Journal of Artificial Intelligence Research 69 (2020) 471-500 Submitted 05/2020; published 10/2020

The PETLON Algorithm to Plan Efficiently for Task-Level-Optimal Navigation

Shih-Yun Lo YUNL@CS.UTEXAS.EDU
Department of Mechanical Engineering,
The University of Texas at Austin

Shiqi Zhang ZHANGS@BINGHAMTON.EDU
Department of Computer Science,
The State University of New York at Binghamton

Peter Stone PSTONE@CS.UTEXAS.EDU
Department of Computer Science,
The University of Texas at Austin and Sony AI

Abstract

Intelligent mobile robots have recently become able to operate autonomously in large-scale indoor environments for extended periods of time. In this process, mobile robots need the capabilities of both task and motion planning. *Task planning* in such environments involves sequencing the robot's high-level goals and subgoals, and typically requires reasoning about the locations of people, rooms, and objects in the environment, and their interactions to achieve a goal. One of the prerequisites for *optimal* task planning that is often overlooked is having an accurate estimate of the actual distance (or time) a robot needs to navigate from one location to another. State-of-the-art *motion planning* algorithms, though often computationally complex, are designed exactly for this purpose of finding routes through constrained spaces.

In this article, we focus on integrating task and motion planning (TMP) to achieve task-level-optimal planning for robot navigation while maintaining manageable computational efficiency. To this end, we introduce TMP algorithm PETLON (Planning Efficiently for Task-Level-Optimal Navigation), including two configurations with different trade-offs over computational expenses between task and motion planning, for everyday service tasks using a mobile robot. Experiments have been conducted both in simulation and on a mobile robot using object delivery tasks in an indoor office environment. The key observation from the results is that PETLON is more efficient than a baseline approach that pre-computes motion costs of all possible navigation actions, while still producing plans that are optimal at the task level. We provide results with two different task planning paradigms in the implementation of PETLON, and offer TMP practitioners guidelines for the selection of task planners from an engineering perspective.

1. Introduction

"Planning," or selecting a sequence of actions to achieve a goal, has been a core focus of interest within the field of Artificial Intelligence (AI) since the field was founded in the 1950's. Initially, the focus of attention was on *task planning* which is concerned with sequencing actions within a symbolic representation of the state space (Fikes & Nilsson, 1971). For example, *if a robot has the goal of obtaining supplies for camping including milk and frozen hot dogs, both of which could spoil if not refrigerated, a symbolic (task) planner, given a domain model that includes coolers, ice,*

and conditions for spoiling, could identify that the robot should first buy a cooler, then a block of ice, and then milk and hot dogs. In general, task planners aim to find the shortest plan in terms of number of symbolic actions. If action costs are available, some task planners are also able to identify the lowest cost plan, which in general may be longer in terms of the number of actions.

A key limitation of task planning is that it assumes that symbolic actions can be executed "atomically." Continuing our camping example, it does not reason about how the robot should traverse continuous space in order to travel from its current location to the store. Rather it assumes that the robot can teleport itself to the next location, perhaps roughly estimating how long it would take to move there in the real world. In contrast, a largely independent thread of research exists on *motion planning* that focuses on producing a continuous motion plan while avoiding collisions with obstacles in 2D or 3D continuous space (Latombe, 2012). Traditionally, motion planning has been concerned with computing a path connecting a start configuration to a goal configuration, without any concern for the sequencing of subgoals. Within the context of mobile robotics, the robots need capabilities of both task and motion planning, so as to sequence actions to accomplish complex tasks, while producing these actions' implementations at the motion level.

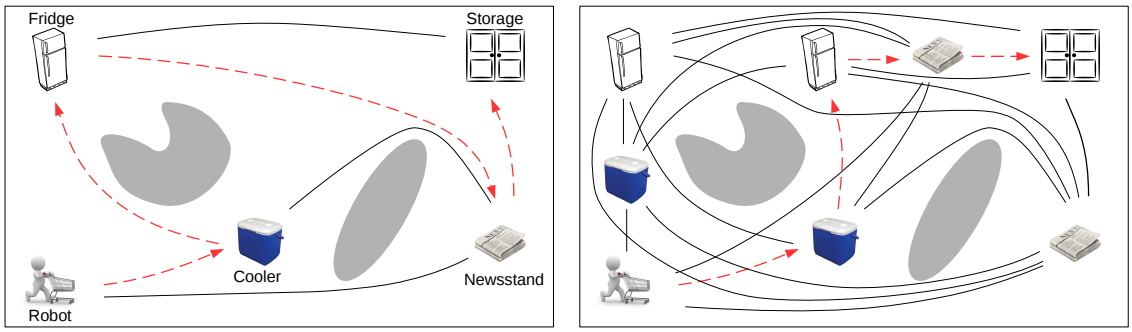
1.1 Two Types of TMP Problems

Task planning and motion planning have historically remained mostly (though not entirely — see related work) independent, because physical robots have only been able to execute very short missions that could be solved entirely with motion planning algorithms. TMP for manipulation (TMP-M) has attracted much attention of late, mainly to ensure the geometric feasibility of symbolic plans in highly confined workspaces, with complex kinematic constraints (Gravot et al., 2005; Erdem et al., 2011; Lagriffoul et al., 2014; Srivastava et al., 2014; Garrett et al., 2018a). Despite the great success of these approaches, **TMP for navigation** (TMP-N), i.e., to select task routes considering task-level domain knowledge and navigation costs, presents sufficiently different challenges that different approaches are needed, and has not yet been well addressed in the literature.

1.2 Challenges of TMP-N

TMP-N frequently arises in large, knowledge-intensive domains, in which a robot has to reason about many objects and their properties, such as feasible locations to acquire and then to store milk. Moreover, solutions to TMP-N may vary significantly in quality, where suboptimal plans may significantly delay task completion schedule, due to long execution time navigating in large environments. As a result, the following two properties are generally necessary to TMP-N algorithms: *scalability* in knowledge representation and reasoning (KRR), and *optimality* in planning actions at the task level.

Because of the recent advances in sustainability of long-term autonomy on mobile service robots in large-scale environments (Khandelwal et al., 2017; Hawes et al., 2017; Veloso, 2018), there is a pressing need to generate task plans that are fully aware of—and indeed dependent upon—the grounded navigation costs of task actions that can only be determined by motion planning algorithms. With well-defined physical constraints of symbolic states and a model of the free space and obstacles in the environment, the navigation costs of all possible task actions can be evaluated and then used to select the optimal task route. However, in cases with combinatorially many possible task sequences, doing so can be computationally infeasible. The following example demonstrates the infeasibility.



(a) Small number of objects.

(b) Large number of objects (doubled)

Figure 1: Camping Preparation example: the robot wants to compute a plan to prepare items for camping, while minimizing overall plan cost. As the number of objects increases, the number of motion cost evaluations increases exponentially.

1.3 A Motivating Example of TMP-N Algorithms: Camping Preparation

Figure 1 illustrates a scenario where a robot needs to prepare items for camping:

> *A hot dog and a newspaper need to be collected and moved to the storage room; and the robot has to have a cooler as the container for hot dogs.*

In the first setting (left subfigure), to find the optimal task plan, the robot has to evaluate 7 motion costs. In the second setting (right subfigure), the number of objects that the robot has to reason about is doubled. As a result, the number of cost evaluations grows to 20, even after the task planner rules out plans that do not meet action preconditions (the robot has to collect a cooler before collecting a hot dog from a fridge). The time complexity of evaluating all navigation actions' motion costs is $O(C(N,2)) = \Theta(N^2)$, where N is the number of objects, and $C(N,2)$ is a N-choose-2 combination operation. In other words, continuing to scale up the number of objects, we can see the number of motion cost evaluations soon becomes prohibitively large.

The aim of this research is to integrate task and motion planning to select the optimal (lowest-cost) task route for robot navigation in a computationally efficient manner. We introduce a novel algorithm, called Planning Efficiently for Task-Level-Optimal Navigation (PETLON), that returns task-level-optimal solutions while significantly reducing the number of motion cost evaluations. We say a task plan, in the form of a sequence of symbolic actions, is task-level-optimal, if its overall action cost is not higher than that of any task plan, where action costs of the task plans are evaluated using a motion planner. PETLON is a general approach that can work with a variety of task and motion planners. In this article, we instantiate PETLON with a probabilistic motion planning algorithm, and with two different task planning paradigms that are based on the Planning Domain Definition Language (PDDL) (McDermott et al., 1998) and Answer Set Programming (ASP) (Lifschitz, 2008; Gelfond & Kahl, 2014) respectively. We evaluate PETLON with mobile robot delivery tasks both in simulation and on a real robot. From the experimental results, we observed that PETLON, while maintaining task-level optimality, significantly improves planning efficiency in most cases, compared with baselines including a competitive, task-level-optimal approach.

The main contribution of this work (i.e., the PETLON algorithm) was previously published as a conference paper authored by the same group of researchers (Lo et al., 2018). In this article, we dis-

cuss related work more comprehensively, include sufficient technical details to ensure the reported results can be reproduced by other researchers, provide a pictorial example for demonstration in detail, conduct more experiments (specifically the "visibility" experiments in Section 6), and point out limitations of our approach while identifying research directions for future work.

The remainder of this article is organized as follows. Section 2 discusses existing TMP algorithms, including their assumptions and application domains. Section 3 formally defines the problem of task-level-optimal TMP for navigation. Section 4 is the main section of this article, and includes our PETLON algorithm and a proof of its task-level optimality. Section 5 presents the technical details of instantiating PETLON on a mobile robot platform, including the selections of task planning and motion planning algorithms. Section 6 details the experiment setup, a list of hypotheses to be evaluated in experiments, and results both in simulation and on a real robot. Finally, we conclude in Section 7, followed by a discussion about the limitations of our approach and a few directions for future work.

2. Related Work

The integration of task and motion planning has a long history. Shakey, the first mobile robot with the ability to perceive and reason about its surroundings (Nilsson, 1984), executed its plans in the real world, and is the earliest example. However, it is not until recently that *task and motion planning* (TMP) has been used as a term to refer to a family of algorithms that *interleave* the processes of task and motion planning. These algorithms have been developed under very different assumptions with very different goals, making direct comparisons a challenge. Here we summarize representative algorithms for this problem.

Hierarchical planning methods aim at computing plans at different abstraction levels (Sacerdoti, 1974), where primitive actions at the bottom level are directly executable. This hierarchical planning idea has been widely applied to sequential decision-making methods, such as classical planning (Nau et al., 2003; Zhang et al., 2015), and stochastic planning (Pineau et al., 2003; Zhang et al., 2013). In particular, when world models are not available, one can learn from reinforcement at different abstraction levels, for example using options for reinforcement learning (Sutton et al., 1999). Fundamentally, TMP algorithms could be considered as a two-layer hierarchical planning approach, where the agent may or may not learn at either level. The uniqueness of TMP within the hierarchical planning context lies in the top and bottom layers corresponding to symbolic (discrete) and geometric (continuous) spaces respectively.

aSyMov is one of the earliest TMP algorithms, which was previously referred to as a *symbolic and motion planning* algorithm (Cambon et al., 2009). The gap between task and motion planners was bridged via a set of predefined *roadmaps* in aSyMov, where a roadmap node represents a configuration and an arc represents a collision-free motion. The search was conducted at the motion level within roadmaps using a motion planning system called Move3D (Simeon et al., 2001) and across roadmaps using a task planning system called Metric-FF (Hoffmann, 2003). Another early TMP algorithm is the so-called *semantic attachment* approach that calls external functions to evaluate grounded predicates and fluent changes. Semantic attachments have been integrated into PDDL planners through adding external "modules." The main difference between PETLON and the above-mentioned methods is that PETLON guarantees task-level optimality.

