Integrated Task and Motion Planning

Caelan Reed Garrett [1], Rohan Chitnis [1], Rachel Holladay [1], Beomjoon Kim [1], Tom Silver [1], Leslie Pack Kaelbling [1] and Tomás Lozano-Pérez [1]

[1]CSAIL, MIT, Cambridge, USA, 02139; email: caelan@csail.mit.edu

Annu. Rev. Control Robot. Auton. Syst. 2021. 2021. 4:1–30

Keywords

task and motion planning, robotics, automated planning, motion planning, manipulation planning

Abstract

The problem of planning for a robot that operates in environments containing a large number of objects, taking actions to move itself through the world as well as to change the state of the objects, is known as *task and motion planning* (TAMP). TAMP problems contain elements of discrete task planning, discrete-continuous mathematical programming, and continuous motion planning, and thus cannot be effectively addressed by any of these fields directly. In this paper, we define a class of TAMP problems and survey algorithms for solving them, characterizing the solution methods in terms of their strategies for solving the continuous-space subproblems and their techniques for integrating the discrete and continuous components of the search.

Contents

1. INTRODUCTION

Robots are playing an increasingly important role in society, and their range of applications is rapidly expanding. These applications have traditionally been in *structured* environments, such as factories, where the robot's interactions are limited and a behavior can be directly specified by a human. However, many of the most exciting potential applications of robots are in highly *unstructured* human environments such as homes, hospitals, or construction sites. In these applications, the robot will generally be tasked with a specific goal, such as cooking and delivering a meal to an elderly resident, but the actions necessary to achieve the goal will vary enormously depending on the state of the environment. For example, the robot might need to open cupboards and remove objects in order to retrieve a bowl that is necessary for preparing the meal (Figure 1). Directly specifying the full behavior policy for a robot operating in these unstructured environments is not practical because the required policy is too complex.

Since the earliest days of robotics, there has been an interest in automated *planning*, developing algorithms for deciding what sequence of commands the robot should execute in order to accomplish some goal (1, 2). The first class of planning problems that arises is to move the robot from one state to another without colliding with objects in the world. This *motion planning* problem was formulated by Lozano-Pérez (3) as a search for paths through the robot's *configuration space*, a space with dimensions representing the controllable joints of the robot, and has been the focus of a great deal of algorithmic development. The most effective methods are based on sampling (4, 5) or constrained optimization (6, 7).

Collision-free robot motion is important but does not enable the robot to alter the world. In order for the robot to, for example, move objects by picking them up and placing them, planning needs to consider a much larger space that encompasses the entire state of the world, which includes any objects the robot has grasped, the grasps it is using, and the poses of the other objects. Conceptually, it makes sense to try to directly extend motion

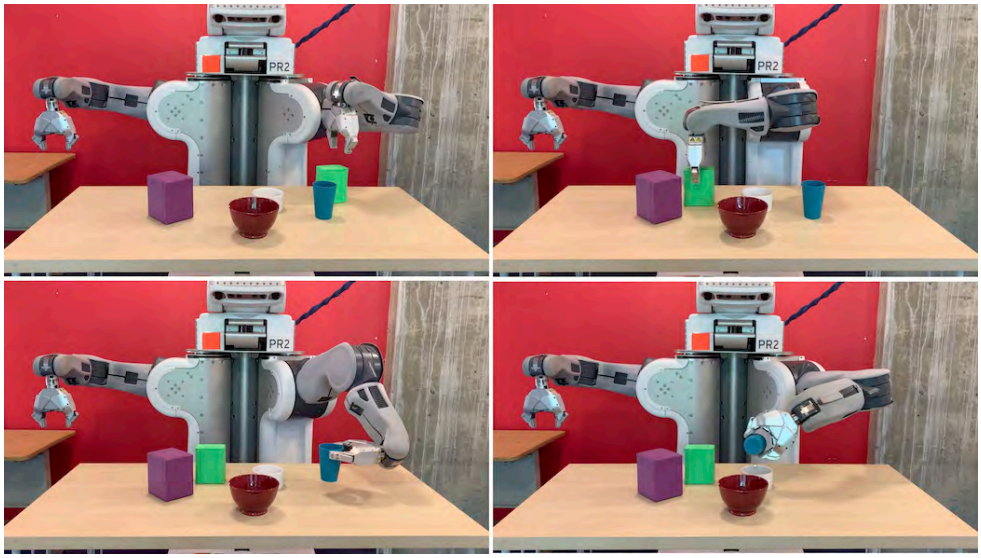

Figure 1: The specified goal is for the contents of the blue cup to end up in the white bowl. Because the green block obstructs reachable grasps for the blue cup, a TAMP algorithm automatically plans to relocate the green block before picking up the blue cup and pouring its contents into the white bowl. From *left-to-right* and *top-to-bottom*: the robot picking up the green block, the robot placing the green block, the robot picking up the blue cup, and the robot pouring the blue cup's contents into the white bowl (8).

planning methods to apply to entire world states, but this approach fails algorithmically. The entire world state, seen as a single kinematic system, is highly *under-actuated*, in the sense that from any configuration, most of the degrees of freedom cannot be changed at will. The robot can only change the position of an object by moving over and touching it.

It is critical to understand the underlying topology of these spaces in order to plan in them. Work by Alami *et al.* (9, 10), Branicky *et al.* (11, 12), and Hauser *et al.* (13, 14) observed that the configuration space of the world has important *modal* structure: depending on where the objects are placed and how they are grasped, the legal changes to the world are in a different *mode* or feasible submanifold of the full space. Furthermore, it is only possible to change modes by moving to an intersection of the feasible space of the current mode and a new one, which is in general an even lower-dimensional subspace. For these reasons, planning is best viewed as a *hybrid* discrete-continuous search problem, of selecting a finite sequence of discrete mode types (*e.g.*, which objects to pick and place), continuous mode parameters (such as the poses and grasps of the movable objects), and continuous motion paths within each mode to a configuration that is in the intersection with the subsequent mode.

The artificial intelligence (AI) community has addressed problems of planning in very large discrete domains (15). Their techniques derive leverage from *factoring*, a source of combinatorial structure in planning problems. Factoring is used to decompose the state space of the world into the Cartesian product of several subspaces, represented in terms of different state variables. Factoring enables compact representation of the *actions* that

can be performed on a state: these are generally described in terms of a small set of state variables that can be changed (while the others are held constant), as well as a condition on other variables that must be satisfied in order for the action to be executed. Furthermore, the AI planning community has developed a repertoire of very effective, domain-independent search algorithms that exploit this type of action representation (16, 17, 18).

Research in *task and motion planning* (TAMP) seeks to combine AI approaches to task planning and robotics approaches to motion planning. A critical requirement for generality in approaches to TAMP actually lies *between* discrete "high-level" task planning and continuous "low-level" motion planning: an intermediate level of selecting the real-valued mode parameters, such as how to grasp and where to place an object, which govern legal continuous motions of the system. This class of problems is computationally difficult in theory (19, 20) and requires algorithmic sophistication in practice.

1.1. Example

An essential component of TAMP problems is the interdependence of the motion-level and task-level aspects of the problem. Approaches that treat these independently, without considering their complex interplay, are unable to solve the general class of problems. Consider a problem in which the robot's goal is for a particular pot (named `A`) to be placed on one of the burners of the stove. If the planner ignores the geometric aspects, it might select a high-level plan "skeleton" of the form:

$$[\texttt{moveF}(q_0, \tau_1, q_1, p_0), \texttt{pick[A]}(q_1, p_0, g), \texttt{moveH[A]}(g, q_1, \tau_2, q_2), \texttt{place[A]}(q_2, p_1, g)]$$

where the `moveF` action involves robot movement when its hand is free, and the `moveH[A]` action involves robot movement when holding object `A`[1]. This plan skeleton has free parameters involving robot configurations (q_0, q_1, q_2), a grasp pose (g), placement poses (p_0, p_1), and paths (τ_1, τ_2). The skeleton imposes constraints on the choices of those values that will enable the plan to achieve the goal. Given this skeleton, it is now necessary to find values for all of these parameters that satisfy the constraints. It may be that there is no satisfying set of values; for instance, a kettle could be occupying the target burner, preventing any safe placement of the pot. In this case, a new skeleton is necessary: the robot will need to first move away the kettle and then place the pot on the stove. This example demonstrates a change in the high-level plan that is necessitated by the low-level geometry.

1.2. Scope

To keep the scope of this survey manageable, we limit the class of problems addressed, and discuss a variety of extensions in Section 4. In particular, we assume that: 1) actions are deterministic, 2) the state of the world is completely known, 3) the robot and every object in the environment is a kinematic assembly of rigid bodies with known shapes, 4) the robot is holonomic, and 5) the goal is specified as a set of requirements on the final robot configuration, object poses, and possibly other state variables such as the cooked state of a dish. This class of problems encompasses a number of problems studied in the robotics community: *pick-and-place planning* (21), *manipulation planning* (22), *navigation among movable obstacles* (NAMO) (23), and *rearrangement planning* (24). An important related

[1] See Figure 7 for a complete definition of these actions.

line of work uses *linear temporal logic* (LTL) to provide high-level specifications for TAMP problems with temporally extended goals (25, 26), but it is beyond the scope of this survey.

We begin with basic background in motion planning, multi-modal motion planning, and task planning (Section 2). Next, we draw from components of these fields in order to formalize TAMP in a manner that allows for many existing approaches to be studied (Section 3.1). We then describe a framework for understanding a broad class of TAMP algorithms in terms of combining (Section 3.3) a search over discrete plan structures with a search over continuous values satisfying constraints (Section 3.2) induced by the discrete structure. We conclude with a short discussion of a rich array of extensions and generalizations of this basic problem class and the approaches to solve them (Section 4).

2. BACKGROUND

TAMP rests on foundations in robot motion planning (Section 2.1), multi-modal motion planning (Section 2.2), and AI task planning (Section 2.3). In this section, we give a compact overview of each of these planning problem classes.

