Learning to Guide Search in Long-Horizon Task and Motion Planning

Christopher Bradley and Nicholas Roy
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology United States
{cbrad, nickroy}@csail.mit.edu

Abstract: Recent work in Task and Motion Planning (TAMP) has enabled a new class of algorithms to better take advantage of off-the-shelf black-box samplers and solvers to find plans. However, many approaches to solving TAMP problems must balance the need to explore new regions of the search space with the computational cost of querying solvers for sub-problems that may be unlikely to succeed. In this work, we present a novel approach for solving TAMP problems, utilizing learned models trained from experience to inform when to attempt to solve potentially expensive sub-problems. We take advantage of highly optimized classical planners by learning representations that can be integrated with existing abstractions to guide search in long-horizon TAMP domains.

Keywords: Task and Motion Planning, Learning

1 Introduction

We aim to enable an autonomous agent to efficiently find low-cost plans for high-dimensional, long-horizon TAMP problems using learned models to guide planning. Consider, for example, a 20 degree-of-freedom (DOF) mobile robot attempting to 'cook' multiple objects in a kitchen environment. Solving this problem involves considering both the discrete sequence of actions (e.g., 'pick up object 'A', navigate within the room, then place 'A' on a new platform') and the constrained continuous and discrete parameters of those actions (a reachable grasp on A, a collision-free trajectory to move between configurations, a platform to place on). Problems of this nature can quickly become computationally challenging as task complexity increases. The more objects the robot is assigned to 'cook', the longer the task horizon grows, the more complex the discrete search becomes, and the more action parameters must be solved [1].

One common approach for solving TAMP tasks like the one described above involves using external samplers/solvers to find satisfying values for individual or sets of parameters (i.e. a grasp sampler for finding stable grasps or a motion planner for solving for trajectories between robot configurations) and then combining the solutions such that all constraints are satisfied. Often times these solvers may need to be queried several times before a solution is found. One major challenge is determining when to query a particular (potentially expensive) solver, and when to search for a more promising option. In [2], the authors propose an algorithm which intermittently searches through a discrete domain defined in Planning Domain Definition Language (PDDL), and queries pre-defined black-box solvers to bind new facts to the domain as needed.

Even with clever approaches such as [2] or [3] however, due to the combinatorial nature of task planning, coupled with the complexity of motion planning, problems can become computationally intractable very quickly as state and action spaces grow [1]. Particularly in problem settings where the chosen black-box samplers/solvers fail frequently (e.g., attempting to place an object on a cluttered platform), repeated failed queries to these solvers can cause planning times to balloon. Moreover, given the fact that simply finding a satisfying solution is so difficult, often times optimizing for metrics like execution time is an afterthought. Unfortunately, knowing ahead of time which subproblems have feasible (or optimal) solutions is as hard as the original planning problem itself. There is no practical way to avoid periodically querying for solutions to subproblems that do not actually have feasible solutions, but we would like to be able to identify ahead of time the likelihood of a solution existing, and subsequently minimize the likelihood of wasting computational effort.

To address these problems, we propose two contributions. The first is a general modification to sampling based strategies for TAMP. Specifically, for each external black-box solver (like a grasp sampler or IK solver) used by the planner, we train a model to predict outcomes (either success or failure) and costs (in both planning and execution time) for different inputs. These models can then be evaluated by a planner to help guide selection of which parameters to use as inputs when querying potentially expensive external solvers. We also propose a modified planning algorithm based on Monte-Carlo Tree-Search (MCTS) to better take advantage of our ability to estimate these outcomes and costs to accelerate planning. Our planner uses the learned model's estimates to guide search in an effort to minimize total planning and execution time. We implement this approach using the PDDLStream framework defined in [2], and demonstrate an improvement in both planning and execution time in a single, simulated environment over a heuristic driven baseline [3].