Hierarchical task networks (HTNs) (Nau et al., 2003), as a framework for task planning, have been integrated with motion planners. One resulting algorithm is SAHTN (Wolfe et al., 2010). To

improve the HTN-level search efficiency, SAHTN uses an *irrelevance* function to rule out irrelevant domain variables (e.g., objects not blocking the way are not modeled in navigation actions). Although SAHTN is a "hierarchically optimal" algorithm, i.e., its optimality is conditioned on the hierarchy, it requires an irrelevance function that is frequently unavailable in practice. Hierarchical task and motion Planning in the Now (HPN) uses depth-first traversal and interleaves planning and execution (Kaelbling & Lozano-Pérez, 2011). HPN's hierarchical representation is similar to ABSTRIPS (Sacerdoti, 1974). More recently, HPN has been extended to model the uncertainty in action outcomes and observability (Kaelbling & Lozano-Pérez, 2013). SAHTN and HPN aggressively trade optimality for efficiency. As a result, they can solve extremely long-horizon planning problems, though plan quality may be sacrificed.

Another realization of TMP was achieved via introducing symbolic state constraints at the task level (Erdem et al., 2011), where new constraints are added into the task planner when no feasible kinematic solution can be found for the plan generated by the current task planner. Task-level optimal solutions can be found in that work, *only if* costs of all actions are evaluated at the motion level, which can be prohibitively time-consuming in practice, whereas PETLON is specifically designed to improve the overall efficiency by avoiding evaluations of all actions' costs. In the work of Dantam, Kingston, Chaudhuri, and Kavraki (2018), this idea was formalized to achieve probabilistically complete TMP, leveraging the incremental solution capability of Satisfiability Modulo Theories (SMT) (De Moura & Bjørner, 2011). The work of Wang, Dantam, Chaudhuri, and Kavraki focused on environments that include uncontrollable agents (such as humans), and produced a policy synthesis approach for TMP problems (Wang et al., 2016).

Off-the-shelf task and motion planners can be integrated using a planner-independent interface (Srivastava et al., 2014). Their task planner is optimistic at the very beginning and generates very short plans that are frequently infeasible at the motion level. To update the task planner, their interface needs to *explain* the motion-level failures to the task planners. While their approach is applicable to our navigation domains, failure diagnosis is generally a difficult reasoning problem, especially when a relatively large number of objects and their properties need to be considered.

Learning algorithms have been incorporated to guide the search in task and motion planning problems. For instance, the framework produced by Srivastava et al. was improved by incorporating reinforcement learning (RL) for plan refinement and learning from expert demonstrations to guide the search at the task level (Chitnis et al., 2016). Recent work aims at predicting *solution constraints* to reduce the search space in task and motion planning problems (Kim et al., 2017). In that work, a new representation was developed to facilitate the transfer of knowledge (in the form of solution constraints) from one problem instance to another to significantly reduce the search space.

Recently, an algorithm called FFRob has been developed for task and motion planning (Garrett et al., 2018a). FFRob does not require a motion planner, but directly conducts task planning over a set of samples generated in the configuration space. In order to efficiently search in this large sample space, FFRob extends the FastForwad heuristic (Hoffmann & Nebel, 2001) to generate a set of heuristics (to reason with geometric constraints) to guide the search. FFRob aims at plans of the shortest length (or "mode-sequence" length in their terms), which is different from our focus. This idea has been further extended to plan in factored transition systems for exposing the topology of their solution space (Garrett et al., 2017a), and to reason with possibly infinite sequences of object poses and static predicates (Garrett et al., 2017b). Separately, a generative adversarial network (GAN) (Goodfellow et al., 2014) was used to learn an action sampler (Kim et al., 2018) for TMP problems, which has been shown to be more effective than a heuristic-based task planner.

Instead of aiming at a general planning framework, researchers have developed algorithms that focus on individual aspects of the TMP problems. Lagriffoul et al. developed algorithms for rejecting inconsistent task actions for kinematically restricted problems (Lagriffoul et al., 2014), where an intermediate layer of constraints is generated and maintained by both task and motion planners. Toussaint developed a layered approach for TMP problems where the goal is specified with a cost function (e.g., to construct a stack of objects while maximizing the height of the physically stable construction), instead of a symbolic goal description (Toussaint, 2015). Focusing on efficient exploration, a selective sampling approach was developed to compute collision-free trajectories that satisfy high-level specifications (Plaku & Hager, 2010). These algorithms focus on one or more aspects of the TMP problem, and can be used to improve the performance of general TMP algorithms.

There has recently been an effort of developing a set of platform-independent benchmark problems for comparing TMP algorithms (Lagriffoul et al., 2018). Each benchmark problem includes a task specification, a motion specification, and a specification of task-motion interaction. However, the evaluations of the current benchmarks do not include optimality, and none of the current benchmarks require long-distance navigation actions. In comparison, PETLON aims at enabling mobile robots to plan efficiently to accomplish tasks that frequently require multiple navigation actions, while minimizing overall action costs.

Focusing on manipulation tasks, e.g., pick-and-place and box-pushing, researchers have developed asymptotically optimal algorithms with piecewise-analytic constraints (Vega-Brown & Roy, 2016; Schmitt et al., 2017). The algorithms went beyond planning in continuous geometric spaces through discretizing configuration spaces of grasping and ungrasping. Similarly to many TMP methods, the discretization significantly improves the efficiency of planning for solving complex tasks (manipulation or not). Despite the fact that these methods focus on manipulation tasks, the idea of "factoring" configuration spaces (Vega-Brown & Roy, 2016), which was further studied in constraint spaces (Garrett et al., 2018b) can potentially be employed to enhance PETLON to make it a globally optimal algorithm.

Although the algorithms discussed in this section are built on very different assumptions, two observations are shared across all of them: I) although some involve navigation actions, *manipulation* has been the main challenge at the motion level, e.g., grasping and ungrasping; and II) none of them guarantees task-level optimality.[1] PETLON is an algorithm that is guaranteed to produce task-level-optimal TMP solutions without requiring the evaluations of all actions' costs. As a result, PETLON is applicable to large-scale robot *navigation* domains that include both low-level (navigational) constraints and high-level (task ordering) constraints.

3. Problem Statement

Within the context of task and motion planning (TMP) problems, existing research has been conducted with very different assumptions and goals (see Section 2). In this section, we formalize TMP for navigation (TMP-N) problems at task and motion levels.

1. The work of Erdem et al. (2011) is an exception, where task-level optimality is achieved in a computationally expensive way. Methods that pre-evaluate all action costs, including that of Erdem et al. (2011), are referred to as "Brute force" in our experiments. PETLON is a more efficient algorithm that retains the guarantee of task-level optimality.

3.1 Planning at Task and Motion Levels

Task Level: Let $\mathcal{D}^t$ specify a task planning domain that includes a set of states, S, and a set of actions, A. We assume a factored state space such that each state $s \in S$ is defined by the values of a fixed set of variables. Each action $a \in A$ is defined by its preconditions and effects. A cost function *Cost* maps the state transition to a real number: $Cost(\langle s,a,s'\rangle) \rightarrow \mathbb{R}$, which represents the cost of action a being executed in state s.

Given domain $\mathcal{D}^t$, a task planning problem is defined by an initial state $s^{init} \in S$ and a specification of the goal that corresponds to a set of goal states $S^G \subseteq S$. A plan, $p \in P$, includes a sequence of transitions that can be represented as: $p = \langle s_0, a_0, \cdots, s_{N-1}, a_{N-1}, s_N\rangle$, where $s_0 = s^{init}, s_N \in S^G$ and P is the set of satisfactory plans.

Solving a task planning problem, using optimal task planner $\mathcal{P}^t$, produces plan p^* that is optimal among all satisfiable plans $p \in P$:

$$p^* = \underset{p \in P}{\operatorname{argmin}} \sum_{\langle s,a,s'\rangle \in p} Cost(\langle s,a,s'\rangle). \tag{1}$$

Motion Level: Let $\mathcal{D}^m$ specify a motion planning domain, where we directly search in the 2D workspace, since in this work we focus on only 2D navigation problems for motion planning. The 2D space is represented as a region in Cartesian space such that the position and orientation of the robot can be uniquely represented as a *pose* (x, y, θ), to define the physical state $x \in X$ of the robot. X is the physical robot state space.

Given $\mathcal{D}^m$ and a robot model $\mathcal{M}$, which includes the physical constraints of the robot, a motion planning problem can be specified by an initial pose x^{init} and a goal set X^{goal}. Some parts of the space are designated as free space, and the rest is designated as an obstacle. $\mathcal{M}$ specifies the kinematic constraints of the robot to plan while checking for collisions with the environment. $\mathcal{M}$ also specifies the dynamical constraints, including actuation limits, response time, and non-holonomic constraints of the platform, to forward simulate the dynamics of the robot $x \rightarrow x'$.

The motion planning problem is solved by the motion planner $\mathcal{P}^m$ to compute a collision-free trajectory ξ^* (connecting x^{init} to a pose $x^{goal} \in X^{goal}$ taking into account any physical constraints specified in $\mathcal{M}$) with minimal trajectory length, or execution cost, e.g. time duration $Len(\xi) = L$. We use Ξ to represent the trajectory set that includes all collision-free trajectories. The *optimal* trajectory is

$$\xi^* = \underset{\xi \in \Xi}{\operatorname{argmin}}\ Len(\xi), \tag{2}$$

where $\xi(0) = x_{init}$ and $\xi(L) = x_{goal} \in X_{goal}$.

Task and Motion Planning: A symbolic state s in $\mathcal{D}^t$ corresponds to a geometric constraint in $\mathcal{D}^m$ that can be represented as a set of poses X in the configuration space. For instance, the symbol "beside a table" corresponds to a (infinite) set of positions within a small range of the table. The geometric constraints ensure the motion-level feasibility between task state transitions.

We use a *state mapping function*, f: $S \rightarrow X$, to map the symbolic state $s \in S$ into a set of feasible poses X in continuous space, for the algorithm to sample from. We assume the availability of at least one pose $x \in X$ in each state s, such that the robot is in the free space of $\mathcal{D}^m$. If it is not the case, the state s is declared *infeasible*. The state mapping function has been referred to in different ways in the

literature, e.g., it has been referred to as a scene graph refinement function in the constraint-based TMP literature (Dantam et al., 2018).

It should be noted that such state mapping functions break global optimality. We would like to be able to guarantee full motion-level optimality. But in continuous domains, such a guarantee is elusive due to the fundamental difference between representations at the two levels (Konidaris et al., 2018). In line with past research (Cambon et al., 2009; Erdem et al., 2011; Srivastava et al., 2014), we use a state mapping function, which makes it possible for us to achieve task-level optimality, i.e., optimality conditioned on the motion planner and state mapping function, which is formally defined in the next subsection.