2.1. Motion planning

The problem of planning motions for a robot with d degrees of freedom can be framed as finding a trajectory for a point representing the robot's configuration through a d-dimensional configuration space. More formally, a motion-planning problem is specified by a configuration space $\mathcal{Q} \subset \mathbb{R}^d$, a constraint $F : \mathcal{Q} \to \{0, 1\}$, an initial configuration $q_0 \in \mathcal{Q}$, and a goal set of configurations $Q_* \subseteq \mathcal{Q}$. The *feasible* configuration space is a subset of $\mathcal{Q}$ that satisfies the constraint: $Q_F = \{q \in \mathcal{Q} \mid F(q) = 1\}$. The objective is to find a *continuous* path $\tau : [0, 1] \to \mathcal{Q}$ such that $\tau(0) = q_0$, $\tau(1) \in Q_*$, and $\forall \lambda \in [0, 1]\ \tau(\lambda) \in Q_F$. The simplest motion-planning problems involve free-space motion, in which the robot simply needs to move through space without colliding. Given a set of objects, defined by their shapes and poses in the world, the constraint $F(q)$ requires that the robot not collide with any object.

Motion planning is PSPACE-hard, but there are exact algorithms that leverage algebraic geometry to solve problems using only polynomial space (proving motion planning is PSPACE-complete) (27). Despite this, the two most widely used approaches are *sampling-based* motion planning (4, 5) and *trajectory optimization* (6, 7). Both classes of algorithms are useful in practice, but are not *complete* due to the fact that they cannot identify infeasible problems. Many sampling-based motion planning algorithms can however, under some robustness conditions, be shown to be *probabilistically complete*, meaning that the probability that they will fail to find a solution, if one exists, converges to zero as the running time increases. LaValle (28) provides a comprehensive overview of motion planning algorithms.

2.2. Multi-modal motion planning

Multi-modal motion planning (MMMP) extends the problem space of planning to include changing the state of other objects in the world (13, 29, 14, 30). To formalize MMMP problems, we need to model changes in the kinematics of the system, extend motion planning to handle constraints beyond collision avoidance, and integrate these components.

2.2.1. Kinematic graphs. One way to represent the geometric state of many environments is to encode the state variables collectively as a *kinematic graph* (28), which makes their

dependencies explicit. In a kinematic graph, vertices represent bodies and the robot's controllable joints, and edges represent attachments. Each edge has an associated relative transformation between the child body and parent body, which is a pose in SE(3). If each body is connected to at most one parent body and the graph is acyclic, this is a *kinematic tree*, for which the full state of the world can be derived from just the joint values of the robot q through *forward kinematics*. The attachments can be of several kinds. The most straightforward is a rigid attachment, which models an object resting stably on a surface or a robot grasping an object in a fixed grasp.

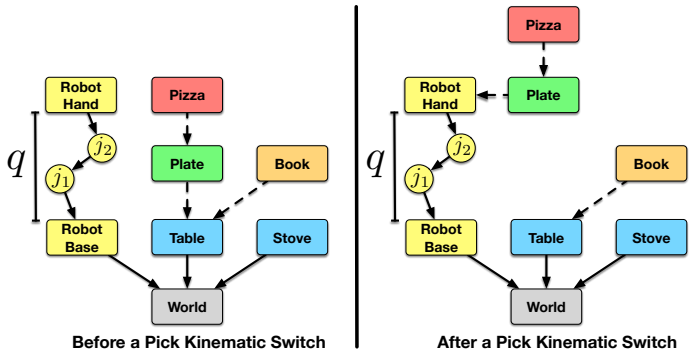

Figure 2: The change to a kinematic tree due to picking up the plate. Square nodes are bodies, and round nodes are robot joints. Lines encode attachments; solid ones are fixed, and dotted ones may be changed. The robot has two joints (j_1, j_2), whose current state is given by configuration q.

Figure 2 represents a kinematic tree for an example kitchen environment that contains a robot manipulator with two joints (j_1, j_2), a fixed `Table` and `Stove`, and movable objects `Plate`, `Pizza`, and `Book`. Initially, `Pizza` rests on the `Plate`, which itself rests on the `Table`. When the robot picks up the `Plate`, it also transitively picks up `Pizza`. This change in the kinematic graph is referred to as a *kinematic switch*, which is a type of *mode switch*. After the switch, as the manipulator moves, the poses of both the `Plate` and `Pizza` change with respect to the *world*; we move through this mode using the same actuators as before, but the feasible configuration space has changed.

2.2.2. Constrained motion planning. When the robot interacts with objects in the world, the effective configuration space is no longer the degrees of freedom of the robot: it corresponds to the state of the *whole* system; we denote this space $\mathcal{W}$. This state can be described by the discrete structure encoded in a kinematic graph, as well as continuous values of the transformations on the edges, which encode static relationships. However, these systems are generally *under-actuated*, meaning that they cannot be locally controlled in arbitrary directions, because we can only directly actuate the robot's degrees of freedom. Despite this, we can indirectly manipulate these objects by controlling the robot.

We begin by considering a simple "single-mode" problem in which the kinematic graph is fixed. Figure 3 (*left*) illustrates a robot gripper pulling a drawer where the gripper pose is fixed relative to the drawer. Although normally the gripper can translate generally in x, z, the drawer only has a single degree of freedom, denoted by joint j. The combined configuration space of the gripper and the drawer is a three-dimensional space with coordinates

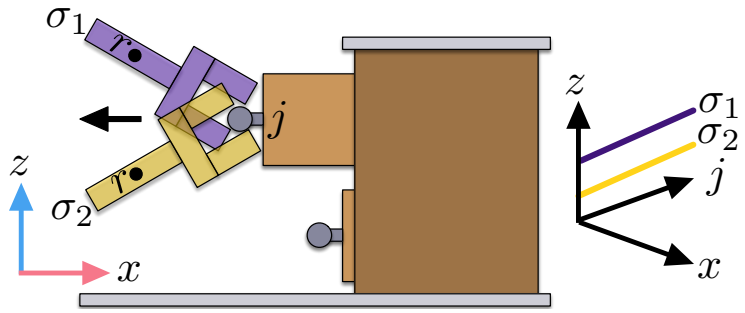

Figure 3: Constrained motion planning for a system in which a gripper pulls a drawer. The pose of the gripper relative to the drawer handle induces the 1D mode constraints σ_1, σ_2.

$\langle x, z, j \rangle$, but they are constrained by the need to keep the gripper attached to the drawer.

In this and many other manipulation problems, the constraint function $F(w)$, which now applies to the whole world configuration $w \in \mathcal{W}$, includes both the collision-free constraint and a kinematic constraint, which causes $\mathcal{W}_F$, the subspace of $\mathcal{W}$ for which F holds, to be lower dimensional than $\mathcal{W}$. As a result, sampling $\mathcal{W}$ randomly will have zero probability of producing a sample in $\mathcal{W}_F$, rendering standard sampling-based motion planning methods ineffective. This difficulty of sampling motivated the development of *constrained motion planners*, which explicitly take these constraints into account and plan within the low-dimensional space $\mathcal{W}_F$.

Dimensionality-reducing constraints are often expressed using a mode parameter σ, a fixed value that affects the constraint $F_\sigma(w)$. In general, σ is real-valued. Here, we illustrate the effect of two different choices of this value, σ_1 and σ_2. Each stipulates a different rigid attachment pose between the gripper and the drawer handle. Figure 3 (*right*) illustrates the combined configuration space and the feasible spaces $\mathcal{W}_{F_{\sigma_1}}$ and $\mathcal{W}_{F_{\sigma_2}}$, which are lines. The modes σ_1 and σ_2 allow motion of the gripper along different 1D lines in this 3D space, depending on the grasp of the drawer handle.

The most general approach for constrained motion planning defines sampling and connecting operations that *project* values onto the constraint surface. This is typically done by starting at a sampled point $\mathcal{W}$ and performing local descent on the constraint violation until convergence. Because this is a numeric optimization, the constraint will generally never be exactly satisfied, but the samples can get ϵ-close to the surface for any $\epsilon > 0$. Several approaches have provided probabilistically complete methods for constrained motion planning using projection (31, 32) and atlas-based techniques (33, 34, 35). See (36, 37) for a comprehensive survey of these techniques.

When the kinematic graph is a tree, the planning problem is much easier. The set of pairwise rigidity constraints specify all poses of objects relative either to the world frame (fixed objects) or to the robot (grasped objects). This collection of poses collectively constitutes a mode σ. We can sample full configurations for the system that exactly satisfy these constraints by simply sampling the robot's degrees of freedom q, and performing forward kinematics to derive the full configuration w.

2.2.3. Multi-modal motion-planning. Constrained motion planning provides a framework for reasoning about systems with many degrees of freedom, but few actuators. However,

it assumes that the constraints themselves remain constant, and therefore is not expressive enough to model multi-step manipulation problems in which the robot must make and break contact, changing the kinematic graph, and therefore the active constraints on its motions. In order to model such problems, we must allow the mode to undergo discrete changes (13, 29, 14, 30). The state of the system is $s = \langle w, \sigma \rangle$, where we can think of $w \in \mathcal{W}$ as the collective configurations of the robot and all other objects or mechanisms in the environment and σ as the additional mode information, indicating for example which objects are currently attached to which others. The control theory community analyzes reachability for a similar class of hybrid systems, except that they typically address problems with a finite set of modes but more complex continuous-time dynamics (38, 39, 40). For the most common cases of MMMP, we can refactor this representation, so that $s = \langle q, K \rangle$, where $q \in \mathcal{Q}$ is the robot configuration and K is a kinematic graph which contains the mode information and implies poses of all the bodies in the system, but we will make our general presentation in terms of $\langle w, \sigma \rangle$.