2 Background

The combined TAMP problem often jointly considers elements of high-level task planning [4, 5] and low-level motion planning [6] in an attempt to solve hybrid discrete/continuous, multi-modal planning problems [1]. Solutions for TAMP problems take the form of a sequence of parameterized actions $\pi = [a_1, a_2, ..., a_n]$ that define a plan, where assigned parameters satisfy each action's constraints [1]. One approach for representing a TAMP problem —which we use in this work— is with an extension of PDDL called PDDLStream [2]. PDDLStream represents the discrete search portion of a TAMP problem in PDDL, augmented by *conditional generators* called *streams*, which model black-box samplers/solvers.

A PDDLStream problem (P, A, S, O, I, G) is specified as sets of predicates P, actions A, streams S, objects O, an initial state I, and a goal state set G. The initial and goal states of a PDDLStream problem are sets of facts: instances of boolean functions called predicates $p(x) \in P$, which are parameterized by tuples of objects $x \in O$. For example, the fact that certifies if the robot is at a given pose is an instantiation of the predicate $AtPose\ ?p$, and is either true of false for different pose objects $?p$. Actions $a \in A$ are defined by two sets of predicates: pre and post conditions, and are parameterized by object tuples x. For a given input x, if the pre-conditions evaluate to true, the action is legal, and the post-conditions specify which facts will change value (either True or False) in the next state.

For certain actions, preconditions may include facts that are either cumbersome or impossible to add to P. For example, it is unclear how one would enumerate all possible $Pose$ objects, which are continuous, without creating a potentially intractably large problem. To account for this, PDDLStream problems include conditional generators called *streams* $s \in S$. Streams consist of: sets of (1) input and (2) output objects, (3) domain predicates which must be true in the input, (4) certified predicates which are deemed true if the queried stream is successful, and (5) an external function that is called when the stream is queried. When an action has a precondition which can only be certified by a stream, that stream can be queried in an attempt to determine if said precondition can be certified. Note that a stream may need to be queried many times before generating an output that jointly satisfies all constraints in a TAMP problem. We refer the reader to [2] for a more detailed description.

There are several TAMP solvers that have been developed over the past few years which use PDDLStream to define a TAMP problem. One general approach is to optimistically assume that any time a stream is needed to certify a fact, it can be queried successfully, then generate abstract plans π, which contain actions that have unbound parameters as a result of this assumption. Using the approach outlined in [2], for a given abstract action plan, we can generate the sequence of streams s

that must be queried to bind the unbound objects in π: the stream-plan ψ. If each $s \in \psi$ is able to be successfully queried for satisfying output objects, then we can return a fully bound action plan, and therefore solve the original TAMP problem. Unsurprisingly however, as the size of the planning domain increases, and as the planning horizon lengthens, PDDLStream problems can become exorbitantly computationally expensive.

3 Our Approach

Our objective is to find plans which solve TAMP problems efficiently, both in terms of wall-clock time spent searching for the plan, and with respect to the time it takes to execute the plan on a robot. To do this we propose an approach for learning to predict the outcome and cost of black-box samplers/solvers (from this point onward referred to as a *stream*) used by existing TAMP algorithms, then use these models to accelerate planning. We first present how we construct and train a simple model to predict the outcome and costs of individual streams. Next, we demonstrate how to estimate the expected cost of a stream plan using these predictions. Finally, we show how this cost estimate is incorporated into a novel planner to more efficiently guide search.

3.1 Learning to Model External Functions

We train separate models for each black-box sampler/solver, mapping an individual stream's inputs to estimates of the outcome and costs of querying and executing the given stream. Each model has two types of input: local information that is defined in the domain of the stream, and global information (e.g., the poses of various other objects in the scene). In our model, both types of information are given as input to a simple Multi-Layered Perceptron (MLP). We apply a fully connected network to produce four outputs: P_S; the probability the stream successfully finds a satisfying output, R_S; the expected cost of querying the stream successfully in wallclock-time, R_F; the expected cost of querying the stream unsuccessfully, and R_M; the expected time it would take to execute any trajectory produced by the stream (if a stream does not generate a trajectory, we set this value to 0).