3.2 Definition of Task-Level-Optimal TMP-N

Building on the definitions of task and motion planning problems in the previous subsection, we formulate TMP-N problems, and define the task-level optimality of algorithms for TMP-N problems. The input of a TMP-N problem is a six-tuple

$$\Omega : \langle \mathcal{D}^t, \mathcal{D}^m, s^{init}, S^G, x^{init}, f, \mathcal{M} \rangle$$

where $x^{init} \in f(s^{init})$, meaning that the geometric initial position is consistent with the symbolic initial state.

A satisfactory output of a TMP-N problem is a two-tuple,

$$\langle p, [\xi_0, \xi_1, \cdots, \xi_{N-1}] \rangle$$

that includes a symbolic plan and a set of collision-free trajectories, where $p(0) = s^{init}$, $p(N) \in S^G$, $|p| = N$, $\xi_0(0) = x^{init}$, $\xi_i(0) \in f(s_i)$, and $\xi_i(T_i) \in f(s_{i+1})$ for $i = \{0, 1, \ldots, N-1\}$. T_i is the end time of trajectory ξ_i.

Finally, we define a task-level optimal plan to be a lowest-cost plan p^*, conditioned on a motion planner $\mathcal{P}^m$ and state-mapping function f:

$$p^* = \underset{p \in P}{\operatorname{argmin}} \Big(\sum_{0 \le i < |p|} Len(\xi_i) | \mathcal{P}^m, f \Big), \tag{3}$$

where $\xi_i = \mathcal{P}^m(\langle s_{i-1}, a_{i-1}, s_i \rangle, f, \mathcal{D}^m, \mathcal{M})$ is the trajectory returned by $\mathcal{P}^m$ given state transition $\langle s_{i-1}, a_{i-1}, s_i \rangle \in p$. In comparison to Equation 1, the added challenge here is that the motion cost function (here *Len*) is initially unknown and can only be estimated by the motion planner $\mathcal{P}^m$ with added computational expense.

Constrained State Mapping Function: The output of state mapping function f is defined as a set of feasible poses in continuous space in Section 3.2. This definition offers more flexibility in implementing f in domains with dynamic obstacles, but breaks the independence of cost evaluations of actions. In order to analyze the optimality of PETLON, we consider a constrained form of f in the following discussion, where the output is a single pose in continuous space.

Next, we present our task-level-optimal algorithm for TMP-N problems, and its two variations that produce different trade-offs over computational expenses between task and motion planning.

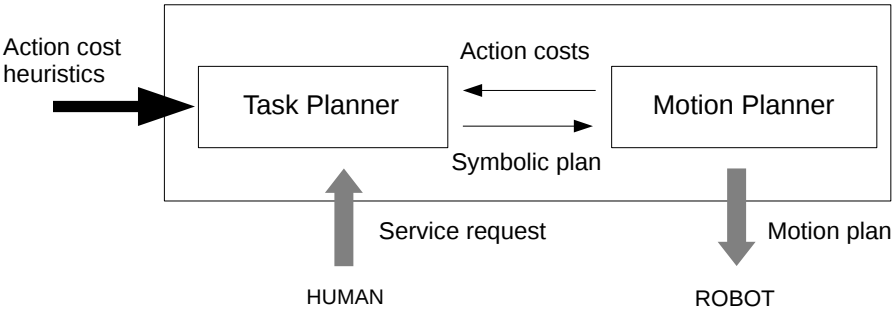


Figure 2: Overview of PETLON to efficiently solve TMP-N problems, with guaranteed task-level optimality.

4. Algorithm

PETLON (Planning Efficiently for Task-Level-Optimal Navigation), is visualized in Figure 2, where task planner $\mathcal{P}^t$ and motion planner $\mathcal{P}^m$ serve as the two main components. The task planner interacts with humans by taking their service requests and generates symbolic plans. The motion planner generates realistic motion costs of each symbolic action. Our guarantee of task-level optimality relies on an admissible heuristic for motion costs, which can be easily obtained in practice, e.g., through straight-line distance in 2D space.

Before introducing the algorithm, it is necessary to first define three functions for evaluating the costs of navigation actions:

- *Heuristic cost function* (h): we use h to represent our admissible heuristic cost function that computes the *Euclidean* distance between x and x' in 2D space.

$$h(x,x') = ||x - x'||_2 \tag{4}$$

- *Evaluated cost function* ($\hat{C}$): function $\hat{C}$ calls our motion planner $\mathcal{P}^m$ to estimate the geometric-level cost of traversing from $x \in f(s)$ to $x' \in f(s')$ given robot workspace $\mathcal{D}^m$:

$$\hat{C}(x,x') = Len\big(\mathcal{P}^m(\langle s,a,s'\rangle, f, \mathcal{D}^m, \mathcal{M})\big). \tag{5}$$

- *Maintained cost function* ($Cost$): values of $Cost(s,a,s')$ are initialized to $h(x,x')$, and then are selectively updated by $\hat{C}$ as the algorithm proceeds.

Overall, h is a very inexpensive operation compared to $\hat{C}$ that relies on calling a motion planner, and h underestimates motion cost. The following relationship among the three cost functions holds throughout the steps in PETLON:

$$h(x,x') \leq Cost(s,a,s') \leq \hat{C}(x,x'). \tag{6}$$

where $x \in f(s)$, $x' \in f(s')$, and a is an action that leads the state transition from s to s'.

Therefore, PETLON is efficient to the extent that it minimizes the number of times the motion planner is called, while still ensuring that the returned plan is the same as if the costs of all actions had been evaluated by the motion planner.

Algorithm 1 presents PETLON, taking the following terms as input:

Algorithm 1 PETLON algorithm

Require: s^{init}, x^{init},S^G,f, h, $\mathcal{D}^t$, $\mathcal{D}^m$,$\mathcal{M}$, $\mathcal{P}^t$, $\mathcal{P}^m$
Ensure: Symbolic plan p that is optimal at the task level
1: Initialize sampled poses $x \in f(s)$ and function *Cost* with h: $Cost(s,a,s') \leftarrow h(x,x')$
2: Initialize empty state-action-state array: A^{evld}
3: Initialize the *so-far-best* cost: $C^{sfb} \leftarrow Inf$
4: **while true do**
5: $[\hat{p}^*, P] \leftarrow \mathcal{P}^t(s^{init}, S^G, Cost, \mathcal{D}^t)$, where $\hat{p}^*$ is optimal given *Cost*, P includes a set of (near-optimal) plans, and $\hat{p}^* \in P$
6: **if** $\langle s,a,s' \rangle \in A^{evld}, \forall \langle s,a,s' \rangle \in \hat{p}^*$ **then**
7: **return** $\hat{p}^*$
8: **end if**
9: **for each** $p \in P$ **and** $Cost(p) < C^{sfb}$ **do**
10: **for each** $\langle s,a,s' \rangle \in p$ **do**
11: Update motion planner $\mathcal{P}^m$
12: **if** $\langle s,a,s' \rangle \notin A^{evld}$ **then**
13: **while** $x' \notin$ FreeSpace($\mathcal{D}^m|\mathcal{M}$) **do**
14: Re-sample $x' \in f(s')$
15: **end while**
16: Evaluate motion cost: $Cost(s,a.s') \leftarrow \hat{C}(x,x')$
17: Append $\langle s,a,s' \rangle$ to A^{evld}
18: **end if**
19: **end for**
20: $C_{plan} = \sum_{\langle s,a,s' \rangle \in p} Cost(s,a,s')$
21: $C^{sfb} = \min(C_{plan}, C^{sfb})$
22: **end for**
23: **end while**

- Initial state s^{init}, initial position x^{init} (in free space of $\mathcal{D}^m$), and goal specification S^G
- State mapping function $f : S \rightarrow X$
- Admissible heuristic cost function $h : (x,x') \rightarrow \mathbb{R}$ (to initialize the maintained cost function *Cost*)
- Task domain description $\mathcal{D}^t$, motion domain description $\mathcal{D}^m$, and robot model $\mathcal{M}$
- Task planner $\mathcal{P}^t : (s^{init}, S^G, Cost, \mathcal{D}^t) \rightarrow p$
- Motion planner $\mathcal{P}^m : (\langle s,a,s' \rangle, f, \mathcal{D}^m, \mathcal{M}) \rightarrow \xi$

PETLON starts by initializing: the maintained cost function *Cost* with our (admissible) heuristic function h (Line 1), an empty plan array to store evaluated state-action-state tuple (Line 2), and a scalar cost value C^{sfb} with *Inf*, indicating the "evaluated" cost of the *so-far-best* plan (Line 3).[2]

2. A "so-far-best" plan is the plan that is currently believed to be optimal using the maintained cost function. PETLON keeps updating the so-far-best plan as more action costs are evaluated.

Entering the first while-loop, PETLON computes a set of symbolic plans P, including the current estimate of optimal plan $\hat{p}^*$, using the maintained cost function *Cost*. If all actions in the current optimal solution have been evaluated, PETLON returns $\hat{p}^*$ as the optimal task-level solution p^*(Line 6-8). If not, PETLON enters the outer for-loop (Lines 9-22), processing one plan $p \in P$ at each iteration. In the inner for-loop, each state-action-state tuple is considered, in a forward order (Lines 10-19). First, $\mathcal{P}^m$ is updated based on post-conditions of the previous action (Line 11), to adapt to potential changes in $\mathcal{M}$, $\mathcal{D}^m$, and feasibility of sampled poses in $\mathcal{P}^m$. If state-action-state tuple $\langle s,a,s'\rangle$ has not been evaluated before, PETLON first checks the feasibility of the sampled end pose x' (Lines 13-15, not necessary if considering static robot configuration space), then evaluates the cost value by calling $\hat{C}$ (Line 16), and last appends $\langle s,a,s'\rangle$ to the evaluated state-action-state set A^{evld} (Line 17).

The so-far-best cost C^{sfb} is updated if the current plan has lower evaluated cost value C_{plan} (Line 21). It maintains the lowest evaluated cost value among all evaluated plans.

4.1 Proof of Task-Level Optimality of PETLON

Proposition 1. *Given motion planner $\mathcal{P}^m$ and state mapping function f, Algorithm 1 returns p^* that has the lowest cost over all satisfactory plans, i.e., the plan returned by Algorithm 1 satisfies Equation 3.*

Proof: Suppose this proposition is false, meaning that there exists at least one plan, p, whose geometric-level cost is lower than that of p^*, the plan returned by PETLON:

$$\sum_{\langle s,a,s'\rangle \in p} \hat{C}(s,a,s') < \sum_{\langle s,a,s'\rangle \in p^*} \hat{C}(s,a,s') \tag{7}$$

where $\hat{C}$ is the *evaluated cost function* as is detailed in Section 4 in the main paper.

In Algorithm 1, the task planner uses function *Cost*, the *maintained cost function*, and ensures that the returned plan is optimal given *Cost*:

$$\sum_{\langle s,a,s'\rangle \in p^*} Cost(s,a,s') \leq \sum_{\langle s,a,s'\rangle \in p} Cost(s,a,s'). \tag{8}$$

where p is an arbitrary satisfactory plan.