More formally, a MMMP problem consists of a finite set $\{\Sigma_1, \ldots, \Sigma_m\}$ of *mode families*, each of which has a real-valued parameter vector θ. Associated with each mode $\sigma = \Sigma(\theta)$ is a constraint function F_σ on full system configurations. At any given time, the system state $\langle w, \sigma \rangle$ is in a single mode σ, but whenever $w \in F_{\sigma'}$, the system may execute a *mode switch*, typically represented by a change to the kinematic tree, into mode σ'. The goal of an MMMP is typically a set of full system configurations $\mathcal{W}_*$, and a solution has the form $[\sigma_0, \tau_0, \sigma_1, \tau_1, \ldots, \sigma_k, \tau_k]$, where $s_0 = \langle w_0, \sigma_0 \rangle$ is the initial state of the system, τ_i is a trajectory in F_{σ_i}, $\tau_0(0) = w_0$, $\tau_i(0) = \tau_{i-1}(1)$ for $i \in \{1, ..., k\}$, and $\tau_k(1) \in \mathcal{W}_*$.

As an example, we model pick and place tasks in this framework. Modes in which the robot is not grasping any objects are *transit modes*, and modes in which the robot is holding an object are *transfer modes* (10, 9, 22). For a robot with a single gripper, there is a *transit* mode family for free motion and a *transfer* mode family for each object that it can grasp. In the transit mode family, the mode parameter is comprised of the fixed world poses of every movable object. In the transfer mode family for a particular object, the mode parameter contains the grasp pose as well as the fixed world poses of every other movable object. Thus, although the system can only operate according to a single mode at a time, the mode parameter is high-dimensional because it contains constraints involving every movable object. For interactions with cyclic kinematic graphs, such as manipulating a drawer or opening a door, a constrained motion-planner (Section 2.2.2) is generally required in order to plan within the mode.

A key challenge in multi-modal motion planning is identifying configurations that are in the intersection of the constraint sets for two modes and thus allow the system to switch between them. This intersection is often lower dimensional than the feasible space Q_{F_σ} of either mode. In a pick-and-place domain, in order to perform a kinematic switch between a transit and transfer mode, the robot's gripper must be in contact with the involved object at a particular pose. This requirement imposes 6 constraints on the robot's configuration, and as a result, the set of solutions is $(d - 6)$-dimensional. Fortunately, solutions can often be found using inverse kinematics (IK) either by projecting random samples into the constraint set using optimization (41) or by analytically solving for the solutions to a reparameterized set that captures its underlying dimensionality (42).

Figure 4 demonstrates a 1D robot (`R`) acting in the presence of a single movable object (`A`). The two plots visualize the 2D combined configuration space of the robot and the movable object. The left plot demonstrates the robot moving during a transit mode. The

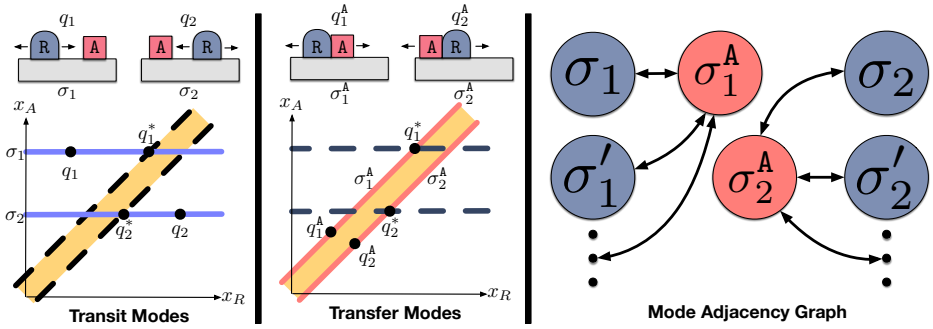

Figure 4: The feasible configuration space for two transit modes σ_1, σ_2 and two transfer modes $\sigma_1^{\mathtt{A}}, \sigma_2^{\mathtt{A}}$ modes. Mode switches between $\sigma_1 \leftrightarrow \sigma_2^{\mathtt{A}}$ occur at configuration q_1^*. Mode switches between $\sigma_2 \leftrightarrow \sigma_2^{\mathtt{A}}$ occur at configuration q_2^*.

two 1D blue lines indicate the space for which the system can change, which depends on the current mode σ_1 or σ_2. These modes correspond to different placements of the movable object, which remains constant. The yellow region corresponds to infeasible states where the robot and the object are in collision. Because the object can be placed anywhere on the interval, there are infinitely many possible transit modes. The center plot demonstrates the robot and object moving during a transfer mode. The robot can attach itself either to the left or right side of the object. As a result, there are two possible transfer modes $\sigma_1^{\mathtt{A}}, \sigma_2^{\mathtt{A}}$, indicated by the 1D red lines. The relative pose between the robot and object remains constant during a transfer mode. The robot can switch between transit and transfer modes at a zero-dimensional (point) intersection between both lines (q_1^*, q_2^*). The right figure visualizes legal mode transitions as a directed graph. The transit modes $\{\sigma_1, \sigma_1', ...\}$ correspond to the robot being on the left of the object, whereas the transit modes $\{\sigma_2, \sigma_2', ...\}$ correspond to the robot being on the right of the object. In order to switch to a new transit mode, the robot must first enter the appropriate transfer mode. Finally, note that the graph is disconnected because the robot is unable to move to the other side of the object.

2.3. Task planning

Within the AI community, there has been a long-standing focus on planning in discrete domains, generally with very large state spaces, but made tractable by using representations and algorithms that exploit underlying regularities in the structure of the domain. Ghallab *et al.* (15) provide a comprehensive discussion of task planning from the AI perspective, and Karpas and Magazzeni (43) survey task planning for robotics.

The simplest formalization of AI planning is to specify a set of states (state space) $\mathcal{S}$, a set of transitions $\mathcal{T} \subseteq \mathcal{S} \times \mathcal{S}$ that describe legal changes to the state, an initial state $s_0 \in \mathcal{S}$, and a set of goal states $S_* \subseteq \mathcal{S}$. Each directed transition $t = \langle s, s' \rangle \in \mathcal{T}$ moves the system from state s to state s'. The objective for a planner is to find a plan π, a sequence of transitions, that advances the initial state s_0 into a goal state $s_* \in S_*$. This problem can be reduced to a graph traversal problem, where the vertices are states and directed edges are transitions, and solved using standard graph-search algorithms. However, the state spaces considered are very large, so it is critical to use a functional representation of $\mathcal{T}$ to "reveal" states incrementally, for example by working forward from the initial state.

The first step toward compact representations and efficient algorithms is to *factor* the state representation into a collection of state variables. More formally, states can be represented using a set of variables $\mathcal{V} = \{1, ..., m\}$, each of which has a finite domain $\mathcal{X}_v$. States are assignments of values $x_v \in \mathcal{X}_v$ for variables $v \in \mathcal{V}$. This induces a state space $\mathcal{S} = \mathcal{X}_1 \times ... \times \mathcal{X}_m$ that is the Cartesian product of each variable's domain. Consider a variation on the example in Figure 2, involving a single robot, a movable pizza, and a movable book. Each state specifies the locations of the robot, the pizza and the book, and the object that the robot is holding (or `None`), as well as whether or not the pizza has been cooked. The set of possible locations are: `Box`, `Plate`, `Table` and `Oven`. The robot can move between any pair of locations, pick up an object at the robot's current location if it is not holding anything, and place an object at the robot's current location. We can describe a state as an assignment of values to the variables `atRob` (4 possible values), `at[Pizza]` (5 domain values), `at[Book]` (5 domain values), `holding` (3 domain values), and `cooked[Pizza]` (2 domain values). A state in this domain can then be defined as

{`atRob`=`Plate`, `holding`=`None`, `at[Book]`=`Table`, `at[Pizza]`=`Box`, `cooked[Pizza]`=`False`}.

Although in total the variables have 19 possible values, there are 600 possible states resulting from the possible combinations of variable values. Generally, the size of the state space grows exponentially in the number of variables.

Next, we need to encode the set of transitions compactly. In many domains, due to locality of effect or other underlying domain properties, transitions change the value of only a small number of the state variables at a time, which allows us to describe large sets of transitions compactly using a single *action* that encodes the *difference* between the two states. These changes can be described by a set of *effects* `eff:` $\{v_1 \leftarrow c_1, ..., v_k \leftarrow c_k\}$ that list the variables that are modified $(v_1, ..., v_k)$ and their resulting values $(c_1, ..., c_k)$. This set of effects describes a large set of state-pairs in the transition: one corresponding to each possible assignment to the values of the unchanged variables.

```
moveF[loc1,loc2]
 pre: atRob = loc1
 eff: atRob ← loc2
moveH[loc1,loc2]
 pre: atRob = loc1, holding = obj
 eff: atRob ← loc2
pick[obj,loc]
 pre: atRob = loc, holding = None, at[obj] = loc
 eff: at[obj] ← None, holding ← obj
place[obj,loc]
 pre: atRob = loc, holding = obj
 eff: at[obj] ← loc, holding ← None
cook[obj]
 pre: at[obj] = Stove
 eff: cooked[obj] ← True
```

Figure 5: A template specification of `moveF`, `moveH`, `pick`, `place`, and `cook` actions.

Another structural property of many domains, which can be used to more compactly express legal transitions, is that each action may correctly be executed in only certain

states. For example, a `pick` action cannot be performed if the robot is already holding an object. We can express this by specifying, for each action, a set of *preconditions* `pre` : $\{x_{v_1}{=}c_1, ..., x_{v_k}{=}c_k\}$ that describe the set of states in which that action can be executed in terms of values of some of the state variables.

One final important structural property is an "object-centric" abstraction: in most problems, the state variables correspond to properties of objects in the domain (*e.g.*, the location or color of a particular cup) or relations among them (*e.g.*, whether a particular cup is inside a particular box). We can take describe the possible actions of a domain generically, via templates that are parameterized by a choice particular objects that are present in a domain instance. This form of abstraction allows the size of the domain description to be independent of the *number* of state variables in the domain.