With these four properties P_S, R_S, R_F, and R_M, we have the ability to compute the expected cost of querying, then executing any trajectory generated by a single stream s:

$$Q(s) = P_S(R_S + R_M) + (1 - P_S)R_F \tag{1}$$

Eq. 1 represents the intuition that we only accrue certain costs (like time to query a stream successfully or time to execute a trajectory) when a stream succeeds, and similarly only suffer the cost of querying a failed stream when it actually fails. The labels for our four outputs can be generated while running a baseline planner [3], simply by tracking the outcome and costs each time a stream is queried. In this way, we can generate training data by running any PDDLStream based planner.

3.2 Evaluating Stream Plans using Learned Models

Eq. 1 defines the expected cost of querying and executing any trajectory generated by a single stream. In order to evaluate the expected cost of a sequence of streams ψ, we must consider the subsequent costs both in the case when a stream succeeds (we attempt to query the next stream in ψ), and when it fails and we are forced to try again.

In [7] and [8], the authors derived from the Bellman equation a method to estimate the cost of a sequence of stochastic actions with binary outcomes. We make a small modification to incorporate the expected cost of execution R_M, and represent the expected cost of stream plan ψ, beginning at stream s_t as:

$$Q_\psi(s_t) = P_{S_t}(R_{S_t} + R_{M_t} + Q_\psi(s_{t+1})) + (1 - P_{S_t})(R_{F_t} + Q_\psi(s_t)), \tag{2}$$

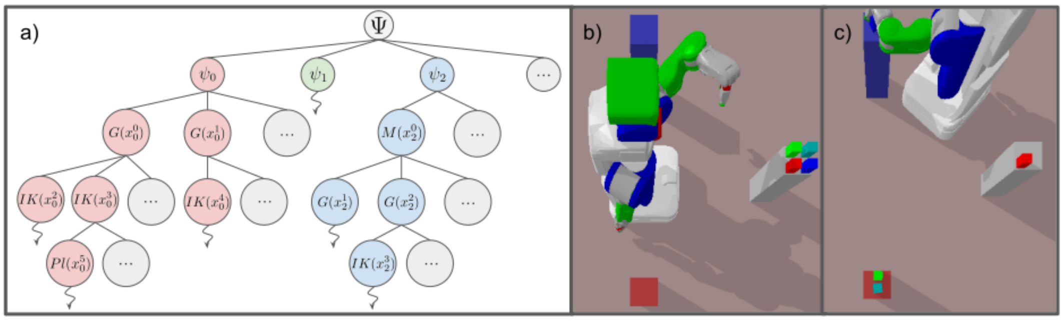

Figure 1: **A)** An illustration of our planning approach. The root node represents a set of stream plans Ψ, and each of its children are a sample ψ from that set. For all subsequent nodes, the available actions consist of querying one of four external samplers/solvers: G: sample grasp, Pl: sample stable placement, IK: solve for kinematically feasible configuration, and M: solve for collision free motion plan. Nodes corresponding to the same ψ share the same color. The subscript for parameters x also corresponds to the stream-plan, while the superscript denotes the node number within a given ψ. We are able to estimate the value of any branch in this tree with our learned models using eq. (4). **B)** An initial configuration from our simulated experiment. As in [3] the goal is for the robot to move each block first to the sink (blue platform), then the stove (red platform). **C)** Here we highlight a potential failure mode in this domain, if the third block is placed poorly on the stove, it may be impossible to safely place all four blocks there without risking collision. Our approach allows us to predict when a block placement will lead to failure later in a plan, allowing us to spend computation elsewhere.

where $Q_\psi(s_t)$ represents the total cost (in seconds) it would take to successfully query each stream $s_t \in \psi$ and execute any trajectories therein, beginning at stream s_t in the plan. Manipulating eq. 2 algebraically, we can re-write the expected cost as:

$$Q_\psi(s_t) = R_{S_t} + R_{M_t} + Q_\psi(s_{t+1}) + \frac{1 - P_{S_t}}{P_{S_t}} R_{F_t}. \tag{3}$$

Due to the nature of stochastic external solvers, any stream can be queried infinitely many times, and may eventually yield a successful output. Eq. 3 represents this intuition, as the expected cost for a particular step in the recursion is simply the cost of successfully querying a stream and executing any generated trajectory, plus the number of times we expect querying s_t to fail, times the expected cost of that failure. The ratio $\frac{1-P_{S_t}}{P_{S_t}}$ represents one less than the expected number of queries before success in the geometric distribution parameterized by P_{S_t}.