Algorithm 1 (Lines 12-18) also ensures that all action costs of the returned plan are evaluated by the motion planner:

$$\sum_{\langle s,a,s'\rangle \in p^*} Cost(s,a,s') = \sum_{\langle s,a,s'\rangle \in p^*} \hat{C}(s,a,s'). \tag{9}$$

We also know that the maintained cost function, *Cost*, cannot return a value that is higher than the evaluated cost, because function *Cost* is initialized by our heuristic cost function, h, that never overestimates the cost (Inequality 6):

$$\sum_{\langle s,a,s'\rangle \in p} Cost(s,a,s') \leq \sum_{\langle s,a,s'\rangle \in p} \hat{C}(s,a,s') \tag{10}$$

From Inequalities 8 and 10, and Equation 9, we draw conclusion:

$$\sum_{\langle s,a,s'\rangle \in p^*} \hat{C}(s,a,s') \leq \sum_{\langle s,a,s'\rangle \in p} \hat{C}(s,a,s') \tag{11}$$

Inequality 7 therefore contradicts with Inequality 11, proving the task-level optimality guarantee of p^*, which concludes the proof by contradiction. ■

PETLON practitioners should be aware that the task-level optimality of PETLON (Proposition 1) is guaranteed, only if the following requirements are satisfied in the implementation.

- The returned plan p^* is optimal using the maintained *Cost*, i.e., the task planner is optimal.
- The returned plan p^* has all its action costs evaluated by the motion planner.
- The heuristic function h is admissible.

The first two items lead to the following necessary condition for PETLON to ensure task-level optimality:

Corollary 1.1. *To ensure task-level optimality, Algorithm 1 (PETLON) must evaluate the optimal task-level solution at least one iteration prior to convergence.*

Proposition 2. *Given motion planner $\mathcal{P}^m$ and state mapping function f, PETLON returns the optimal task plan (p^*) in finite steps.*

Proof: Algorithm 1 terminates when the returned plan p has all its action costs evaluated by the motion planner, as suggested in Line 6. PETLON conducts at least one action cost evaluation at each iteration before termination. Given that the state space at the task level is finite, we can guarantee the convergence of Algorithm 1 within finite iterations. ■

Although Proposition 2 and its proof are rather straightforward, they are included here to highlight that PETLON terminates in finite time, while improving the clarity and completeness of this article. It should be noted that the above proof of PETLON's task-level optimality (Proposition 1) builds on a constrained form of the state mapping function (f) as explained in Section 3.2. Under the constrained form of f, PETLON is optimal at the task level, and is globally suboptimal. If we consider the original form of f, PETLON will be downgraded to a locally suboptimal algorithm: there can be shorter motion plan consistent with the task-level symbolic plan. It is acknowledged that there exist TMP algorithms that ensure local optimality using gradient-based optimization (Toussaint, 2015), and global optimality using abstractions (Vega-Brown & Roy, 2018). In contrast, we use off-the-shelf task planners from the literature of classical AI. Such planners provide more flexibility in task specification and planner construction, but are less flexible to the interplay with motion planners.

4.2 Variations of PETLON

Based on Corollary 1.1, in Line 5 of Algorithm 1, the returned plan set P by the task planner can include an arbitrary number of feasible plans for evaluation and not necessarily the optimal solution using *Cost* until close to convergence. Depending on the desired trade-off among task planner computation, number of iterations before convergence, and number of calls for motion evaluation $\hat{C}$, PETLON has various options for its task planner behavior on Line 5 [3]:

3. The options are not mutually exclusive, but rather indicate "properties" of the generated plans. Many task planners generate suboptimal plans before the optimal solution is computed, which can be used to populate the sets of plans specified by Options 1, 2, and 3.

Option 1. Find a non-empty set of feasible plans. (A plan is feasible if its final state in the goal state space $s_N \in S^G$ satisfies the goal specification.)

Option 2. Find a non-empty set of feasible plans that have plan cost less than or equal to C^{sfb} based on the maintained *Cost* function.

Option 3. Find a non-empty set of feasible plans that includes the optimal plan $\hat{p}^*$ based on the maintained *Cost* function.

The last option is the most computationally expensive, but is required at least one iteration before convergence to ensure task-level optimality, according to Corollary 1.1. The first and second options are computationally cheaper, and their output of feasible plans also helps narrow down the candidate set of potentially good plans. The higher the number of feasible plans that the task planner returns (with possibly higher computational cost), potentially the earlier PETLON converges. This trend arises because more potentially-good actions are returned at each iteration for evaluation.

Therefore, at each iteration, based on the choice of options in task planner behavior and how close the algorithm is to convergence, PETLON can choose to evaluate various numbers of plans in P that potentially contribute to the optimal solution. Convergence can be estimated by observing if C^{sfb} has improved in the past iterations. With different combinations of task planner behavior and number of actions being evaluated, PETLON can maintain the desired computation performance in the target domain. More practical configurations and implementation details can be found in Section 6. We here present the two basic configurations of PETLON that we use in our experiments:

- **OptOne**: At each iteration, given the maintained *Cost*, only the motion costs of the actions in the optimal $\hat{p}^*$ are evaluated.

- **OptAll**: At each iteration, the actions in all plans in P whose costs are lower than the so-far-best cost C^{sfb} (the lowest cost over all evaluated plans) are evaluated.

The two configurations both require that the optimal plan $\hat{p}^*$ is computed by the task planner in each iteration (Option 3). As OptOne only evaluates the optimal task plan $\hat{p}^*$, this configuration reduces the number of calls to the motion planner, which is beneficial when the motion planner is computationally expensive and the task planner is very efficient. OptAll on the other hand has computational advantages when the task planner is relatively expensive, because it trades more action cost evaluations for fewer iterations to converge compared to OptOne. It should be noted that OptAll requires the functionality of a task planner returning not only the optimal plans but also some suboptimal ones. This functionality is not supported by some off-the-shelf task planning systems, e.g., Fast-Downward (Helmert, 2006).

These two configurations are evaluated with two different task planners in Section 6, where more configurations are also introduced to illustrate the computational trade-offs.

Illustrative Example: Figure 3 shows a complete example of using PETLON (OptOne configuration) to generate a task-level-optimal solution without pre-evaluating costs of all actions. The robot needs to collect a hot dog (which requires a cooler) and a newspaper for camping, and move both of them to a storage room in the top-right corner. Lengths of trajectories in each subfigure represent the current *maintained cost function* of $Cost(s,a,s')$. Each line segment being replaced by a dashed trajectory corresponds to calling the *evaluated cost function* of $C(\hat{x},x')$ to evaluate action costs using

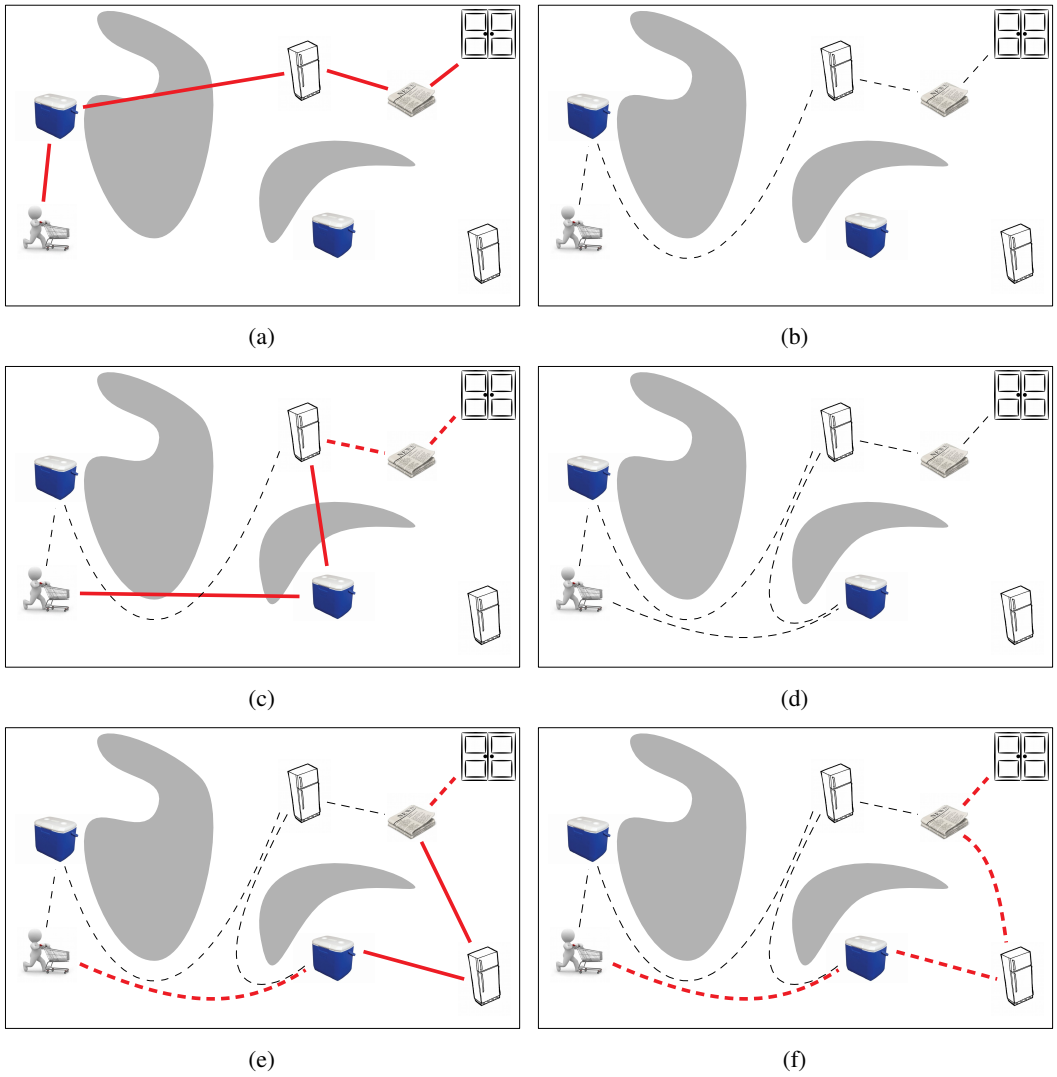


Figure 3: An illustrative example of PETLON (OptOne configuration) generating a task-level-optimal TMP solution, while avoiding pre-computing costs of all actions. Solid trajectories correspond to action costs from h, our admissible heuristic action cost function. Dashed trajectories correspond to action costs that are evaluated by the motion planner. Bold trajectories (in red color) correspond to the so-far-best solutions.

the motion planner, i.e., Line 16 in Algorithm 1. For instance, from Figure 3(c) to Figure 3(d), costs of two actions are evaluated by the motion planner.