We illustrate the basic principles of task planning via an example in Figure 5, which specifies the preconditions and effects of the `moveF`, `moveH`, `pick`, `place`, and `cook` actions. This specification needs to be coupled with a listing of the *actual* entities in any domain instance, such as the names of objects (`Pizza` and `Book`) and locations (`Box`, `Plate`, `Oven`, and `Table`), to yield a complete transition-system specification. Then, the state variables are `holding`, `atRob`, along with `cooked[obj]` and `at[obj]` for each actual object name `obj`. Similarly, the actual possible actions are generated by substituting all combinations of object and location constants in for the template variables. For example, with two objects and four locations, there are 8 instances of the `place` action.

To clarify the use of template variables, note the `pick` action description: it is a *template* describing a finite number of action instances, one for each discrete value of `obj` and `loc`. But note that these two variables play different roles. As the number of possible values of `obj` increases, the *dimensionality* of the state of the problem (characterized by the *number* of state variables) increases; as the number of possible values of `loc` increases, the *domains* of the `at[obj]` variables increases but the number of variables does not.

The final component of a planning problem is a description of the set of goal states, which has the same form as an action precondition, as a conjunction of values of some state variables, where all unmentioned state variables may have any arbitrary value. For example, the following goal description encodes the entire set of states in which the pizza is cooked and on the plate: {`at[Pizza]`=`Plate`, `cooked[Pizza]`=`True`}. The solution to a task-planning problem is a sequence of action instances $a_1, \ldots, a_k$, that induces a state sequence $s_0, \ldots, s_k$, where each s_i is a state expressed as an assignment of values to state variables, s_0 is the initial state of the planning problem, s_i satisfies the preconditions of a_{i+1}, s_{i+1} is the result of executing a_i in s_i, and s_k satisfies the goal conditions.

To finish our example, let the initial state be

$$s_0 = \{\texttt{holding=None}, \texttt{atRob=Table}, \texttt{at[Pizza]=Box}, \texttt{at[Book]=Table}, \texttt{cooked[Pizza]=False}\}.$$

Solving this task requires first placing the pizza on the oven to cook it and then relocating it to the plate:

$$\pi = [\texttt{moveF[Table,Box]}, \texttt{pick[Pizza]}, \texttt{moveH[Box,Oven]}, \texttt{place[Pizza]},$$
$$\texttt{cook[Pizza]}, \texttt{pick[Pizza]}, \texttt{moveH[Oven,Plate]}, \texttt{place[Pizza]}].$$

One focus of AI planning has been to define languages for specifying planning problems. The one shown in Figure 5 is similar to a lifted version of *simplified action specification* (SAS+) (44). The most widely-used formalism is *planning domain definition language*

(PDDL) (45), which can be seen as a transition system where state variables are Boolean facts. The AI planning community has developed *domain-independent* algorithms that can operate on any problem written in a planning language, without any additional information about the problem. A factored planning representation enables efficient algorithms for solving *relaxed problems*, simplified versions of the original problem, and using their solutions to estimate the distance to a goal state (18).

Finally, there are several extensions to the basic task planning formalism (46, 47, 48) that are relevant to TAMP. One of these is *numeric planning*, which involves planning with real-valued variables such as time, fuel, or battery charge. Recent approaches support planning with convex dynamics (49) and non-convex dynamics by discretizing time (50). Although these methods have many use-cases, they currently cannot be directly applied to most TAMP problems because they assume the set of actions is finite.

3. TASK AND MOTION PLANNING

To find solutions to TAMP problems, we need to integrate aspects of motion planing, multi-modal motion planning, and task planning. In this section, we introduce a framework for describing TAMP problems and algorithms that allows us to describe most of the broad range of existing methods within a unified framework, and which we hope elucidates modeling and algorithmic trade-offs among them. We begin by providing a formalism for describing TAMP problems, then characterize solution methods in terms of their strategies for sequencing actions, for selecting their continuous parameters, and for integrating these methods.

3.1. TAMP problem description

Informally, TAMP problems use compact representational strategies from task planning to describe and extend a class of MMMP problems. TAMP is an extension of MMMP in that there may be additional state variables that are not geometric or kinematic, such as whether the lights are on or the pizza is cooked. We begin by articulating a generic MMMP, using an extension of a task-planning formulation, in Figure 6. There are two extensions of the task-planning formalism visible here. First, there are *continuous* action parameters. Second, in addition to preconditions and effects we have a new type of clause, called `con` for *constraint*. It is a set of constraints that all must hold true among the continuous parameters of the action in order for it to be a legal specification of a transition of the system.

`moveWithin[i]`(θ, w, τ, w')
con: $\tau(0) = w,\ \tau(1) = w',\ \left(\forall t \in [0,1]\ F_{\Sigma_i(\theta)}(\tau(t))\right)$
pre: `mode` $= \Sigma_i(\theta)$, `conf` $= w$
eff: `conf` $\leftarrow w'$
`switchModes[i,j]`(w, θ, θ')
con: $F_{\Sigma_i(\theta_1)}(w),\ F_{\Sigma_j(\theta_2)}(w)$
pre: `mode` $= \Sigma_i(\theta_1)$, `conf` $= w$
eff: `mode` $\leftarrow \Sigma_j(\theta_2)$

Figure 6: A formalization of MMMP in the style of task planning. There is a `moveWithin` action for each mode family Σ_i and a `switchModes` action for each mode family pair Σ_i, Σ_j.

This formulation does not extend to the basic formulation of MMMP, but it provides a clear articulation of the overall system dynamics. In a domain with a large number of

objects, there will be a large number of mode families, each of which requires specifying a constraint on a very high-dimensional world configuration space. What TAMP adds is the ability to "unpack" the entities in the problem description into sub-parts that are simpler to describe and that reveal substructure in the problem that enables algorithmic insights. We will illustrate this process in a TAMP generalization of the "cooking" domain from Section 2.3 using one particular formalization style, shown in Figure 7.

Consider an example which has five movable objects, `A` through `E`. We decompose the system configuration w into state variables `atRob`, `holding`, `at[A]`, `at[B]`, `at[C]`, `at[D]`, and `at[E]`. The discrete state variable `holding` can take values ranging over {`None, A, B, C, D, E`} and specifies the current mode family Σ. The variable `atRob` is now a robot configuration, and `at[obj]` is the pose of object `obj` relative to either the world coordinate frame (when `holding` $\neq$ `obj`) or the robot hand coordinate frame (when `holding` $=$ `obj`).

The `moveF` (move while the gripper is free) and `moveH` (move while the gripper is holding) actions describe transit and transfer motion within modes. The `pick` action corresponds to a switch from a transit mode to a transfer mode, while the `place` action corresponds to a switch from a transfer mode to a transit mode.

The sparsity of effect of planning action descriptions is a good match for articulating which state variables are changed (and, implicitly, which ones stay the same). We can see that, in each action description, the `eff:` clause indicates just the variables that change. When the preconditions involve discrete constant values (such as `None`), they are being used to specify the mode family of the initial state of the transition. The advantage of being able to use templates is apparent: the `moveH` action has a template parameter `obj`, meaning that there is a mode family for each object being held.

`moveF`$(q, \tau, q', p^{\mathtt{A}}, \ldots, p^{\mathtt{E}})$
con: `Motion`(q, τ, q'), `CFreeW`(τ), `CFreeA`$(p^{\mathtt{A}}, \tau)$, ..., `CFreeE`$(p^{\mathtt{E}}, \tau)$
pre: `holding` $=$ `None`, `atRob` $= q$, `atA` $= p^{\mathtt{A}}$, ..., `atE` $= p^{\mathtt{E}}$
eff: `atRob` $\leftarrow q'$

`moveH[obj]`$(g, q, \tau, q', p^{\mathtt{A}}, \ldots, p^{\mathtt{E}})$
con: `Motion`(q, τ, q'), `CFreeW[obj]`(g, τ),
`CFreeA[obj]`$(p^{\mathtt{A}}, g, \tau)$, ..., `CFreeE[obj]`$(p^{\mathtt{E}}, g, \tau)$
pre: `holding` $=$ `obj`, `at[obj]` $= g$, `atRob` $= q$, `atA` $= p^{\mathtt{A}}$, ..., `atE` $= p^{\mathtt{E}}$
eff: `atRob` $\leftarrow q'$

`pick[obj]`(q, p, g)
con: `Stable[obj]`(p), `Grasp[obj]`(g), `Kin[obj]`(q, p, g)
pre: `holding` $=$ `None`, `atRob` $= q$, `at[obj]` $= p$
eff: `holding` $\leftarrow$ `obj`, `at[obj]` $\leftarrow g$

`place[obj]`(q, p, g)
con: `Stable[obj]`(p), `Grasp[obj]`(g), `Kin[obj]`(q, p, g)
pre: `holding` $=$ `obj`, `atRob` $= q$, `at[obj]` $= g$
eff: `holding` $\leftarrow$ `None`, `at[obj]` $\leftarrow p$

`cook[obj]`(p)
con: `Stable[obj]`(p), `OnStove[obj]`(p)
pre: `at[obj]` $= p$
eff: `cooked[obj]` $\leftarrow$ `True`

Figure 7: One formalization of TAMP for an environment that contains the movable objects `A`, `B`, `C`, `D`, and `E`. Actions now have real-valued parameters and constraints on these parameters.