By rolling out the recursive steps, we can write the expected cost analytically:

$$Q_\psi(s_t) = \sum_{s_t \in \psi} \left(R_{S_t} + R_{M_t} + \frac{1 - P_{S_t}}{P_{S_t}} R_{F_t} \right). \tag{4}$$

As a result, we are able to estimate the expected cost of planning and execution from any point in a given ψ, simply by utilizing each remaining stream's four estimated values according to Eq. 4.

3.3 Planning with our Models

To take advantage of our ability to estimate the expected cost of finding the unbound parameters in a stream plan, we formulate the TAMP problem as a stochastic search problem. In [3], the authors propose using Progressive Widening for Upper Confidence Bounded Trees (PW-UCT) [9, 10] to search for parameters that satisfy the constraints of an action plan. We build upon this approach for our planning algorithm, and so outline it briefly here.

3.3.1 Searching for Abstract Stream Plans

The first step in our approach is to generate several abstract action plans $\pi \in \Pi$ — sequences of actions which would represent a successful plan if all preconditions are met — from a PDDLStream problem using an off-the-shelf top-k planner [11]. As mentioned in Sec. 2, some or all of an action's preconditions may be facts certifiable only by a stream. However, during the search for abstract action plans, we do not explicitly query those streams (execute the functions associated with them), as this would be prohibitively expensive. Instead we solve for abstract plans π, and compute the associated stream-plan ψ: the sequence of streams that must be successfully queried to solve for the unbound parameters in π. Once we have a set of k stream plans $\psi \in \Psi$, we can begin to attempt to search for the unbound parameters.

3.3.2 Searching within Stream Plans using PW-UCT

There are four distinct steps in a PW-UCT search problem. First is *selection*, where the existing tree is navigated according to the UCT heuristic (5) to find a node to expand. Next, in the *expansion* phase, a child from the selected node is generated and appended to the tree. Then, in the *simulation* step, we consider continually adding nodes from the newly expanded child node until either an expansion fails, or we successfully reach our goal. Finally, upon the conclusion of the roll-out, we update the statistics (total node visits and total accrued reward) of all visited nodes via *back-propagation*. This process is repeated until a solution is found.

Our tree is built as follows. At the root node (level 0), the available actions correspond to selecting one of the stream plans returned by the top-k planner. After this choice, each level-1 subtree is associated with different evaluations for that stream plan. Each level in a sub-tree corresponds to a specific stream and each node in a level to a different query to that stream in an attempt to bind parameters. Because streams can be queried infinitely many times and potentially produce novel results, there are infinitely many actions from each node (although a tree will only get as deep as the length of a stream plan). Refer to figure 1 for a depiction of a growing search tree.

During tree traversal from the root to a leaf in the selection phase of search, at each node, we must decide if we are going to query the stream for a new set of groundings for its output, or if we are going to select a child node with an existing assignment. In [3], this decision is governed by the progressive-widening inequality [10]: $N(v)^\alpha > (N(v)-1)^\alpha$, where $N(v)$ represents the number of visits to a node v, and α is an exploration constant. If the condition is true, then a new child node is created from node v by querying the associated stream. If not, and multiple children exist from a particular node, we consider the UCT equation (5) to select the next node:

$$argmax_{v_i} \frac{Q(v_i)}{N(v_i)} + c\sqrt{\frac{2ln(N(v))}{N(v_i)}}, \tag{5}$$

where v_i represents a child of node v, $N(v)$ denotes the number of times a node has been visited, and $Q(v_i)$ gives the estimated value of a particular child node. Once a node is selected for expansion, the associated stream is queried, and, if the query is successful, the parameters of that stream are bound to the output. From there, the simulation and back-propagation steps are taken, and selection begins again. By growing the tree in this way, we only evaluate streams that are determined by UCT to best balance exploiting high value branches, and exploring new bindings. Once a full sequence of actions has been successfully bound, we have solved the original TAMP problem, and can return the full plan.

