcal{G}_i \rangle\}_{i=1}^N$ for a given domain (i.e. sharing $\langle \mathcal{P}, \mathcal{A}, \mathcal{S} \rangle$), we construct input/output pairs as follows. For each problem in $t_i \in T$:

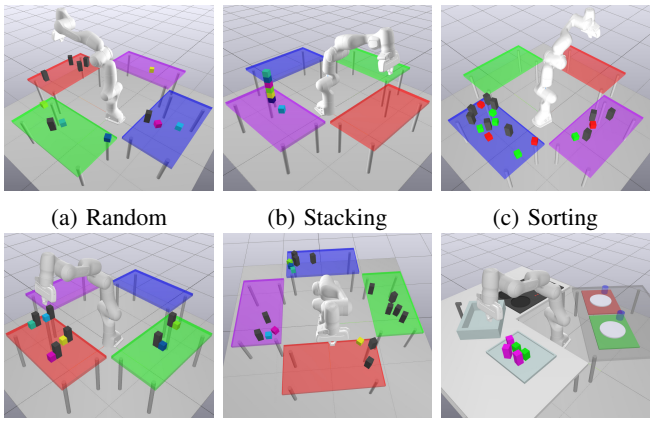
1) Run an existing PDDLStream algorithm to find a plan π_i, recording a list of stream instances $\{s_{ij}(\bar{x}_{ij})\}_{j=1}^M$ produced along the way.
2) For each s_{ij}, use PREIMAGE(π_i) to assign a binary label y_{ij} as described.
3) Save $\{\langle \mathcal{O}_i, \mathcal{I}_i, \mathcal{G}_i, s_{ij}, y_{ij} \rangle\}_{j=1}^M$ and $\langle \mathcal{P}, \mathcal{A}, \mathcal{S} \rangle$.

C. Modelling Relevance of Streams

Given the $\mathcal{I}$, $\mathcal{G}$, the current set of optimistic stream results I^* and grounded stream results I_{ground}, we want to assign a *relevance score* $y \in [0, 1]$ to a stream result $s(\bar{x}) \rightarrow \bar{o}$.

MLP representations of streams. Our model architecture is motivated by the observation that a stream represents constraint(s) between its inputs $\bar{x}$ and outputs $\bar{o}$, and that for each domain there are only a handful of explicitly defined streams. We model each stream with a simple multi-layer perceptron (MLP) M_s, which has fixed input and output sizes proportional to s.input and s.output. Given a representation of the input objects in the context of the planning problem, M_s is tasked with producing suitable embedding output objects, as well as a score representing the relevance of those outputs to the planning problem. As depicted in the bottom right of Fig. 2, these M_s can be recursively combined in exactly the same way as streams, producing relevance scores and dense vector representations for each object their associated streams create.

GNN representations of objects and facts. To obtain the representations for the initial objects in the planning problem, we employ *Graph Neural Networks* [24] owing to their ability to represent objects based on their relations to one another. The input to the GNN, which we call the *Problem Graph*, is a graph with nodes representing the PDDL objects and edges representing the facts that relate those objects to one another in both $\mathcal{I}$ and $\mathcal{G}$ (see bottom left of Fig. 2). This relational representation is similar to that used in [22], and allows our system to generalize to different numbers and identities of objects. The node feature vectors consist of the world pose of the object if it exists, and padding otherwise.



(a) Random (b) Stacking (c) Sorting

(d) Non-monotonic (e) Clutter (f) Kitchen

Fig. 3: A representative sample of the various experimental problem types.

The edge feature vectors have a one-hot encoding of their corresponding predicate, and a flag if the fact appears in $\mathcal{I}$ (1) or $\mathcal{G}$ (0). This graph representation is extensible and can include other geometric (or semantic) information about the scene.

The end-to-end model architecture $\mathcal{M}_\theta$ is depicted in the bottom row of Fig. 2. A given planning problem is used to produce a Problem Graph. A GNN produces an embedding of each object according to its relations in both the goal and the initial state. These representations can then be used as input to stream MLPs, which are combined recursively to obtain representations and scores for optimistic objects produced by their associated streams. This entire computation is differentiable, so $\mathcal{M}_\theta$ is trained end-to-end to minimize a weighted cross-entropy, giving a higher penalty to false negatives.

$$\mathcal{L}(T) = \sum_{i=1}^{N} \sum_{j=1}^{M_i} \mathcal{L}_{CE}(y_{ij}, \mathcal{M}_\theta(\mathcal{I}_i, \mathcal{G}_i, s_{ij}(\bar{x}_{ij}))) \quad (1)$$

V. Experimental Results

Experimental testing of INFORMED was conducted across six problem types in two *domains*: blocks world and kitchen. We compare our approach to the best performing PDDLStream algorithm presented in [1]. Across all problem types, we find that INFORMED consistently outperforms ADAPTIVE.