At the beginning, the robot has not used the motion planner to evaluate any motion costs, and its *maintained cost function*, *Cost*, stores the same cost values as in the *heuristic cost function*, h. The task planner computes a plan as shown in Figure 3(a), which turns out to be suboptimal after the two cost evaluations shown in Figure 3(b). PETLON then merges the evaluated costs of the two

actions into *Cost*, and reactivates the task planner. The resulting plan is shown in Figure 3(c), which once again turns out to be suboptimal after costs of all involved actions are evaluated. After a total of seven action cost evaluations, the final TMP solution is generated as shown in Figure 3(f), which is guaranteed to be task-level-optimal as proven in Section 4.1.[4]

4.3 Task-level Optimality vs. Motion-level Optimality

Task-level optimality of a TMP problem is defined in Eq. 3 as to only optimize over the task action sequence, given sampled poses from f and motion planner $\mathcal{P}^m$. In comparison, to ensure motion-level optimality of TMP problems, the TMP system is required to directly optimize at the motion level, while considering the constraints from the task level. Motion-level optimality is equivalent to global optimality within the TMP context.

If the feasible sampled poses $X \in f(s)$ cover a relatively small area, which can be achievable by appropriately choosing the task-level state abstraction, the lack of global motion-level optimality does not deteriorate the final plan quality very much. This condition often holds in large-scale navigation domains. For example, when entering a narrow corridor, the precise location from where the robot enters has limited impact on the final travel length along the corridor.

However, in situations where this assumption is not true, to ensure the overall plan quality and therefore the meaningfulness of task-level optimality, one needs to take further steps: either to refine the task-level action description to cover smaller-area-of-coverage sets of potential action poses, or to maintain separate trees for each task plan p, which keep track of all sampled poses of each task-level action and optimizes globally the choices of samples along the task-action sequence. This procedure could be treated as a separate thread on motion refinement of task actions sequences, but would break the use of piece-wise motion refinement in PETLON to potentially achieve better overall plan quality. Due to the fact that such a process is unnecessary in our experiment domain (and most navigation domains in general) and the extra computation ($|X|^N$ to search among all possible poses for each plan p), we leave defining such a procedure for future work and focus here instead on ensuring PETLON's task-level optimality.

4.4 Highly Constrained and Dynamic Environments

In this subsection, we discuss the applications of PETLON to highly constrained and dynamic environments.

Highly Constrained Environments In highly constrained TMP domains (e.g., the ones that require manipulation actions in high-dimensional space), existing research has developed methods to ensure motion-level feasibility of task-level actions while remaining computationally efficient. Example methods include efficient end-pose re-sampling (Erdem et al., 2011; Srivastava et al., 2014), backtracking at the task level to re-sample action sequences (Stilman & Kuffner, 2005; Lagriffoul et al., 2014), and supervised learning approaches (Kim et al., 2018; Konidaris et al., 2018). Take action backtracking for example; when failing to sample feasible poses for the current symbolic state, it re-samples the end poses of previous task actions, in order to sample new initial poses for the current task action. PETLON can leverage these methods to enhance its efficiency and applicability

4. In comparison, current task-level-optimal methods, e.g., the one from Erdem et al. (2011), would require the agent to pre-evaluate costs of all actions. In this example, using these baseline methods, there are totally 21 actions whose costs must be pre-evaluated so as to guarantee the task-level-optimal solution.

in kinematically more challenging domains. TMP-N domains, e.g., robots operating in large areas such as airport terminals, do not generally fall into this category.

Theoretically, pose re-sampling triggers motion-cost updates of related task-level actions, i.e., costs of all $\langle s, a, s' \rangle$ tuples in A^{evld}, to ensure the task-level optimality suggested in Proposition 1 (Section 4.1). This process ensures that the costs of navigation actions in A^{evld} are up to date. As for our test domains, the re-sampling process (Line 14 in Algorithm 1) is seldom reached.

Dynamic Environments The evaluated cost function $\hat{C}$ defined in Equation 5 and implemented in Line 16 takes in the motion planning domain $\mathcal{D}^m$ and estimates the geometric-level cost of traversing from $x \in f(s)$ to $x' \in f(s')$. In static environments, such geometric-level cost corresponds to the effort required to travel between locations, taking into account the known static obstacles. Travel cost estimates through planning path length using traditional motion planning algorithms, e.g. A^*, Dijkstra's, PRM, and RRT, can then serve as a good metric given accurate map information and robot sensing capability, as the uncertainty in planning is limited in static environments.

Real-world environments are however oftentimes highly dynamic. It is still ongoing research to estimate travel cost in dynamic environments, given motion uncertainties by humans, time-variant events in the environment and limited sensing (Yi et al., 2015). We do not elaborate on those techniques, but acknowledge their impact on the plan quality of TMP solutions. They are important when deploying robots in the real world. Techniques to address such challenges are often developed as additive features to motion planning algorithms, e.g. to maintain a time-variant cost map, pre-trained from past data, for real-time travel estimation between waypoint traversal in the workspace (Shiarlis et al., 2017). These approaches will not break the task-level optimality guarantee of PETLON, as long as an admissible heuristic h (a lower-bounded cost estimator) and an upper bounded cost estimator $\hat{C}$ are available, as shown in Inequality 6.

4.5 Laziness of PETLON

PETLON has the spirit of *lazy learning* algorithms that "defer processing of their inputs until they receive requests for the information" (Aha, 1997). More specifically, both configurations of PETLON delay the evaluation of action costs until the corresponding actions are recommended by the task planner. Bohlin and Kavraki developed the Lazy PRM algorithm for efficient motion planning, where Lazy PRM defers collision checks in the roadmap until a path is generated (Bohlin & Kavraki, 2000), and this idea was combined with a bi-directional sampling strategy to further improve the performance (Sánchez & Latombe, 2002). Hauser developed lazy-PRM algorithms that go beyond generating admissible solutions, and further guarantee that the generated motion trajectories are asymptotically optimal (Hauser, 2015). Their methods delay collision tests until they are absolutely needed for checking that a candidate path is a solution. In contrast, the laziness of PETLON lies in the interplay between task and motion planners, and the motion planner not only checks feasibility but also evaluates action costs. Within the TMP context, Garrett et al. developed a new formulation, called a *constraint network*, for manipulation domains that can be modeled with factored transition systems (Garrett et al., 2018b). Their representation and algorithms have been demonstrated and evaluated using motion planning and pick-and-place tasks, and are particularly useful for manipulation domains with many objects. In particular, their *Focused Algorithm* uses lazy samples that are optimistically assumed to be satisfying all constraints; optimality was not their focus. In contrast, PETLON's laziness on the cost evaluation is in the space of complete motion trajectories, instead of in the high-dimensional configuration, or constraint space.

5. Algorithm Instantiation

Our task planner has been implemented using two declarative languages: Answer Set Programming (ASP) (Lifschitz, 2008; Gelfond & Kahl, 2014) and Planning Domain Definition Language (PDDL) (McDermott et al., 1998). ASP is a popular general knowledge representation and reasoning (KRR) language, and has been used for solving task planning problems (Lifschitz, 2002). PDDL was developed for the International Planning Competition (IPC) and has been maintained by the IPC community. A recent empirical comparison has shown that PDDL-based planners perform better when tasks require long solutions, and ASP-based planners perform better when tasks require complex reasoning (Jiang et al., 2019b). The goal of implementing both ASP-based and PDDL-based planners is to provide evidence for our hypothesis that PETLON is not sensitive to task planner selection.

At the task level, static predicates such as `has_door` and `inside` describe the spacial constraints on room accessibility and person locations respectively. We model three actions (`moveto`, `fetch`, and `deliver`), and represent the task states through non-static predicates (`delivered, loaded, at`). To map the task state to continuous space, we include a set of *geometric instances* using predicate `beside`, to describe task states in a small spacial area. For instance, `beside(fridge)` can be realized through function f as a pose within small area of the target fridge.

We use PRM* (Karaman & Frazzoli, 2011) to implement our motion planner. Compared to planners using a tree structure, such as RRT (LaValle, 1998), the graph structure of PRM* is computationally preferable due to the fact that the graph can be reused for multiple path evaluations with different start/end points. Further, its asymptotic optimality, meaning almost-sure convergence to the optimal solution, ensures higher-quality motion plans, at least when using high sampling density. In cases where the non-holonomic robot dynamics have non-negligible effects on estimated travel costs, the use of motion primitives (Butzke et al., 2014), or tree-based approaches which forward simulate robot motion are required. It should be noted that PETLON is not restricted to specific task or motion planning algorithms. The higher complexity the motion planner has, the greater the potential advantage PETLON can bring by saving computation for motion evaluation.

We implemented our ASP-based and PDDL-based task planners using award-winning solvers of Clingo (Gebser et al., 2014) and FastDownward (FD) (Helmert, 2006) respectively. As mentioned in Section 4.2, suboptimal plans can be used in early phases of PETLON to quickly update C^{sfb}. Planners in their greedy modes to find a solution as quickly as possible serve well for this purpose. We primarily evaluated PETLON with Options 2 and 3 (Section 4.2), which facilitate the planner configurations OptAll and OptOne, as discussed in Section 4.2. When using either OptOne or OptAll, task-level optimality is guaranteed per Proposition 1. Per Corollary 1.1, different planners (Options 1, 2, or 3) can be used during different planning iterations to maximize computational efficiency while still ensuring optimality at convergence. For instance, using Option 1, e.g. greedy best-first search algorithms, can reduce early-stage computation requirements before switching to an optimal planner (Option 3) to ensure optimality. The performance impact given different implementations of Option 1 is briefly discussed along with the Anytime property of PETLON in Section 6.3. The main reason for excluding Option 1 from the evaluation is that the resulting performance highly depends on the implementation of the greedy planner, making quantitative results less meaningful.

We use a laptop equipped with 2.2GHz i7 processor and 16GB RAM on OS X for all reported results.

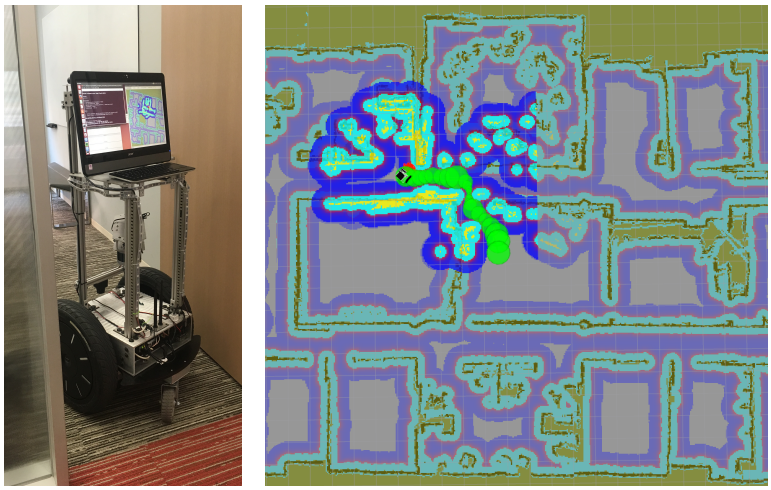


Figure 4: Left: Robot platform used in this research. Right: Occupancy-grid map (inflated) and a motion trajectory, indicated by a sequence of green planned for navigating through an office door.

6. Experiments

PETLON has been implemented on a real robot as shown in Figure 4 (Left). The right of the figure shows the occupancy-grid map and the robot planning to navigate through an office door. The robot uses an RMP 110 mobile platform, onboard auxiliary battery, desktop computer (with touchscreen), and Velodyne VLP-16 for perception (Khandelwal et al., 2017).