3.3.3 Guiding Search

The UCT equation (5) relies on an estimate of Q in order to guide search. In [3], the authors propose a heuristic based on the depth of the search tree, and accrued reward. Such heuristics, while useful, require domain knowledge, may necessitate tuning, and cannot adapt to different streams within a plan. To more efficiently guide search, we use our learned models, applied recursively to each

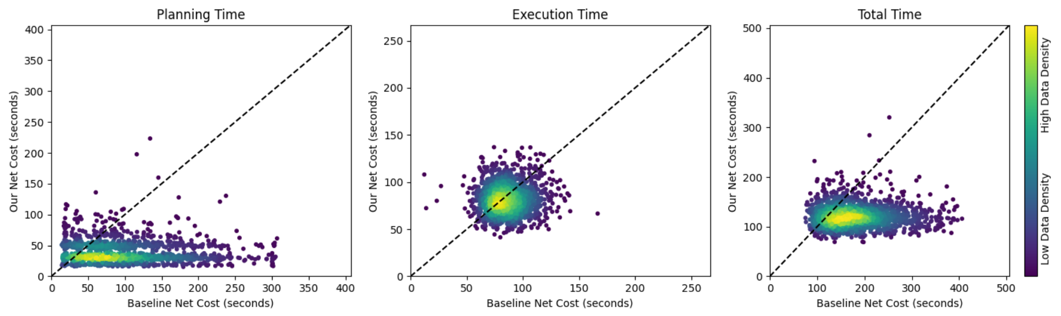

Figure 2: A comparison between the cost-to-go of baseline planner [3] and our Learned TAMP Planner for 1,600 simulated trials in the kitchen domain. Each point in the scatter-plot represents the outcome of a single trial, where density highlights the overlapping results. Our agent outperforms the baseline across all metrics, with a 45% improvement in expected total planning time.

component of the remaining stream plan, to estimate Q. Specifically, we evaluate eq. (4) for any node visited in search in order to get an estimate for remaining expected cost. If, in the application of eq. (4), we encounter an stream input that has not yet been bound to a real parameter, we pass the model a zero-vector of the same shape in its place to make a predictions without that information. For each node, we use the negative of the final output from (4) as an estimate for Q in (5). By using our trained model's predictions in this way, we are able to make decisions on which external functions to query using past experience, taking the into account all information that may influence the likelihood of success and costs of pursuing a particular branch in the search space.

We also propose one further alteration to PW-UCT, which involves taking advantage of our ability to estimate the value of querying a stream to avoid depending on the progressive-widening heuristic. Instead of relying on the inequality defined earlier as in [3], we simply consider the act of sampling a new node as another action in eq. (5). If the UCT heuristic for querying the current stream again —according to the estimated cost from (4)— is higher than that of any the available children, we do so, and add a new child to the current node. This allows us to completely forgo the need to rely on the hand-tuned progressive-widening heuristic, and guide our search purely through the predictions of our learned models. Because the exploration factor in (5) guarantees we will eventually re-sample every node, we retain the guarantee that we will eventually expand our search breadth with this method. In the next section, we demonstrate the benefits of using our approach in a simulated kitchen environment.

4 Experimental Results

To highlight the capabilities of our learned planner, we implement our approach in a simulated scenario. In an effort to make comparison straight-forward, we use an environment tested in [3]; specifically, their 'kitchen' domain. In this task, the robot is a simulated PR2[12] with five available actions: pick up an object, place an object on a platform, move between configurations, cook an object, and clean an object. Picking up an object requires sampling a grasp and computing a collision free robot configuration. Placing an object requires sampling a safe position on a platform and another collision free configuration. Moving the robot requires solving for a collision free trajectory between two robot configurations. Finally, an object is cleaned when placed on the sink and cooked when placed on the stove. The task specification is to first 'clean', then 'cook' all four blocks initially placed on the table platform. We once again refer the reader to [3] for a more detailed description of the problem domain.