The blocks world and kitchen domains are loosely based on those outlined in [6]. Each domain has a specification of the predicates, actions and streams. Within each domain, there are a diverse set of *problems*, specified by initial ($\mathcal{I}$) and goal ($\mathcal{G}$) states. Blocks world problems are divided into five categories, each presenting unique geometric and logical challenges, all using the same domain and stream specification. Refer to Fig. 3 for visualizations of the various (sub-)domains.

A. Blocks World

The blocks world domain, with actions *move*, *pick*, *place*, *stack*, and *unstack*, supports many problems with differing challenges. There are two types of items in these problems (not differentiated logically): blocks (shorter) and blockers (taller) (Fig. 3a-3e). The robot arm is surrounded by four tables.

Stacking. Stacking problems consist of blocks initially arranged in random towers uniformly randomly distributed across the tables. The goal is to have a specific stack of all the blocks on a specific table (see Fig. 3b)

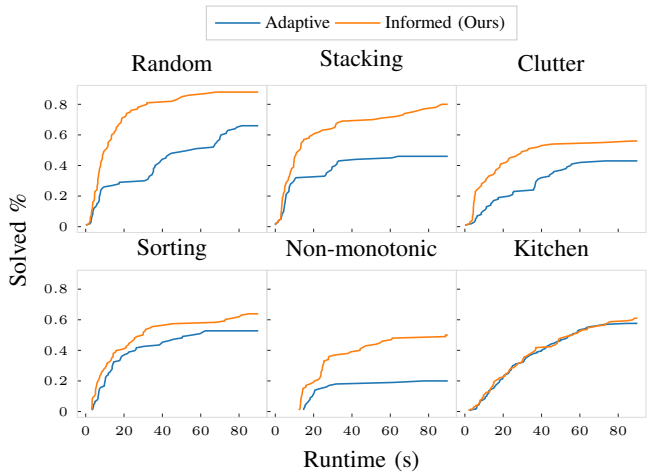


Fig. 4: A comparison of ADAPTIVE and INFORMED on percentage solved as a function of runtime.

TABLE I: Performance of INFORMED and ADAPTIVE on held-out test problems. Success rates show the percentage of problems that could be solved in the allotted time. All runs are conducted with 90s timeouts. We report the average runtimes over all problems of a given type.

		Adaptive		Informed (Ours)	
Domain	**# Probs**	**Solved (%)**	**Time (s)**	**Solved (%)**	**Time (s)**
Clutter	100	43.00	63.26	**56.00**	**48.37**
Kitchen	177	57.63	55.55	**61.02**	**54.89**
Non Monotonic	100	20.00	76.83	**50.00**	**59.44**
Random	100	66.00	51.90	**88.00**	**22.98**
Sorting	108	52.78	52.72	**63.89**	**45.53**
Stacking	100	46.00	55.22	**80.00**	**32.85**

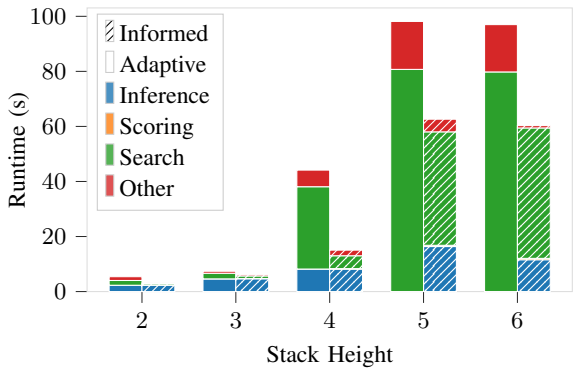


Fig. 5: **Stacking**: A breakdown of the runtimes of ADAPTIVE and INFORMED into the constituent phases.

As shown in Table II, ADAPTIVE struggles to find a solution as the number of blocks in the goal stack increases.

Fig. 5 shows that INFORMED drastically reduces the time spent in solving the induced planning problems as the stack height increases. Our system was deployed on a real Franka-Emika Panda Robot as shown in Fig. 1, and successfully tested on the stacking sub-domain. This demonstrates that the trajectories produced by INFORMED can be executed by a real robot.

TABLE II: **Stacking:** The percentage of problems solved across varying stack heights.

Stack Height	Adaptive	Informed
2	100.00	100.00
3	100.00	100.00
4	93.33	100.00
5	0.00	72.22
6	0.00	57.14

Non-monotonic. In $\mathcal{I}$, blockers and blocks come in pairs (see Fig. 3d) uniformly randomly distributed across the green and red tables. It is usually kinematically infeasible to grasp the block without first moving the blocker, but this is not known to the planner. $\mathcal{G}$ specifies where each block should be placed (on the purple or blue tables) and that each blocker be returned to its starting position (if moved).