Just as we have decomposed the configuration and the mode, we can decompose con-

straints, expressing them as conjunctions of constraints with smaller arity. For example, the `pick` action has the constraint `Kin[obj]`(q, p, g). For any particular value of `obj`, representing an actual object in the domain, this represents a kinematic constraint, saying that if the robot is in configuration q and holding object `obj` in grasp g, then `obj` will be at pose p. In `moveF` and `moveH`, the `Motion`(q, τ, q') constraint specifies the relationship between a trajectory τ and two robot configurations, asserting that $\tau(0) = q$, $\tau(1) = q'$, and τ is continuous. Notice that the trajectory τ appears neither in the preconditions nor effects of these actions; they are auxiliary parameters that describe motion *within* the modes. The `Stable[obj]`(p) constraint requires that p be a pose representing a stable placement for object `obj` on a static object in the world. Similarly, the `OnStove[obj]`(p) constraint requires that p be a stable placement where `obj` is specifically on a stove. The `Grasp[obj]`(g) constraint defines stable grasp poses (transforms between the hand frame and object frame) g for object `obj`. This set may be finite if there only a few known grasps but could be uncountably infinite, in general. The collision-free constraint `CFreeA[obj]`(p, g, τ) asserts that if object `A` is at pose p, the robot is holding `obj` in grasp g, and it executes trajectory τ, no collision will occur. The constraint `CFreeW[obj]`(g, τ) is defined similarly except that it involves the fixed objects in the world (indicated by the abbreviation `W`). Finally, although not pictured, because the $p^{\texttt{A}}, \ldots, p^{\texttt{E}}$ parameters in the `moveF` and `moveH` actions are each only mentioned in a single constraint and precondition, they can be compiled away using state constraints (51) or inference rules (axioms) (52), resulting in these actions templates being independent of the number of objects in the problem instance.

3.1.1. The form of solutions. In preparation for studying algorithms for solving TAMP problems, it is useful to examine the form of a solution, which is a finite sequence of action instances $\pi = [a_1, \ldots, a_k]$, where each a_i includes assigned values for all parameters that satisfy that action's constraints. These actions induce a state sequence $[s_0, s_1, \ldots, s_k]$, where each s_i is a state expressed as an assignment of values to state variables, s_0 is the initial state of the problem, s_{i-1} satisfies the preconditions of a_i, s_i is the result of executing a_i in s_{i-1}, and s_k satisfies the goal conditions. Selecting the action templates and values for the template variables specifies the *form* of a solution, which we call a *plan skeleton*. If the skeleton is fixed, the set of variables for which values must be selected is determined, and the problem that remains is one of selecting those values so that the constraints of the actions in the skeleton are satisfied.

Consider a TAMP problem with a single movable object `A`. Suppose the initial state is $s_0 = \{\texttt{atRob=}\mathbf{q_0}, \texttt{at[A]=}\mathbf{p_0}, \texttt{holding=None}, \texttt{cooked[A]=False}\}$, where the bold mathematical symbols $\mathbf{q_0}, \mathbf{p_0}$ are real-valued constants. The set of goal states can be defined using conditions and constraints, such as `cooked`A`=True`. One possible plan skeleton is:

$$\begin{aligned}\pi = [&\texttt{moveF}(\mathbf{q_0}, \tau_1, q_1), \texttt{pick[A]}(q_1, \mathbf{p_0}, g_2), \\ &\texttt{moveH[A]}(q_1, \tau_3, q_3), \texttt{place[A]}(q_3, p_4, g_2), \texttt{cook[A]}(p_4)].\end{aligned} \tag{1}$$

where $q_1, \tau_1, g_2, q_2, \tau_3, p_4$ are the free parameters. Plan skeletons can be visualized graphically by enumerating the sequence of $|\pi| + 1$ values of each state variable, as well as motion parameters τ_1, τ_3, and associating the constraints of the ith action with the appropriate $i - 1$ and i state variables. Figure 8 illustrates this plan skeleton in the form of a dynamic factor graph (53). Round nodes represent state variables, and rectangular nodes represent constraints. Gray nodes have constant values, and colored nodes represent variables. Each vertical column corresponds to a state; the actions in the skeleton, which are responsible

for the state changes are shown between the state columns, at the top. Multiple nodes of the same color represent a single variable that is constrained to maintain its current value across multiple steps of the plan. Each constraint is connected to the variables it constrains. Any assignment of values to the variables that satisfies all the constraints "fills out" the skeleton into a complete legal plan that is guaranteed to achieve the goal. However, it may be the case that no satisfying assignment exists.

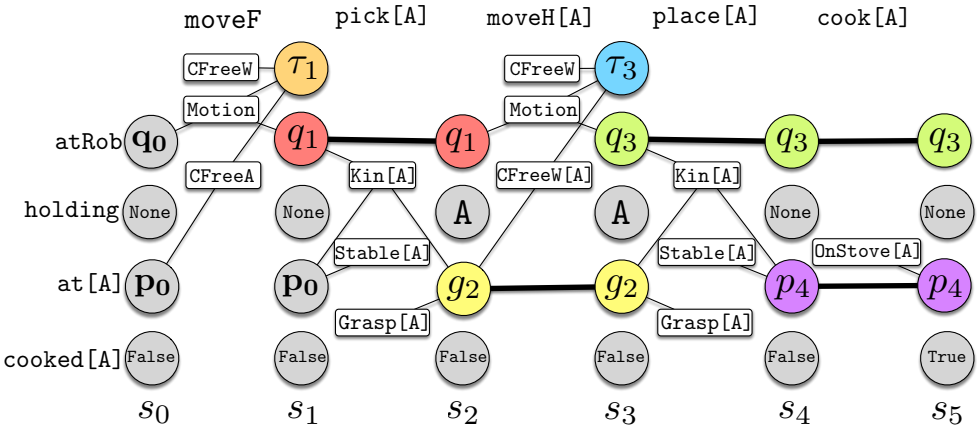

Figure 8: The plan skeleton from Equation (1). Grey values are constants. Thick black lines display equality constraints that persist over time.

3.2. Hybrid constraint satisfaction

Finding an assignment of values to the parameters of a plan skeleton that satisfy the associated constraints is a *hybrid constraint satisfaction problem* (H-CSP). Although many parameters are inherently continuous, some may have discrete domains. For example, there might be a finite set of stable resting surfaces for a particular object. Figure 9 compresses the plan skeleton in Figure 8 into a *constraint network*, a bipartite graph from parameters to constraints, by removing redundant constraints, constants, and parameters (54). Although TAMP is decidable via computational-geometry algorithms, just as in motion planning, most practical approaches use optimization or sampling to solve the underlying H-CSPs. Another dimension of variability in solution approaches is whether the method attempts to satisfy the entire constraint set at once or not: methods vary dramatically in their high-level control structure for handling the search over skeletons and parameter values, and make different demands on constraint satisfaction methods.

3.2.1. Joint Satisfaction. The most straightforward strategy for approaching an H-CSP is to reduce it to a constrained mathematical program and solve for values for all the free parameters at once. Although there is a vast literature on mathematical programming, solving programs corresponding to TAMP H-CSPs is often very difficult due to high dimensionality in continuous parameter space, the inclusion of discrete parameters, and the non-convexity of the constraints. There is no efficient, general solution method for these mathematical programs. There are, nonetheless, some approaches of practical value.

When all decision variables are real-valued, a common solution strategy is to minimize

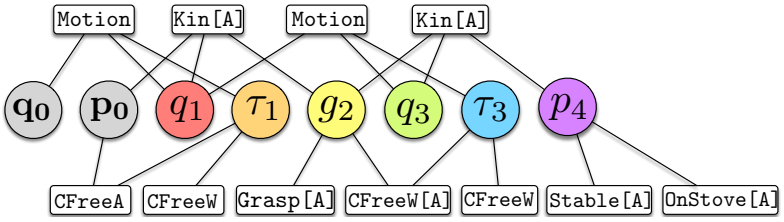

Figure 9: A simplification of the constraint network in Figure 8.

an objective function, which incorporates both the *hard* constraint violation and any *soft* action cost penalties, using local-descent methods, though these are guaranteed to reach only a *local* optimum of the objective function, which may not satisfy the constraints. Equation (2) displays a mathematical program corresponding to the constraint network in Figure 9. The trajectories τ_1, τ_3 are approximated as a sequence of robot configurations $\tau[0], \tau[1], ..., \tau[T]$ where T is a hyperparameter. Each constraint is associated with a real-valued (and often once or twice differentiable) function, which is expressed either in an equality ($g(\ldots)$) or inequality ($h(\ldots)$) constraint for the mathematical program. Although it is not a focus of this survey, optimization can also fluidly incorporate action costs, enabling it to identify a solution that is not only feasible but also low-cost. For example, Equation (2) minimizes the combined cost of moving through a `moveF` mode ($f_{\texttt{moveF}}(...)$) and a `moveH[A]` ($f_{\texttt{moveH[A]}}(...)$) mode, each of which are sums of a function defined on adjacent configurations that comprise trajectory parameter τ_1 or τ_3. More generally, *mixed-integer programming* (MIP) techniques are required. One prominent algorithm for solving MIPs is branch-and-bound, which performs a discrete search over assignments to the integer variables; then, conditioned on an assignment for each integer variable, the resulting mathematical program is real-valued and can be addressed by descent.

$$
\begin{aligned}
\underset{q_1,\tau_1,g_2,\tau_3,q_3,p_4}{\text{minimize}} \quad & \textstyle\sum_{t=1}^{T} f_{\texttt{moveF}}(\tau_1[t], \tau_1[t-1]) + \sum_{t=1}^{T} f_{\texttt{moveH[A]}}(g_1, \tau_3[t], \tau_3[t-1]) \\
\text{subject to} \quad & g_{\texttt{Grasp[A]}}(g_1) = 0,\ g_{\texttt{Stable[A]}}(p_4) = 0,\ h_{\texttt{OnStove[A]}}(p_4) \leq 0 \\
& g_{\texttt{Kin[A]}}(q_1, \mathbf{p_0}, g_2) = 0,\ g_{\texttt{Kin[A]}}(q_3, p_4, g_1) = 0 \\
& h_{\texttt{Motion}}(\tau_1[t], \tau_1[t-1]) \leq 0,\ h_{\texttt{Motion}}(\tau_3[t], \tau_3[t-1]) \leq 0 && \text{for } t \in [T] \\
& h_{\texttt{CFreeW}}(\tau_1[t]) \leq 0,\ h_{\texttt{CFreeA}}(\mathbf{p_0}, \tau_1[t]) \leq 0 && \text{for } t \in [T] \\
& h_{\texttt{CFreeW}}(\tau_3[t]) \leq 0,\ h_{\texttt{CFreeW[A]}}(g_1, \tau_3[t]) \leq 0 && \text{for } t \in [T] \\
& \tau_1[0] = \mathbf{q_0},\ \tau_1[T] = \tau_3[0] = q_1,\ \tau_3[T] = q_3
\end{aligned}
\tag{2}
$$

3.2.2. Individual Satisfaction. An alternative approach to solving H-CSPs is to generate small groups of parameter values that satisfy a single constraint or a small set of constraints, and combine them. A *sampler* takes one or more constraints and generates a sequence of assignments of values to the free parameters, where each assignment that is generated is guaranteed to satisfy the constraints.