In this domain, we ran over 1600 trials for both the uninformed baseline planner[3] and our learned planner (trained on 100 trials worth of data from the baseline planner), initializing each trial with new random seed. Over those trials, we recorded both the planning and execution time, and plotted these values for the individual trials (along with their sum) in fig. 2. As shown in the plots, we demonstrate a mean improvement in total time of approximately 63 seconds. Considering the per trial improvement, nearly 59 seconds of that decrease comes from the planning cost, and around 4 seconds of improvement from execution cost. This notable difference —an approximate 45% reduction in expected total time per trial— indicates that our solution can find plans of similar execution time to a baseline planner but with significantly less planning time. This signals the importance of carefully reasoning about which streams are actually evaluated in TAMP solvers, and is an highlights the potential of our approach.

5 Related Works

5.1 Task and Motion Planning

Viewing TAMP as a hybrid constraint satisfaction problem [1], most TAMP approaches can be categorized as solving the constraints jointly or individually. In the first approach, the problem is written as one large constrained optimization problem (typically a Mixed Integer Program), where discrete components such as which block to pick up are represented by integers, and the trajectory optimization is real-valued [1, 13, 14]. These joint optimization strategies can be appealing, as satisfying a single optimization solves the entire problem. However, such approaches are often limited in that certain aspects of the problem may not be easily differentiable, efficiently re-using computation can be difficult, and it might not always be straight-forward to incorporate off-the-shelf samplers/solvers [1]. The primary alternative approach is to consider solving for sets of parameters that satisfy small groups of constraints, and combine the solutions into actions and plans. For example, we can sample a block placement on a platform that is free of collisions, then confirm it is positioned so that there exists a kinematically feasible configuration for the robot to execute such a placement. Approaches that break up the problem in this way can take advantage of external samplers and solvers that are optimized for specific sub-problems (e.g, Fast Downward for the discrete task problem, efficient inverse-kinematics solvers, or neural networks for grasp sampling) [1, 2, 3]. In this work, we build upon this second approach.

5.2 Planning with Learned Models

There has been significant recent progress in improving planning for TAMP problems using learned models. Most relevant to this work are those contributions which attempt to accelerate search from experience [15, 16, 17, 18, 19, 20, 21, 22]. [18] and [19] learn explicitly which components of a given domain (objects and actions respectively) are relevant for a particular TAMP problem specification, though do not further guide search within their reduced domain. [20] learns an action sampling distribution for geometric motion planning problems, but does not take advantage of off-the-shelf samplers/solvers. [17] learns a Q-function as a heuristic to use in search for a geometric TAMP problems, however does not learn to bind the continuous parameters of its actions. Closely related, [22] specifically scores the relevance of streams within the PDDLStream [2] framework, though does not consider the search for an action's parameters. Finally, [16] attempts to learn a feasibility predictor from images to accelerate the work in [3]. However, they do not predict costs, and simply threshold their predicted feasibility to bound branches in search.

Outside of TAMP, there has been further work in planning with learned outcomes. Specifically, [7, 8] learn models to predict the success/failure and costs of actions for planning in long-horizon, partially observable domains, however are limited to navigation tasks. We build upon the intuition of this work in sec. 3.2. In [23], the authors propose simultaneously learning outcome prediction and action dynamics, and plan purely in a learned latent space. Due to the nature of their learned representation however, they cannot utilize optimized off-the-shelf planners to solve TAMP problems.

6 Conclusion

In this work, we propose a novel approach for planning in long-horizon TAMP problems. Our contributions include a method for learning to predict the outcome and associated costs of querying potentially expensive black-box samplers/solvers, and using the predictions to guide search in our planner. We demonstrate an approximately 45% reduction in planning and execution time compared to an uninformed baseline [3].

In future work, we plan to expand our experimental results from a single simulated domain to both more simulated environments, as well as to a real-world robot. We also hope to augment our models to, in addition to learning to predict the estimates of outcome and costs, also predict a latent representation of the objects produced by querying a stream (e.g., a learned representation of a pose object). This representation could then be used as input to subsequent streams —replacing any zero-vector inputs as demonstrated in [22]— and enable us to better model expected costs for longer-horizon tasks. Perhaps such a model would enable greater performance in improving execution time.