TABLE III: **Non-monotonic:** The percentage of problems solved across varying number of blocks.

Num Blocks	Adaptive	Informed
2	0.95	1.00
3	0.05	0.50
5	0.00	0.00
6	0.00	0.00

Problems of this type test the planners' ability to incorporate feedback from sampling into the optimistic planning. For each block that has to be moved, any plan that doesn't first move the obstructing blocker will (usually) fail, and the planner must backtrack to find a new plan. As shown in Table III, INFORMED is able to solve 30 more problems than ADAPTIVE, but both fail to solve any problems containing five or more block/blocker pairs.

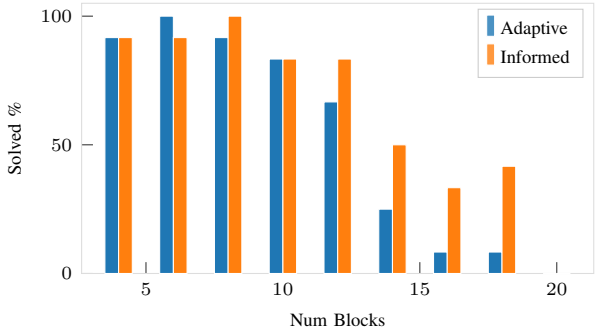


Fig. 6: **Sorting**: Percentage solved as a function of the number of blocks to be sorted.

Sorting. There are red and green blocks, and blockers, uniformly randomly distributed across the purple and blue tables (see Fig. 3c). There are no stacks in $\mathcal{I}$ or $\mathcal{G}$, and the green and red blocks must be placed on the green and red tables, respectively. The blockers must remain on their initial tables. Sorting problems contained the largest number of objects in any of our tests.

As shown in Table I, INFORMED solved 11 additional test problems, roughly a 20% improvement over ADAPTIVE. Fig. 6 shows that this is due to solving a higher proportion of problems with 12 or more blocks.

Clutter. As in sorting, blocks are initially placed at random on the tables, except that the object positions are sampled using ordered Poisson-Disc Sampling, which tends to cause the objects to clump together [25]. Each problem contains twice as many blockers as blocks, and the goal is to achieve a specific arrangement of the blocks, which may include stacks or placement on particular tables. Unlike Sorting and Non-monotonic problems, blockers do not feature in the goal, and so are only relevant in that they may obstruct motions, grasps or placements. INFORMED and ADAPTIVE solve 56% and 43% percent of problems, respectively.

Random. The Random problem type contains a mixture of the difficulties in all of the other types. The goal state may include stacks of up to 6 blocks, and problems may contain up to 6 blockers which obstruct actions. We include it to assess performance on "typical" problems which are not generated to induce any particular difficulty. INFORMED solves 88% of the test problems, compared to 66% for ADAPTIVE. Fig. 4 shows that for any choice of time limit, informed is able to solve a significantly higher proportion of problems.

B. Kitchen

A robot arm must complete various tasks in the kitchen involving different types of items (which are not differentiated logically): cabbages (green), radishes (purple), and glasses (blue). Items can be cleaned if they are in the sink, and clean items can be cooked if they are on a burner. Problems involve a random combination of items distributed randomly across available surfaces (e.g., the tray). Goals involve cooked cabbages on the plates and clean glasses on the place-mats.

Kitchen problems can be geometrically difficult (including *infeasible task actions* [6]), as objects may obstruct the arm from grasping or placing. ADAPTIVE's planning time is generally negligible in this domain, and thus there is not much room for improvement from using INFORMED. We find that INFORMED matches the performance of ADAPTIVE (see Fig. 4), suggesting that the overhead from our proposed method is negligible.

VI. Conclusion

INFORMED is a new PDDLStream algorithm that allows for a guided search in constructing optimistic planning problems using streams. We propose a method for learning to guide this search from past experience using a GNN-based model architecture. INFORMED turns the breadth-first search in the space of stream instantiations to a best-first search, more effectively discretizing the task and motion planning problem. We evaluate across two domains and a variety of problem types, and show that INFORMED outperforms existing PDDLStream algorithms. We apply this system on a real-world block stacking task using a 7DOF arm, demonstrating the viability of the produced trajectories.

References

[1] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, "Pddlstream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning," 2020.

[2] C. R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, and T. Lozano-Pérez, "Integrated task and motion planning," 2020.

[3] L. P. Kaelbling and T. Lozano-Pérez, "Hierarchical task and motion planning in the now," in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 1470–1477.