The test domain is our office environment, with the map pre-scanned and constructed by running the SLAM algorithm (Thrun et al., 2005) (loaded before robot planning). Part of the domain is shown in Figure 4 on the right, containing seven rooms, four people, and four types of items (accordingly four types of containers). Each type of container has two to three instances. This test domain is later referred to as the **base domain**, on which we create variant domains for evaluating PETLON in different categories.

6.1 Baseline Methods

The goal of PETLON is to significantly reduce overall planning time while guaranteeing task-level optimality. Therefore, we evaluate PETLON based on both computational time and resulting plan quality by comparing PETLON (two configurations) to baselines with the following action cost formulations:

- *Constant cost:* Task actions are assumed to share the same unit cost. As a result, task planners generate plans with the fewest actions ($Cost = 1$). Our hypothesis was that this baseline would perform the worst in plan quality and the best in efficiency (due to the absence of a motion planner).
- *Heuristic cost:* Task actions are assumed to have cost equivalent to the Euclidean distance traveled (the motion planner is never called, and $Cost{=}h$ all the time).

- *Brute force:* Costs of all task actions are evaluated by the motion planner beforehand ($Cost$=$\hat{C}$). Our hypothesis was that this baseline would produce task-level-optimal solutions, but does not perform as well as PETLON in efficiency.

All three baselines use optimal task planners. However, depending on the motion cost evaluation strategy, these baselines produce trajectories of dramatically different quality. Only the brute-force baseline and PETLON (both configurations) generate task-level-optimal plans. Their computational times are then compared to demonstrate the improvement in planning efficiency introduced by PETLON. The other two baselines produce task-level-suboptimal plans.

6.2 Illustrative Example

A plan quality comparison between the output from PETLON and the output from the *heuristic cost* baseline is shown in Figure 5. In this example, the robot needs to deliver a bottle of juice (5 instances marked as blue downward triangles) and a newspaper (4 instances marked as magenta upward triangles) to a target person (solid green circle). The initial state is specified using `at(corridor)` and `beside(init_pos)`, and the goal state is specified using the following four literals:

```
delivered(alice,n). newspaper(n). delivered(alice,j). juice(j).
```

where `n,j` are available locations of newspapers and juices.

While considering an environment with low visibility from one location to another, such as an office domain, heuristic cost functions (such as suggested in Equation 4) may greatly underestimate the true motion cost value, resulting in suboptimal plans (50m vs. 37m in travel length in this case). In this example, the geometric instances in the task planning domain are of four types: `fridge`, `newsstand`, `door`, and `person`. The task planner decides the order of the subtasks (such as `fetch(newspaper)`), and which instance to fulfill the action preconditions (such as `beside(newsstand1)`).

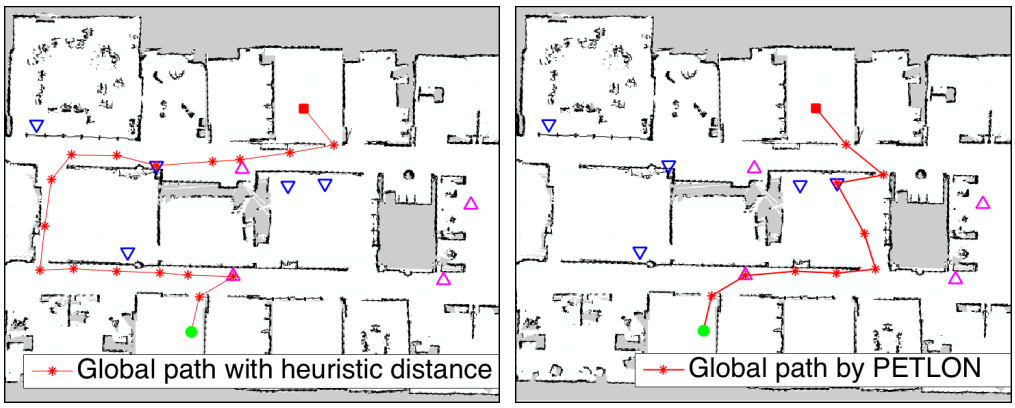
This pairwise example illustrates the necessity of cost evaluations using a motion planner (the baseline does not do so) and the importance of task-level optimality (the suboptimal solution causes a significant delay of task completion).

6.3 Experimental Results

Our central hypothesis is that PETLON is more efficient than planning approaches that pre-compute motion costs of all possible navigation actions, while still producing task-level-optimal solutions. Accordingly, we conducted the following four sets of experiments focusing on evaluating the performance of PETLON in efficiency and plan quality under different conditions.

For the motion planner, we draw samples with a density of two poses per square meter, and the resultant plan quality has small variations (mostly within one meter) among trials.

Overall Performance of Six Planning Strategies The two PETLON configurations with Clingo and FD implementations are compared with the baselines, on the task of delivering two specified kinds of items to a target person. As explained in Section 5, FD does not output suboptimal solutions, so we can only evaluate the OptOne configuration of PETLON using the FD solver. Each data point corresponds to an average over eight trials, and there are six strategies in total in this set of experiments.



(a) Heuristic cost baseline (b) PETLON

Figure 5: The heuristic cost baseline uses only the Euclidean distance for plan cost value estimate (28.4m) and results in the suboptimal solution with actual length as 50m. The optimal solution by PETLON has higher heuristic cost value estimate (31.3m) but shorter actual length as 37m, compared to the heuristic-cost baseline.

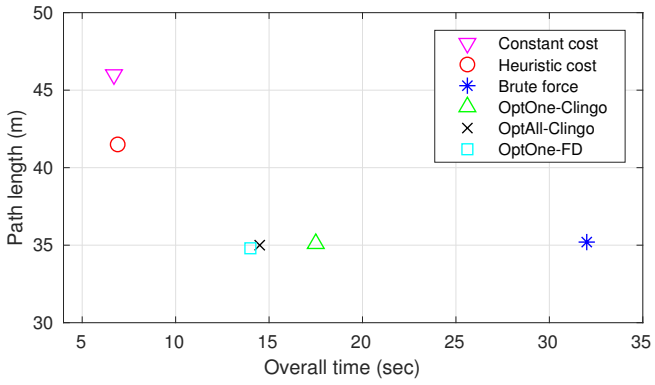


Figure 6: Plan quality (path length) vs. overall planning time. We compare six different planning algorithms using the base domain. PETLON (three implementations) significantly reduces the overall planning time, while ensuring task-level optimality. Since we prioritize optimality over planning efficiency in our evaluations, the constant-cost and heuristic-cost baselines are not included in the following evaluations.

Figure 6 reports their overall performance. We can see that PETLON (all three versions) significantly reduces the planning time (x-axis) to less than 20 seconds, in comparison to the brute-force baseline that took more than 30 seconds, while ensuring the best-quality solution (y-axis). It should be noted that, our domain is not very knowledge-intensive in the sense of the numbers of objects and their properties. Real-world applications are frequently much more knowledge-intensive than the brute-force baseline is able to handle; in domains with many objects that are irrelevant to the tested

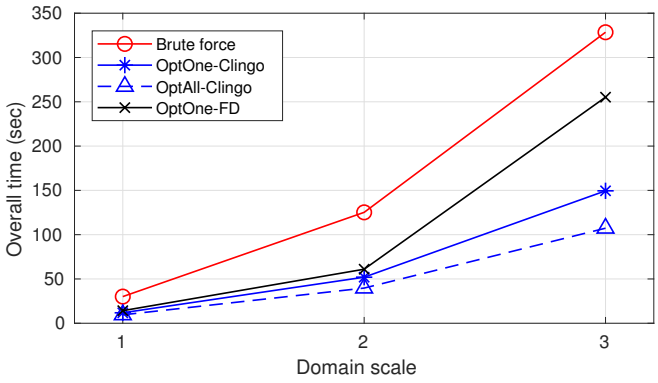


Figure 7: Overall planning time given different domain scale-up. PETLON (three versions) is more efficient than the baseline in every implementation.

	Domain scale		
	1	2	3
OptOne-Clingo	10.75	9.00	11.00
OptOne-FD	13.60	11.00	12.00
OptAll-Clingo	16.00	17.50	25.75
Brute-force baseline	325.00	1295.00	2850.00

Table 1: The average number of cost evaluations conducted in the motion planner, in different domain scales.

task request, the brute-force runtime grows exponentially (in the number of objects), and hence is not applicable for relatively large domains. The outputs of the other two baselines, *Constant cost* and *Heuristic cost*, are much worse in terms of quality due to their ignorance of true action costs.

Efficiency (Planning Time) in Domains of Different Sizes In this experiment, with the same task specification, we scale up the domain size in terms of both the number of objects (adding complexity for $\mathcal{P}^t$), and map size (adding complexity for $\mathcal{P}^m$) by appending N copies of the base domain to one another (left to right, then top to down), following its same structure. The initial positions of the robot are randomly selected in the three domains, to increase the likelihood that the robot will traverse newly-appended map areas.

The overall planning time includes the time consumed by both the task planner (Clingo-based or FD-based) and the motion planner. The task planners and motion planner are sensitive to the increasing number of objects and map size respectively. It should be noted that we add more objects and increase domain sizes, but the delivery task, as delivering a juice and a newspaper, does not change. Correspondingly, the length of the generated symbolic plan does not change.

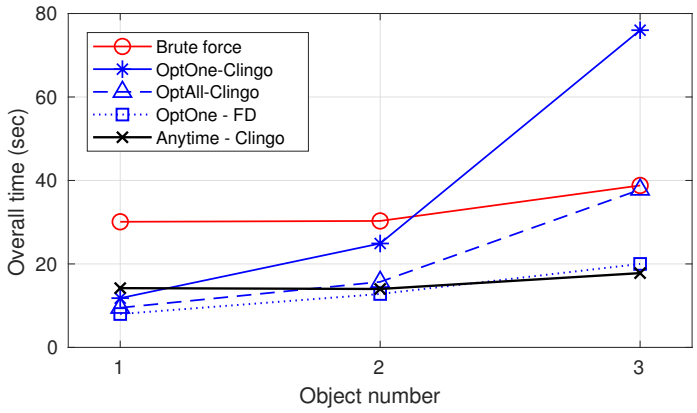


Figure 8: Overall planning time given tasks of different numbers of object to deliver. PETLON that uses the FD-based task planner (OptOne configuration) performs the best. The anytime configuration greatly improves computation compared to other ASP-based implementations.

Figure 7 shows the results of overall planning time given different levels of domain scale-up ($N = 2$ and $N = 3$). We see PETLON performs significantly better than the brute-force baseline, because brute-force has to evaluate costs of a combinatorially growing number of actions. Table 1 shows the number of motion cost evaluations in different domain scales. From the last row, we can see that in the domain setting of $3x$ scale-up, brute-force conducts 2850 motion cost evaluations, whereas PETLON evaluates fewer than 26.