A challenge when designing samplers is dealing with constraints whose set of satisfying values has lower dimension than combined domains of the free parameters. For example, the `Stable[obj]`(p) constraint requires object `obj` to rest perpendicular to a 2D plane within a 3D pose space, so this constraint lies in SE(2) despite the set of object poses being in SE(3). The rejection-sampling strategy of sampling at random from a bounded region of SE(3) will

have zero probability of producing a value satisfying this constraint. However, samples can be produced by directly sampling `Stable` rather than SE(3). Low-dimensional constraints remain problematic when attempting to produce values that also satisfy other constraints. For example, consider solving for values of q, p, g that satisfy both `Stable[obj]`(p) and `Kin[obj]`(q, p, g). Here, the difficulty is finding a pose p that satisfies `Stable` while also admitting values of q, g that satisfy `Kin`. One solution is to explicitly design samplers that operate on larger collections of constraints; this approach generally reduces to the joint satisfaction approach (Section 3.2.2).

Alternatively, one can design *conditional samplers* that take in *input* values for some of the parameters in the constraint(s) and produce satisfying *output* values for the rest of the parameters. Intuitively, these samplers consume values already known to satisfy some constraints and find completing values that are compatible for additional constraints. In the above example, a conditional sampler for `Kin[obj]` that takes in p, g as inputs can consume a placement pose sampled by `Stable[obj]` and produce configurations q through finding inverse kinematics (IK) solutions. In the event that no IK solution exists, the conditional sampler returns an empty sequence, effectively rejecting the input values. Boolean tests for a constraint can also be represented within this framework as degenerate "samplers" that perform a check on the input values but do not generate any output values. For example, the collision-free constraints `CFreeW`, `CFreeA`, and `CFreeW[A]` can be evaluated by querying a collision checker. In some applications, it may be beneficial to specify several conditional samplers for an individual constraint, which represent different partitions into input and output parameters. For example, an alternative sampler for `Kin[obj]` takes in q, g and performs forward kinematics to produce a pose for `obj` that satisfies the constraint.

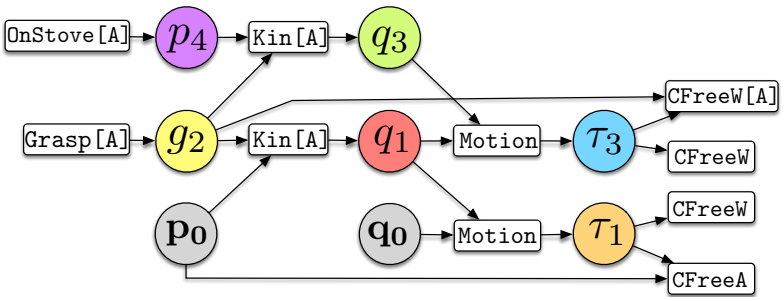

Figure 10: A sampling network for the constraint network in Figure 9.

More generally, several conditional samplers can be *composed* to form a *sampling network* (55), a directed acyclic graph defined on free parameters and conditional samplers. A directed edge from a parameter to a sampler indicates that the parameter is an input to the sampler. A directed edge from a sampler to a parameter indicates that the parameter is an output of the sampler. Each parameter is required to be the output of exactly one sampler. This process is similar in spirit to converting a factor graph (constraint network) into a directed acyclic Bayesian network (53). Figure 10 gives an example sampling network for the factor graph in Figure 9.

3.2.3. Comparison. There is a trade-off between satisfying constraints individually versus jointly. Individual satisfaction allows particular constraint types to be addressed using a special-purpose procedure, which is well equipped for that constraint, and provides a

framework for modularly combining them. For example, efficient algorithms for inverse kinematics and motion planning can be used to respectively generate robot configurations and trajectories. Often, values generated in an attempt to satisfy one H-CSP can be reused in other, related H-CSPs. In fact, values can even be usefully generated without a particular H-CSP in mind as shown in Section 3.3.2.

When jointly solving the complete set of constraints for a plan skeleton, only a single solution is required because, by construction, it has satisfied all relevant constraints. In comparison, when constructing samplers and conditional samplers, it is important that they, in the limit as the number of samples goes to infinity, cover the complete space of feasible solutions, because some samples may be ruled out by other constraints in the problem. Another advantage of joint satisfaction is that constraints on one parameter can transitively influence the selection of values for other parameters, directing the search. Many methods for joint satisfaction requires the constraints to be made available in analytic form, enabling fast and accurate computation of derivatives used in descent methods. However, some constraints, such as collision constraints, are difficult to define in a differentiable form. In such cases, sampling, which only requires black-box access to the constraint for use in rejection sampling, can be a more effective strategy, although its success is strongly dependent on the volume of solutions within the sampled space.

Finally, although we contrast these techniques, one can integrate both strategies. For example, an algorithm could use individual sampling to generate values that satisfy `Stable[A]`, `Grasp[A]`, and `Kin[A]` but use joint satisfaction to solve for trajectories τ that satisfy `Motion`, `CFreeW`, `CFreeA`, and `CFreeW[A]`.

3.3. Combining action sequence and parameter search

We now have the tools to search for action sequences (Section 2.3) and to solve H-CSPs (Section 3.2). In this section, we discuss strategies for combining them into integrated TAMP algorithms. We would like to order the decision-making in a way that minimizes the overall runtime of the algorithm, for a problem distribution. There are several intuitive principles for organizing the search, which are sometimes in conflict with one another. One way to reduce search effort is to prune infeasible decision branches as quickly as possible (which is sometimes called failing fast (56)). We would also prefer to postpone expensive computations until most of the rest of a potential solution is found. For example, in many manipulation applications, collision checking is expensive, due to the geometric complexity of 3D meshes and the need to check at a fine resolution to ensure safety, so we might wish to *lazily* postpone this operation (57, 58). At the same time, we would like to balance the computational effort spent on each component, for example, by not spending too much time trying to satisfy the H-CSP associated with a single skeleton or even single constraint, in case it is unsatisfiable. Additionally, information gained in one branch of a high-level search, such as the solution or infeasibility of a subproblem, can often be re-used to make another branch of the search more efficient.

We begin by focusing on the overall control flow of TAMP algorithms, which determines the relative ordering of action-sequencing and H-CSP (sub) problem solution. There are three predominant classes of strategies: *sequence before satisfy*, in which we find whole plan skeletons and then try to satisfy all of their constraints; *satisfy before sequence*, in which we find sets of satisfying assignments for individual constraints and attempt to assemble actions that use those values into complete plans; and *interleaved*, in which actions are added to the

plan and additional constraints are satisfied incrementally. We conclude by addressing an important aspect of making these approaches efficient, which is to take advantage of previous subproblem assignments or failures, in order to avoid re-addressing related subproblems. Figure 11 illustrates the first two classes of TAMP strategies as flowcharts.

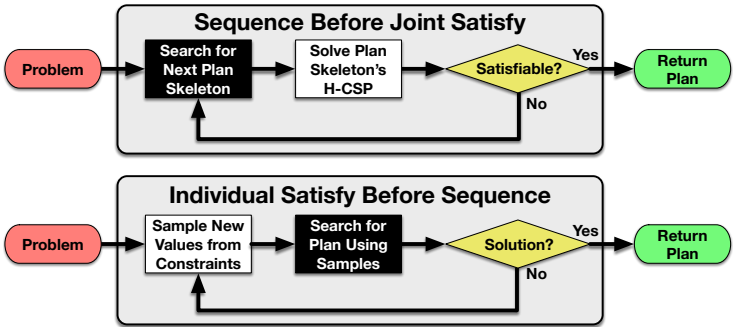

Figure 11: Flowcharts for two representative TAMP algorithms. *Top*: an algorithm that iteratively searches in the space of unbound plans and jointly satisfies the set of constraints (Section 3.3.1). *Bottom*: an algorithm that iteratively performs individual sampling before searching in the space of fully-bound plans (Section 3.3.2).

Throughout the discussion of control structures, it is important to remember that sampling and optimization techniques are typically only semi-complete, in that they are not able to certify that a problem instance is infeasible—they simply fail to find a solution in the time available to them. Even for feasible H-CSPs these algorithms may still need to be run for an extremely long time if, for example, a feasible problem only admits a tiny volume of solutions. Handling this is complicated by the fact that we might be forced to consider a possibly unbounded number of H-CSPs simultaneously. To simplify the discussion, we can think about this process happening *non-deterministically*, where intuitively a separate thread is created for each H-CSP. We can always simulate this behavior in a single process by appropriately revisiting the threads, making sure that none are starved.