[4] M. Toussaint, K. Allen, K. Smith, and J. Tenenbaum, "Differentiable physics and stable modes for tool-use and manipulation planning," in *Proceedings of Robotics: Science and Systems*, Pittsburgh, Pennsylvania, June 2018.

[5] M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins, "Pddl–the planning domain definition language, technical report," *Yale University, New Haven, CT*, 1998.

[6] F. Lagriffoul, N. T. Dantam, C. Garrett, A. Akbari, S. Srivastava, and L. E. Kavraki, "Platform-independent benchmarks for task and motion planning," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 3765–3772, 2018.

[7] S. Cambon, R. Alami, and F. Gravot, "A hybrid approach to intricate motion, manipulation and task planning," *The International Journal of Robotics Research*, vol. 28, no. 1, pp. 104–126, 2009.

[8] E. Plaku and G. D. Hager, "Sampling-based motion and symbolic action planning with geometric and differential constraints," in *2010 IEEE International Conference on Robotics and Automation*. IEEE, 2010, pp. 5002–5008.

[9] S. Kambhampati, M. Cutkosky, M. Tenenbaum, and S. H. Lee, "Combining specialized reasoners and general purpose planners: A case study," in *AAAI*, 1991.

[10] M. Toussaint, "Logic-geometric programming: An optimization-based approach to combined task and motion planning," pp. 1930–1936, 2015. [Online]. Available: http://ijcai.org/Abstract/15/274

[11] T. Migimatsu and J. Bohg, "Object-centric task and motion planning in dynamic environments," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 844–851, 2020.

[12] N. T. Dantam, Z. K. Kingston, S. Chaudhuri, and L. E. Kavraki, "An incremental constraint-based framework for task and motion planning," *The International Journal of Robotics Research*, vol. 37, no. 10, pp. 1134–1151, 2018.

[13] K. Hauser and J.-C. Latombe, "Integrating task and prm motion planning: Dealing with many infeasible motion planning queries," in *ICAPS09 Workshop on Bridging the Gap between Task and Motion Planning*. Citeseer, 2009.

[14] Z. Wang, C. R. Garrett, L. P. Kaelbling, and T. Lozano-Pérez, "Learning compositional models of robot skills for task and motion planning," *CoRR*, vol. abs/2006.06444, 2020. [Online]. Available: https://arxiv.org/abs/2006.06444

[15] T. Silver, R. Chitnis, J. Tenenbaum, L. P. Kaelbling, and T. Lozano-Pérez, "Learning symbolic operators for task and motion planning," *arXiv preprint arXiv:2103.00589*, 2021.

[16] J. Loula, K. Allen, T. Silver, and J. Tenenbaum, "Learning constraint-based planning models from demonstrations," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 5410–5416.

[17] K. Kase, C. Paxton, H. Mazhar, T. Ogata, and D. Fox, "Transferable task execution from pixels through deep planning domain learning," *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 10 459–10 465, 2020.

[18] Y. Zhu, J. Tremblay, S. Birchfield, and Y. Zhu, "Hierarchical planning for long-horizon manipulation with geometric and symbolic scene graphs," *ArXiv*, vol. abs/2012.07277, 2020.

[19] D. Driess, J.-S. Ha, and M. Toussaint, "Deep visual reasoning: Learning to predict action sequences for task and motion planning from an initial scene image," 2020.

[20] D. Driess, O. Oguz, J.-S. Ha, and M. Toussaint, "Deep visual heuristics: Learning feasibility of mixed-integer programs for manipulation planning," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 9563–9569.

[21] B. Kim and L. Shimanuki, "Learning value functions with relational state representations for guiding task-and-motion planning," in *Proceedings of the Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, L. P. Kaelbling, D. Kragic, and K. Sugiura, Eds., vol. 100. PMLR, 30 Oct–01 Nov 2020, pp. 955–968. [Online]. Available: https://proceedings.mlr.press/v100/kim20a.html

[22] T. Silver, R. Chitnis, A. Curtis, J. Tenenbaum, T. Lozano-Perez, and L. P. Kaelbling, "Planning with learned object importance in large problem instances using graph neural networks," 2020.

[23] D. Gnad, A. Torralba, M. Dominguez, C. Areces, and F. Bustos, "Learning how to ground a plan -partial grounding in classical planning," 01 2019.

[24] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, C. Gulcehre, F. Song, A. Ballard, J. Gilmer, G. Dahl, A. Vaswani, K. Allen, C. Nash, V. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu, "Relational inductive biases, deep learning, and graph networks," 2018.

[25] R. Bridson, "Fast poisson disk sampling in arbitrary dimensions," in *ACM SIGGRAPH 2007 Sketches*, ser. SIGGRAPH '07. New York, NY, USA: Association for Computing Machinery, 2007, p. 22–es. [Online]. Available: https://doi.org/10.1145/1278780.1278807