Planning Time Given Different Tasks In this set of experiments, we vary the tasks by increasing the number of objects that need to be delivered using the base domain setup. The goal is to evaluate how sensitive the planning algorithms are given more complex tasks, i.e., tasks that require more task-level actions. PETLON may take substantially many iterations to converge given a task that requires many actions, so the computational expense of task planning can become a concern. We used both Clingo and FD task planners in our experiment. Intuitively, Clingo can be relatively more sensitive to plan length as it is a general-purpose reasoner not fine-tuned for planning tasks, whereas the FD planner is developed specifically for efficiently computing plans that include many actions.

The results are shown in Figure 8. As plan length increases (x-axis), OptOne-FD begins outperforming all other implementations, whereas the Clingo task planner is very sensitive to plan length. In such scenarios, making more calls to the task planner may not trade off favorably against motion evaluation. To address this issue, we introduce the "Anytime" configuration of PETLON, which trades off the optimality guarantee with superior efficiency without much loss of plan quality.

Anytime Property of PETLON, an Illustrative Example In situations with strict time bounds, it can be useful for a planner to have an "anytime" property, meaning that the algorithm can terminate early while outputting monotonically improved solutions over time. In our case, we would like to see that PETLON produces good-quality plans (sequence of actions) given an early termination.

Figure 9 shows how the so-far-best cost C^{sfb} and the cost of $\hat{p}^*$ computed using *Cost* evolve over nine iterations until convergence. Note that C^{sfb} reaches the optimal value of PETLON (OptOne on

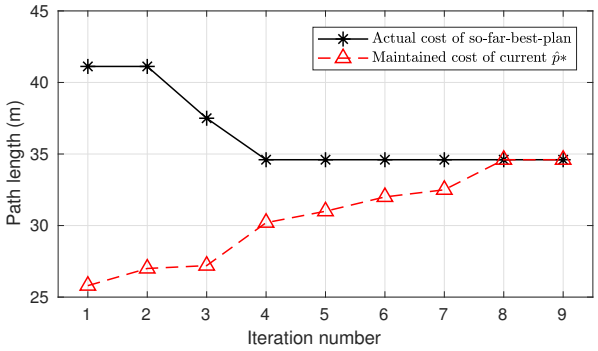


Figure 9: PETLON is an anytime algorithm that can return a valid solution even if it is interrupted before termination.

	Number of object		
	2	3	4
OptAll-Clingo	29.6	35.1	44.9
AnyTime-Clingo	29.9	35.1	45.0

Table 2: Plan quality analysis for two to four objects of delivery, units in meter(m): the Anytime configuration implements OptAll and terminates after the first iteration. Compared to OptAll, which does not terminate until convergence, the high quality of the Anytime configuration shows the potentially beneficial trade-off between computational efficiency and guaranteed optimality.

the base domain) by iteration 4, while PETLON continues evaluating other potential satisfiable plans and finally, at iteration 9, it reports that the plan found at iteration 4 is optimal. This illustrative trial demonstrates PETLON's good anytime performance. We collected the time lengths of PETLON finding the optimal solution (after that, PETLON continues to evaluate other plans to ensure task-level optimality).

Now, let us come back to the results presented in Figure 8 (the "Anytime-Clingo" curve in particular). We can see Anytime-Clingo performs the best in terms of computational time. Table 2 shows that Anytime-Clingo produces near-optimal solutions at task level. The computational results of Anytime-Clingo are reported along with other implementations in Figure 8, where we can see Anytime-Clingo performs the best in comparison to all other planning strategies.

Note that, Clingo is actually capable of outputting all feasible plans, or all plans with costs lower than a certain value. With those two implementations, PETLON guarantees optimality within *two* calls of the task planner, by setting the second call to output all plans lower than the so-far-best cost, or, to output all feasible plans in the first iteration, and evaluate the rest without more calls to the task planner. This implementation is practical given its early convergence and therefore fewer calls to the task planner. Moreover, due to the convergence criteria of PETLON (per Corollary 1.1), PETLON can apply any efficient task planner in the first iteration, e.g. a best-first search algorithm, to save computation. PETLON can also directly make use of anytime task planners, such as LAMA (Richter & Westphal, 2010) and JASPER (Xie et al., 2014), while still guaranteeing

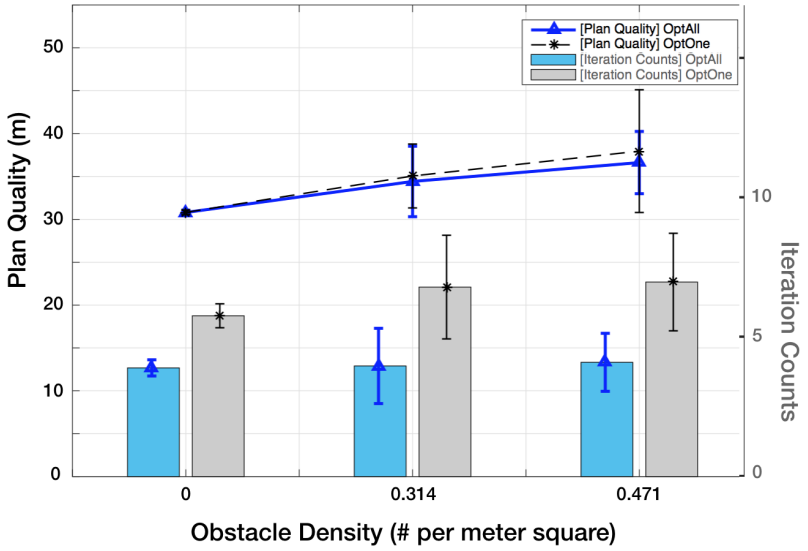


Figure 10: Plan quality and computation analysis given different density of randomly generated obstacles: the more dense the obstacles are, the lower the plan quality is, and the more iterations PETLON take to converge. Such performance degradation is more significant in the OptOne configuration than in OptAll.

task-level optimality. The experimental setup in Figure 8 is purely to demonstrate potential issues while using PETLON, given different characteristics brought about by the choices of planners.

Domains with Different Levels of Visibility In motion planning, visibility function $V(q)$ maps point q to a set of points that are reachable through a local path planner, where a line segment is the simplest implementation of a local path planner (LaValle, 2006). When planning to travel in an environment with perfect visibility, i.e., an obstacle-free environment, the Euclidean-distance heuristic is equivalent to the length of the motion path required to travel between two locations. As the environment becomes more cluttered, the heuristic estimate becomes less effective, and PETLON relies more on the motion planner to evaluate the true travel cost, which causes extra computational expense. Our hypothesis is that, the more cluttered the environment is, the more motion evaluations/iterations are required to produce the task-level-optimal solution.

In this set of experiments, the robot plans in the base domain to fetch two objects for delivery. We manipulated domain visibility by randomly sampling different numbers of obstacles, producing different obstacle densities. Obstacles are of size $0.4m$ by $0.4m$, and are large enough to potentially block the office entrances or prevent the robot from fetching objects in areas. Table 3 shows that a larger number of obstacles renders more trials that are identified as being infeasible by the motion planner.

We forced PETLON to terminate by iteration 10 (if it is ever reached) in each trial. We tested both OptOne and OptAll configurations, using Clingo for task planning and PRM* for motion planning. Within PRM*, we used the same setting, described in Section 5, in all experiments (including the same sample density and number of nearest neighbors) despite their different levels of visibility.

	Number of obstacles		
	0	120	180
OptAll-Clingo	0%	3%	23%
OptOne-Clingo	0%	17%	28%

Table 3: Percentage of trials (out of totally 64 trials) identified as being infeasible by the motion planner. As more obstacles are added, the environment's the visibility decreases, and the motion planner finds it more difficult to compute geometrically feasible plans, which results in more infeasible trials.

The experimental results are shown in Figure 10, where each data point corresponds to an average of 64 trials, and infeasible trials have been excluded. As the environment becomes more cluttered, the plan quality declines, which corresponds to the generated plan length being higher. This is not only because the robot needs longer paths to avoid obstacles, but also because it is less likely that the generated plan converges to the optimal within the allowed ten iterations. This plan quality reduction is more severe in OptOne, in comparison to OptAll.

7. Conclusions and Future Work

In this article, we introduce a novel algorithm, PETLON, that fully integrates task and motion planning for mobile robot service tasks. PETLON stands for "Planning Efficiently for Task-Level-Optimal Navigation" and is designed to produce task-level optimal plans while maintaining efficient planning time. PETLON has been evaluated using maps modeled after a real office environment, and has also been implemented on a real robot. Results show that PETLON significantly reduces the overall planning time compared to a baseline that pre-computes motion costs of all actions, while still maintaining task-level optimality.

This work opens a number of new research directions on task and motion planning (TMP), especially in navigational domains (TMP-N). In the future, we intend to extend the work to dynamic environments, where cost estimates reflect plan quality during real-world execution. Currently we only use the total length of motion trajectories to evaluate the quality of task plans. Other motion criteria can be considered for cost evaluations. For instance, actions that involve navigation through crowds are expected to produce higher costs due to the expected delay in execution. In domains where safety is a major concern, safety measures can also be incorporated into the evaluation process (Ding et al., 2020). We also intend to extend the work with an exploration mechanism, for the robot to interact with the real world to learn costs of navigation actions (Jiang et al., 2019a).

One of the limitations of this research is the assumption of the existence of a "state mapping function" presented in Section 3. This function breaks global optimality, and downgrades PETLON to a task-level-optimal approach. Also, this function itself can be computationally costly, especially in highly constrained or dynamic environments. There are incremental TMP methods from the literature that incrementally incorporate motion feasibility information into the task planner, resulting in a probabilistically-complete TMP method called IDTMP (Dantam et al., 2016). A promising direction for future work is to exploit the complementary features of PETLON's optimality and IDTMP's completeness toward more advanced TMP algorithms.

As discussed in Section 2, researchers have developed a number of benchmark domains for evaluations of TMP algorithms. The work of Lagriffoul et al. (2018) is one such effort. We plan to better standardize the tasks used in this article, including "camping" and "delivery" tasks, and use them as benchmarks for the evaluation of TMP-N algorithms. On the other hand, although developed for navigation domains, there is the potential to apply PETLON in manipulation domains, where pose sampling and action feasibility becomes challenging.

Acknowledgements

A portion of this work has taken place in the Learning Agents Research Group (LARG) at the Artificial Intelligence Laboratory, The University of Texas at Austin. LARG research is supported in part by grants from the National Science Foundation (CPS-1739964, IIS-1724157, NRI-1925082), the Office of Naval Research (N00014-18-2243), Future of Life Institute (RFP2-000), Army Research Office (W911NF-19-2-0333), DARPA, Lockheed Martin, General Motors, and Bosch. The views and conclusions contained in this document are those of the authors alone. Peter Stone serves as the Executive Director of Sony AI America and receives financial compensation for this work. The terms of this arrangement have been reviewed and approved by the University of Texas at Austin in accordance with its policy on objectivity in research. Zhang's work is supported in part by grants from the National Science Foundation (NRI-1925044), Ford Motor Company (URP Awards), OPPO (Faculty Research Award), and SUNY Research Foundation.

References

Aha, D. W. (Ed.). (1997). *Lazy Learning*. Kluwer Academic Publishers, Norwell, MA, USA.

Bohlin, R., & Kavraki, L. E. (2000). Path planning using lazy prm. In *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 521–528.