3.3.1. Sequencing first. The earliest algorithms for TAMP committed to a strict *hierarchy* of first finding an action sequence, and then finding continuous parameter values. For example, Shakey (2) performed STRIPS planning over high-level abstract actions, such as which room to move to, and then planned low-level motions that realized the high-level plan, with no mechanism for finding an alternative high-level plan if the lower-level motions were not possible. So, Shakey aggressively assumed that problems satisfy the *downward refinement* property. More formally, a two-level hierarchy satisfies the downward refinement property if *every* solution to the high level can be refined into a solution at the low level (59). When this property holds, problems can be completely disentangled into separate task planning and motion planning problems, so an algorithm that determines a plan skeleton based on values of discrete template arguments strictly before solving the associated constraint-satisfaction problem is complete. In TAMP problems in practice, downward refinement rarely holds. As soon as geometric or kinematic considerations make some high-level plans infeasible (because, for example, three objects do not actually fit into the box we planned to put them in, or because the grasp needed to remove an object from a shelf will not work

to place it on the stove and there is no surface available to use for regrasping), then we cannot inflexibly commit to any abstract plan without knowing its geometric and kinematic feasibility.

However, even when downward refinement does not hold, a top-down problem decomposition can be very effective, as long as there is a mechanism for "backtracking" and trying alternative high-level plans when the lower-level solver fails (60, 61, 55). Figure 11 (*top*) illustrates this approach, in which there is an outer loop representing a search over legal plan skeletons; for each plan skeleton, we attempt to solve the associated H-CSP and if we succeed, we return the complete solution, otherwise, we return to the outer loop and try another skeleton. In many everyday TAMP applications, action sequencing is relatively inexpensive, making it advantageous to find a plausible action sequence before satisfying constraints. Furthermore, solving H-CSPs can be computationally expensive, so by only attempting to solve H-CSPs that correspond to viable plan skeletons, we can potentially save substantial computation time.

3.3.2. Satisfaction first. An alternative strategy is motivated by the fact that task planning in finite domains can often be very efficient in even very large problem instances, and therefore seeks to reduce the hybrid problem of TAMP to one or more discretized planning problems by generating values of continuous quantities, such as poses and configurations, and computing in advance which constraints they satisfy (22, 14, 21, 55). For example, one might sample, for an environment with some fixed support surfaces, a set of values p_i such that `Stable[A]`(p_i) holds. Approaches that perform satisfaction first almost always use individual satisfaction (Section 3.2.2), which is typically implemented using sampling, because they aim to generate values that are useful for a variety of plan skeletons. A single round of sampling will in general not suffice. When the discrete planning problem given a particular set of values is infeasible, it is necessary to generate more samples and try again, as illustrated in Figure 11 (*bottom*).

Satisfying before sequencing is advantageous when the computational effort of repeatedly sequencing and failing to satisfy the associated H-CSP outweighs the computational effort of eagerly generating values that satisfy constraints upfront. This often is the case when one or more of the following are true: 1) sampling is efficient and does not result in a combinatorial explosion of sampled values, 2) each discrete action sequencing search has non-negligible overhead, and 3) sampled values are unlikely to satisfy critical constraints.

3.3.3. Interleaved. There are many ways to interleave the searches for the action sequence and parameter values. In some cases, we would like to pre-sample state variable values, such as robot configurations and object poses, but defer the computation of motion parameter values, such as collision-free trajectories between two robot configurations. In this case, the domains of the state variables have already been discretized. Conditioned on an assignment of values to every non-motion parameter (state variable) for an action instance, each motion parameter is only affected by the constraints of that particular action, which means that the problem of finding satisfying values for its parameters is independent of finding parameters for other actions. Thus, the existence of a satisfying assignment to the motion parameters can be evaluated *online* during action sequencing in order to only compute values for action instances encountered during the search. The strategy was first applied to TAMP under the name of semantic attachments (62, 63, 64). Although this strategy limits the amount of interleaving that is possible, it is appealing in that the state space is fixed during sequencing;

it only identifies that some transitions are infeasible.

Interleaved action and parameter search can also aid the search for plan skeletons when sequencing first. One source of search control is to observe that, in order for a plan skeleton to admit a satisfying assignment, all of its subsequences must also have satisfying assignments. Thus, a partial plan skeleton can be pruned from the search if its H-CSP is infeasible. Some approaches even solve relaxations of the induced H-CSPs that omit certain constraints such as motion constraints with many decision variables, which are often satisfiable and thus are uninformative (65, 66, 67, 61, 68, 69).

A more general control structure is to perform a tree search, with layers alternating between selecting an action template and sampling parameter values for that action that satisfy the partial skeleton constraints. A substantial difficulty in this approach is that tree nodes may have infinitely many successor nodes due to the possibly infinitely many satisfying parameter values and action instances that could be performed. Thus, it is important that the search be *persistent* (70, 71), in the sense that it will revisit previous search nodes indefinitely in order to generate additional samples for continuous parameters.

3.4. Communication between subproblems

TAMP strategies require solving multiple H-CSP subproblems. These problems often have shared substructure that can be exploited, resulting in substantial reductions in computation. The primary algorithmic question is whether to share information about sets of constraints that *can* be satisfied (*positive*) or about sets of constraints that *cannot* be satisfied (*negative*). Algorithms that satisfy constraints individually typically take the positive approach, and algorithms that satisfy constraints jointly typically take the negative approach. Although we will discuss these approaches separately, it is possible to develop algorithms that use both, possibly to handle different types of constraints.

3.4.1. Positive methods. Positive methods are straightforward: whenever any H-CSP is solved, whether it contains one or many constraints, they add each constraint in the H-CSP, along with its satisfying assignment, to a database of *constraint elements*, known solutions to constraints. For methods that satisfy before sequencing (Section 3.3.2), this database is used to instantiate action instances before sequencing. If action sequencing fails to find a solution, the feedback is that the current database is insufficient and more values must be sampled. Some methods that sequence before satisfying (Section 3.3.1) also use positive feedback. The *focused* algorithm presented in (72, 55, 73) plans using a mixed set of sampled values and free parameters (optimistic values), which represent values that are not yet available, but that might potentially be generated by a sampler.

3.4.2. Negative methods. Alternatively, instead of recording solutions to constraints, an algorithm could identify unsatisfiable *counterexample* H-CSPs. Because any H-CSP that contains an unsatisfiable subproblem is itself unsatisfiable, any H-CSP that contains a recorded counterexample can be pruned; this can, in turn, prevent action sequencing from exploring plan skeletons that are as-yet unexplored, but have the same failure case as a previously explored skeleton. Since most H-CSP solvers are only semi-decision procedures, they can never determine with certainty that a problem is infeasible. One way to handle this problem is to assume unsatisfiability initially, but, as discussed in Section 3.3, allocate a thread to each H-CSP that continually searches for a solution in case one exists. If one of these

threads returns with a solution, the H-CSP is removed from the counterexample set.

A key algorithmic concern here is identifying informative counterexamples. The smaller a counterexample is, the more H-CSPs and thus plan skeletons it can prune. For example, consider the constraint network in Figure 9 and suppose that placement $\mathbf{p_0}$ is on a tall shelf that the robot cannot reach. If we could isolate constraint $\texttt{Kin[A]}(q_1, \mathbf{p_0}, g_2)$ as the bottleneck, rather than the full H-CSP, any plan skeleton that attempts to pick `A` at its initial placement will be pruned, informing the planner that `A` cannot be manipulated.

The problem of identifying small counterexamples can itself be time-consuming, requiring the original H-CSP to be decomposed into smaller H-CSPs, each of which is individually tested for unsatisfiability. Several TAMP approaches have proposed heuristic methods for diagnosing failure and repairing the problem (74, 75, 76, 60). There are also several good domain-independent strategies. For problems in which the state variables are pre-sampled (3.3.3), the remaining H-CSP is often disconnected, and thus each unsatisfiable connected component can be independently added as a counterexample (77, 78, 79). There are methods from the discrete SAT literature that, for unsatisfiable propositional formulas, identify *unsatisfiable cores* (80), small subsets of constraint that cause unsatisfiability. These ideas can be extended to continuous mathematical programs, where real-valued constraint violation feedback can improve the efficiency of the search for counterexamples (81, 82, 83).

3.5. Taxonomy

Table 1 illustrates a representative set of MMMP and TAMP algorithms, categorized in terms of how they solve for continuous parameter values and how they combine searching for the mode-family or task-level structure of a plan with searching for continuous values. This table is meant to provide broad coverage, but is not exhaustive. Each row lists one of three strategies for integrating constraint satisfaction and action sequencing: satisfaction first (Section 3.3.2), interleaved satisfaction and sequencing (Section 3.3.3), and sequencing first (Section 3.3.1). Each column lists one of three strategies for performing constraint satisfaction: assuming the state variables are pre-discretized and solving for motion parameters, individual sampling (Section 3.2.2), and joint optimization (Section 3.2.1).

4. EXTENSIONS

There are many ways to extend the basic TAMP problem class and associated algorithms; these are areas of current active research and future interest.

4.1. Kinodynamic systems

We have focused on domains with quasi-static dynamics (after the robot executes an action, the objects end in a stable state which persists until the robot's next action) and simple rigid-body kinematics. Extending TAMP to handle deformable objects and liquids as well as to full dynamics, such as throwing, are important directions. Several TAMP approaches have already demonstrated the ability to plan for kinodynamic systems (100, 68, 69).