This work demonstrates that we can vastly decrease the costs of planning and execution in TAMP problem by learning where to best spend computational resources. Our learned representation is able augment the hand-written abstraction of a PDDLStream problem, allowing us to take advantage of classical search techniques like [11] along with optimized trajectory planners, while still learning from past experience to improve overall efficiency.

References

[1] C. R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, and T. Lozano-Pérez. Integrated task and motion planning. *Annual review of control, robotics, and autonomous systems*, 4:265–293, 2021.

[2] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling. Pddlstream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, pages 440–448, 2020.

[3] T. Ren, G. Chalvatzaki, and J. Peters. Extended tree search for robot task and motion planning. *arXiv preprint arXiv:2103.05456*, 2021.

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

[5] E. Karpas and D. Magazzeni. Automated planning for robotics. *Annual Review of Control, Robotics, and Autonomous Systems*, 3:417–439, 2020.

[6] S. M. LaValle. *Planning algorithms*. Cambridge university press, 2006.

[7] G. J. Stein, C. Bradley, and N. Roy. Learning over subgoals for efficient navigation of structured, unknown environments. In *Conference on robot learning*, pages 213–222. PMLR, 2018.

[8] C. Bradley, A. Pacheck, G. J. Stein, S. Castro, H. Kress-Gazit, and N. Roy. Learning and planning for temporally extended tasks in unknown environments. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4830–4836. IEEE, 2021.

[9] G. M. J. Chaslot, M. H. Winands, H. J. v. d. Herik, J. W. Uiterwijk, and B. Bouzy. Progressive strategies for monte-carlo tree search. *New Mathematics and Natural Computation*, 4(03): 343–357, 2008.

[10] R. Coulom. Computing "elo ratings" of move patterns in the game of go. *ICGA journal*, 30 (4):198–208, 2007.

[11] D. Speck, R. Mattmüller, and B. Nebel. Symbolic top-k planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 9967–9974, 2020.

[12] Pr2 - robots: Your guide to the world of robotics. `https://robots.ieee.org/robots/pr2/`. (Accessed on 11/01/2022).

[13] M. A. Toussaint, K. R. Allen, K. A. Smith, and J. B. Tenenbaum. Differentiable physics and stable modes for tool-use and manipulation planning. 2018.

[14] E. Fernandez-Gonzalez, E. Karpas, and B. Williams. Mixed discrete-continuous planning with convex optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

[15] D. Driess, J.-S. Ha, R. Tedrake, and M. Toussaint. Learning geometric reasoning and control for long-horizon tasks from visual input. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 14298–14305. IEEE, 2021.

[16] L. Xu, T. Ren, G. Chalvatzaki, and J. Peters. Accelerating integrated task and motion planning with neural feasibility checking. *arXiv preprint arXiv:2203.10568*, 2022.

[17] B. Kim and L. Shimanuki. Learning value functions with relational state representations for guiding task-and-motion planning. In *Conference on Robot Learning*, pages 955–968. PMLR, 2020.

[18] T. Silver, R. Chitnis, A. Curtis, J. B. Tenenbaum, T. Lozano-Perez, and L. P. Kaelbling. Planning with learned object importance in large problem instances using graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 11962–11971, 2021.

[19] C. Agia, K. M. Jatavallabhula, M. Khodeir, O. Miksik, V. Vineet, M. Mukadam, L. Paull, and F. Shkurti. Taskography: Evaluating robot task planning over large 3d scene graphs. In *Conference on Robot Learning*, pages 46–58. PMLR, 2022.

[20] B. Kim, L. Kaelbling, and T. Lozano-Pérez. Guiding search in continuous state-action spaces by learning an action sampler from off-target search experience. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[21] T. Pan, A. M. Wells, R. Shome, and L. E. Kavraki. Failure is an option: Task and motion planning with failing executions. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 1947–1953. IEEE, 2022.

[22] M. Khodeir, B. Agro, and F. Shkurti. Learning to search in task and motion planning with streams. *arXiv preprint arXiv:2111.13144*, 2021.

[23] D. Xu, A. Mandlekar, R. Martín-Martín, Y. Zhu, S. Savarese, and L. Fei-Fei. Deep affordance foresight: Planning through what can be done in the future. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6206–6213. IEEE, 2021.