Butzke, J., Sapkota, K., Prasad, K., MacAllister, B., & Likhachev, M. (2014). State lattice with controllers: Augmenting lattice-based path planning with controller-based motion primitives. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pp. 258–265. IEEE.

Cambon, S., Alami, R., & Gravot, F. (2009). A hybrid approach to intricate motion, manipulation and task planning. *The International Journal of Robotics Research*, *28*(1), 104–126.

Chitnis, R., Hadfield-Menell, D., Gupta, A., Srivastava, S., Groshev, E., Lin, C., & Abbeel, P. (2016). Guided search for task and motion plans using learned heuristics. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pp. 447–454. IEEE.

Dantam, N. T., Kingston, Z. K., Chaudhuri, S., & Kavraki, L. E. (2016). Incremental task and motion planning: A constraint-based approach.. In *Robotics: Science and systems*, Vol. 12, p. 00052. Ann Arbor, MI, USA.

Dantam, N. T., Kingston, Z. K., Chaudhuri, S., & Kavraki, L. E. (2018). An incremental constraint-based framework for task and motion planning. *The International Journal of Robotics Research*.

De Moura, L., & Bjørner, N. (2011). Satisfiability modulo theories: introduction and applications. *Communications of the ACM*, *54*(9), 69–77.

Ding, Y., Zhang, X., Zhan, X., & Zhang, S. (2020). Task-motion planning for safe and efficient urban driving. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Erdem, E., Haspalamutgil, K., Palaz, C., Patoglu, V., & Uras, T. (2011). Combining high-level causal reasoning with low-level geometric reasoning and motion planning for robotic manipulation. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 4575–4581. IEEE.

Fikes, R. E., & Nilsson, N. J. (1971). Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, *2*(3-4), 189–208.

Garrett, C. R., Lozano-Pérez, T., & Kaelbling, L. P. (2017a). Sample-based methods for factored task and motion planning. In *Robotics: Science and Systems*.

Garrett, C. R., Lozano-Pérez, T., & Kaelbling, L. P. (2017b). Strips planning in infinite domains. *arXiv preprint arXiv:1701.00287*.

Garrett, C. R., Lozano-Pérez, T., & Kaelbling, L. P. (2018a). Ffrob: Leveraging symbolic planning for efficient task and motion planning. *The International Journal of Robotics Research*, *37*(1), 104–136.

Garrett, C. R., Lozano-Pérez, T., & Kaelbling, L. P. (2018b). Sampling-based methods for factored task and motion planning. *The International Journal of Robotics Research*, *37*(13-14), 1796–1825.

Gebser, M., Kaminski, R., Kaufmann, B., & Schaub, T. (2014). Clingo= asp+ control: Preliminary report. *arXiv preprint arXiv:1405.3694*.

Gelfond, M., & Kahl, Y. (2014). *Knowledge representation, reasoning, and the design of intelligent agents: The answer-set programming approach*. Cambridge University Press.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems*, pp. 2672–2680.

Gravot, F., Cambon, S., & Alami, R. (2005). asymov: a planner that deals with intricate symbolic and geometric problems. In *Robotics Research. The Eleventh International Symposium*, pp. 100–110. Springer.

Hauser, K. (2015). Lazy collision checking in asymptotically-optimal motion planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2951–2957. IEEE.

Hawes, N., Burbridge, C., Jovan, F., et al. (2017). The strands project: Long-term autonomy in everyday environments. *IEEE Robotics & Automation Magazine*, *24*(3), 146–156.

Helmert, M. (2006). The fast downward planning system. *Journal of Artificial Intelligence Research*, *26*, 191–246.

Hoffmann, J. (2003). The metric-ff planning system: Translating"ignoring delete lists"to numeric state variables. *Journal of Artificial Intelligence Research*, *20*, 291–341.

Hoffmann, J., & Nebel, B. (2001). The ff planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, *14*, 253–302.

Jiang, Y., Yang, F., Zhang, S., & Stone, P. (2019a). Task-motion planning with reinforcement learning for adaptable mobile service robots. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 7529–7534. IEEE.

Jiang, Y., Zhang, S., Khandelwal, P., & Stone, P. (2019b). Task planning in robotics: an empirical comparison of PDDL- and ASP-based systems. *Frontiers of Information Technology & Electronic Engineering*, *20*(3), 363–373.

Kaelbling, L. P., & Lozano-Pérez, T. (2011). Hierarchical task and motion planning in the now. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 1470–1477. IEEE.

Kaelbling, L. P., & Lozano-Pérez, T. (2013). Integrated task and motion planning in belief space. *The International Journal of Robotics Research*, *32*(9-10), 1194–1227.

Karaman, S., & Frazzoli, E. (2011). Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, *30*(7), 846–894.

Khandelwal, P., Zhang, S., Sinapov, J., Leonetti, M., Thomason, J., Yang, F., Gori, I., Svetlik, M., Khante, P., Lifschitz, V., et al. (2017). Bwibots: A platform for bridging the gap between ai and human–robot interaction research. *The International Journal of Robotics Research*, *36*(5-7), 635–659.

Kim, B., Kaelbling, L. P., & Lozano-Pérez, T. (2017). Learning to guide task and motion planning using score-space representation. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pp. 2810–2817. IEEE.

Kim, B., Kaelbling, L. P., & Lozano-Pérez, T. (2018). Guiding search in continuous state-action spaces by learning an action sampler from off-target search experience. In *Proceedings of Thirty-Second AAAI Conference on Artificial Intelligence*.

Konidaris, G., Kaelbling, L. P., & Lozano-Perez, T. (2018). From skills to symbols: Learning symbolic representations for abstract high-level planning. *Journal of Artificial Intelligence Research*, *61*, 215–289.

Lagriffoul, F., Dantam, N. T., Garrett, C., Akbari, A., Srivastava, S., & Kavraki, L. E. (2018). Platform-independent benchmarks for task and motion planning. *IEEE Robotics and Automation Letters*, *3*(4), 3765–3772.

Lagriffoul, F., Dimitrov, D., Bidot, J., Saffiotti, A., & Karlsson, L. (2014). Efficiently combining task and motion planning using geometric constraints. *The International Journal of Robotics Research*, *33*(14), 1726–1747.

Latombe, J.-C. (2012). *Robot motion planning*. Springer Science & Business Media.

LaValle, S. M. (1998). Rapidly-exploring random trees: A new tool for path planning..

LaValle, S. M. (2006). *Planning algorithms*. Cambridge university press.

Lifschitz, V. (2002). Answer set programming and plan generation. *Artificial Intelligence*, *138*(1), 39–54.

Lifschitz, V. (2008). What is answer set programming?. In *Proceedings of the 23rd national conference on Artificial intelligence-Volume 3*, pp. 1594–1597. AAAI Press.

Lo, S.-Y., Zhang, S., & Stone, P. (2018). Petlon: Planning efficiently for task-level-optimal navigation. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pp. 220–228.

McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D., & Wilkins, D. (1998). Pddl-the planning domain definition language..

Nau, D. S., Au, T.-C., Ilghami, O., Kuter, U., Murdock, J. W., Wu, D., & Yaman, F. (2003). Shop2: An htn planning system. *Journal of artificial intelligence research*, *20*, 379–404.

Nilsson, N. J. (1984). Shakey the robot. Tech. rep., DTIC Document.

Pineau, J., Montemerlo, M., Pollack, M., Roy, N., & Thrun, S. (2003). Towards robotic assistants in nursing homes: Challenges and results. *Robotics and autonomous systems*, *42*(3-4), 271–281.

Plaku, E., & Hager, G. D. (2010). Sampling-based motion and symbolic action planning with geometric and differential constraints. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pp. 5002–5008. IEEE.

Richter, S., & Westphal, M. (2010). The lama planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research*, *39*, 127–177.

Sacerdoti, E. D. (1974). Planning in a hierarchy of abstraction spaces. *Artificial intelligence*, *5*(2), 115–135.

Sánchez, G., & Latombe, J.-C. (2002). On delaying collision checking in prm planning: Application to multi-robot coordination. *The International Journal of Robotics Research*, *21*(1), 5–26.

Schmitt, P. S., Neubauer, W., Feiten, W., Wurm, K. M., Wichert, G. V., & Burgard, W. (2017). Optimal, sampling-based manipulation planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3426–3432.

Shiarlis, K., Messias, J., & Whiteson, S. (2017). Rapidly exploring learning trees. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pp. 1541–1548. IEEE.

Simeon, T., Laumond, J.-P., & Lamiraux, F. (2001). Move3d: A generic platform for path planning. In *Proceedings of the 2001 IEEE International Symposium on Assembly and Task Planning (ISATP2001)*, pp. 25–30. IEEE.

Srivastava, S., Fang, E., Riano, L., Chitnis, R., Russell, S., & Abbeel, P. (2014). Combined task and motion planning through an extensible planner-independent interface layer. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pp. 639–646. IEEE.

Stilman, M., & Kuffner, J. J. (2005). Navigation among movable obstacles: Real-time reasoning in complex environments. *International Journal of Humanoid Robotics*, *2*(04), 479–503.

Sutton, R. S., Precup, D., & Singh, S. (1999). Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, *112*(1-2), 181–211.

Thrun, S., Burgard, W., & Fox, D. (2005). *Probabilistic robotics*. MIT press.

Toussaint, M. (2015). Logic-geometric programming: An optimization-based approach to combined task and motion planning. In *International Joint Conference on Artificial Intelligence*.

Vega-Brown, W., & Roy, N. (2016). Asymptotically optimal planning under piecewise-analytic constraints. In *Workshop on the Algorithmic Foundations of Robotics*.

Vega-Brown, W., & Roy, N. (2018). Admissible abstractions for near-optimal task and motion planning. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, pp. 4852–4859.

Veloso, M. M. (2018). The increasingly fascinating opportunity for human-robot-ai interaction: The cobot mobile service robots. *ACM Transactions on Human-Robot Interaction (THRI)*, *7*(1), 5.

Wang, Y., Dantam, N. T., Chaudhuri, S., & Kavraki, L. E. (2016). Task and motion policy synthesis as liveness games. In *Proceedings of the Twenty-Sixth International Conference on International Conference on Automated Planning and Scheduling*, pp. 536–540. AAAI Press.

Wolfe, J., Marthi, B., & Russell, S. (2010). Combined task and motion planning for mobile manipulation. In *Proceedings of the Twentieth International Conference on Automated Planning and Scheduling*, pp. 254–257. AAAI Press.

Xie, F., Müller, M., & Holte, R. (2014). Jasper: the art of exploration in greedy best first search. *The Eighth International Planning Competition (IPC-2014)*, 39–42.

Yi, S., Li, H., & Wang, X. (2015). Understanding pedestrian behaviors from stationary crowd groups. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3488–3496.

Zhang, S., Sridharan, M., & Washington, C. (2013). Active visual planning for mobile robot teams using hierarchical pomdps. *IEEE Transactions on Robotics*, *29*(4), 975–985.

Zhang, S., Yang, F., Khandelwal, P., & Stone, P. (2015). Mobile robot planning using action language bc with an abstraction hierarchy. In *International Conference on Logic Programming and Nonmonotonic Reasoning*, pp. 502–516. Springer.