4.2. State and action uncertainty

A critical issue when acting in the real world is uncertainty. In the presence of future-state uncertainty, a planning algorithm might need to take into account multiple possible

	Pre-discretized	Sampling	Optimization
Satisfaction First	Ferrer-Mestres* (84, 85)	Siméon† (22) Hauser† (13, 29, 14) Garrett* (86, 21) Krontiris† (87, 88) Akbari* (89) Vega-Brown† (90)	
Interleaved	Dornhege* (62, 63, 91) Gaschler* (92, 93, 94) Colledanchise* (95)	Gravot* (96, 97) Stilman† (23, 98, 99) Plaku† (100) Kaelbling* (101, 102) Barry† (103, 30, 104) Garrett* (70, 71) Thomason* (105) Kim* (106, 107) Kingston† (108)	Fernandez-Gonzalez* (109)
Sequence First	Nilsson* (2) Erdem* (74, 75) Lagriffoul* (65, 66, 67) Pandey* (110, 111) Lozano-Pérez* (112) Dantam* (77, 78, 79) Lo* (113)	Wolfe* (114) Srivastava* (76, 60) Garrett* (55, 73)	Toussaint* (61, 68, 69) Shoukry* (81, 82, 83) Hadfield-Menell* (115)

Table 1: A table that categorizes MMMP and TAMP approaches, based on how they solve HCSPS and how they integrate with constraint satisfaction with action sequencing. Approaches for MMMP are designated with †, and approaches for TAMP are designated with *. Each table cell is listed chronologically.

outcomes of an action and ensure that there are actions it can take in response, to avoid unlikely but disastrous outcomes. More difficult, but pervasive, is uncertainty about the present state. In this case, the problem can be treated as a "belief-space" planning problem, in which the planner reasons explicitly about the agent's state of information about the world and takes actions both to gain information and to drive the world into a desired belief state. Several approaches for deterministic observable TAMP have been extended to handle these challenges. (102, 116, 117, 118)

4.3. Planning and learning

A critical question to ask is where TAMP models come from. Most work in TAMP assumes perfect observability, control actuation, and knowledge of the kinematics and shape of objects. Machine learning methods can help with the process of acquiring models in non-ideal domains as well as speeding computation. In particular, learning methods can improve TAMP in several ways:

- **Learning models.** Given a controller, whether acquired via learning or hand-built, the constraints that allow us to characterize successful executions for the TAMP planner may not be obvious, but they, too, can be learned from experience (8).
- **Learning search guidance.** Classic task-planning algorithms derive domain-independent search heuristics from the action descriptions, but there are opportu-

nities to automatically learn domain-dependent search heuristics (119, 120), in the form of policies or value function estimates (121) or action-orderings as well (122). Learning search guidance has been hugely influential in games like Go (123). In TAMP problems, it is more difficult because it is much less clear how to encode the state of the problem (object shapes and poses) in a way that affords generalization from current function approximation methods, and because the goal must be encoded into the prediction as well, but there is initial progress in this area (106, 124, 125).

- **Learning sampling guidance.** Many TAMP planners use conditional samplers as part of their strategy for solving underlying H-CSPs. Learning can make sampling much more effective, in two different ways, one in which the learning happens *during* a single search process and one in which the learning happens across problem instances. In the forward-search algorithms that interleave selection of action and parameters, we can derive inspiration from *Monte Carlo tree search* (MCTS) (126, 127), in which experience with trying to expand nodes in a branch of the tree is used to form local estimates of the likelihood that a solution lies along that branch. Sampling for continuous parameter values can itself be similarly guided, using techniques for optimistic global optimization (128, 107). Samplers can also be learned from previous experience using generative models such as Generative adversarial networks (GANs) (129).

SUMMARY POINTS

1. TAMP selects the sequence of high-level actions that the robot should take, the hybrid parameter values that determine how the action is performed, and the low-level motions that safely execute the action.
2. TAMP approaches build on research in motion planning, multi-modal motion planning, and task planning.
3. Many TAMP approaches can be seen as integrating a search over plan skeletons (partially specified plans) and the satisfaction of constraints over hybrid action parameters.
4. Existing approaches can be usefully categorized according to how they address and integrate these two types of decisions.

FUTURE ISSUES

1. Further investigation is needed of strategies that combine sampling and optimization approaches to TAMP.
2. TAMP methods should be extended to plan in more realistic environments that, for example, involve deformable objects, time, dynamics, liquids and other agents.
3. Uncertainty is central to all real-world robot applications; future TAMP methods should consider both future-state and present-state uncertainty.
4. Incorporating learning-based methods into planning will enable planners to reason with learned action models, requiring less human-provided domain knowledge.

DISCLOSURE STATEMENT

The authors are not aware of any affiliations, memberships, funding, or financial holdings that might be perceived as affecting the objectivity of this review.

ACKNOWLEDGMENTS

We gratefully acknowledge support from NSF grants 1523767 and 1723381; from AFOSR grant FA9550-17-1-0165; from ONR grant N00014-18-1-2847; from the Honda Research Institute; and from SUTD Temasek Laboratories. Caelan Garrett, Rachel Holladay, Rohan Chitnis and Tom Silver are supported by NSF GRFP fellowships. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of our sponsors.

LITERATURE CITED

1. Fikes RE, Nilsson NJ. 1971. {STRIPS}: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence* 2
2. Nilsson NJ. 1984. Shakey the Robot. Tech. Rep. 323, Artificial Intelligence Center, SRI International
3. Lozano-Perez T, Wesley MA. 1979. An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles. *Communications of the ACM* 22
4. Kavraki LE, Svestka P, Latombe JC, Overmars MH. 1996. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation* 12
5. LaValle SM, Kuffner JJ. 2001. Randomized kinodynamic planning. *International Journal of Robotics Research* 20
6. Ratliff N, Zucker M, Bagnell JA, Srinivasa S. 2009. CHOMP: Gradient optimization techniques for efficient motion planning. In *IEEE International Conference on Robotics and Automation*
7. Schulman J, Duan Y, Ho J, Lee A, Awwal I, et al. 2014. Motion planning with sequential convex optimization and convex collision checking. *International Journal of Robotics Research* 33
8. Wang Z, Garrett CR, Kaelbling LP, Lozano-Perez T. 2020. Learning compositional models of robot skills for task and motion planning
9. Alami R, Simeon T, Laumond JP. 1990. A geometrical approach to planning manipulation tasks: The case of discrete placements and grasps. In *International Symposium on Robotics Research*
10. Alami R, Laumond JP, Siméon T. 1994. Two manipulation planning algorithms. In *Workshop on Algorithmic Foundations of Robotics*
11. Branicky MS, Curtiss MM. 2002. Nonlinear and hybrid control via rrts. In *Proc. Intl. Symp. on Mathematical Theory of Networks and Systems*, vol. 750
12. Branicky MS, Curtiss MM, Levine J, Morgan S. 2006. Sampling-based planning, control and verification of hybrid systems. *IEE Proceedings-Control Theory and Applications* 153
13. Hauser K, Latombe JC. 2010. Multi-modal Motion Planning in Non-expansive Spaces. *International Journal of Robotics Research* 29
14. Hauser K, Ng-Thow-Hing V, Gonzalez-Baños H. 2011. Randomized multi-modal motion planning for a humanoid robot manipulation task. *International Journal of Robotics Research* 30
15. Ghallab M, Nau DS, Traverso P. 2016. Automated planning and acting. Cambridge University Press
16. Bonet B, Geffner H. 2001. Planning as heuristic search. *Artificial Intelligence* 129

17. Hoffmann J, Nebel B. 2001. The {FF} Planning System: Fast Plan Generation Through Heuristic Search. *Journal Artificial Intelligence Research* 14
18. Helmert M. 2006. The Fast Downward Planning System. *Journal of Artificial Intelligence Research* 26
19. Deshpande A, Kaelbling LP, Lozano-Perez T. 2016. Decidability of Semi-Holonomic Prehensile Task and Motion Planning. *Workshop on Algorithmic Foundations of Robotics*
20. Vendittelli M, Laumond JP, Mishra B. 2015. Decidability of robot manipulation planning: Three disks in the plane. In *Workshop on Algorithmic Foundations of Robotics*
21. Garrett CR, Lozano-Perez T, Kaelbling LP. 2017. FFRob: Leveraging symbolic planning for efficient task and motion planning. *The International Journal of Robotics Research*
22. Siméon T, Laumond JP, Cortés J, Sahbani A. 2004. Manipulation planning with probabilistic roadmaps. *International Journal of Robotics Research* 23
23. Stilman M, Kuffner JJ. 2005. Navigation among movable obstacles: Real-time reasoning in complex environments. *International Journal of Humanoid Robotics* 2
24. King J, Cognetti M, Srinivasa S. 2016. Rearrangement planning using object-centric and robot-centric action spaces. In *IEEE International Conference on Robotics and Automation*
25. Belta C, Bicchi A, Egerstedt M, Frazzoli E, Klavins E, Pappas GJ. 2007. Symbolic planning and control of robot motion [grand challenges of robotics]. *IEEE Robotics & Automation Magazine* 14:61–70
26. Plaku E, Karaman S. 2016. Motion planning with temporal-logic specifications: Progress and challenges. *AI communications* 29:151–162
27. Canny J. 1988. The complexity of robot motion planning. MIT press
28. LaValle SM. 2006. Planning Algorithms. Cambridge University Press
29. Hauser K. 2010. Randomized belief-space replanning in partially-observable continuous spaces. In *Workshop on Algorithmic Foundations of Robotics*
30. Barry J, Kaelbling LP, Lozano-Perez T. 2013. A hierarchical approach to manipulation with diverse actions. In *IEEE International Conference on Robotics and Automation*
31. Stilman M. 2007. Task constrained motion planning in robot joint space. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*
32. Stilman M. 2010. Global manipulation planning in robot joint space with task constraints. *IEEE Transactions on Robotics* 26
33. Berenson D, Srinivasa SS, Ferguson D, Kuffner JJ. 2009. Manipulation planning on constraint manifolds. In *IEEE International Conference on Robotics and Automation*
34. Berenson D, Srinivasa SS. 2010. Probabilistically complete planning with end-effector pose constraints. In *International Conference on Robotics and Automation*
35. Berenson D, Srinivasa S, Kuffner J. 2011. Task space regions: A framework for pose-constrained manipulation planning. *International Journal of Robotics Research* 30
36. Kingston Z, Moll M, Kavraki LE. 2018. Sampling-based methods for motion planning with constraints. *Annual review of control, robotics, and autonomous systems* 1
37. Kingston Z, Moll M, Kavraki LE. 2019. Exploring implicit spaces for constrained sampling-based planning. *International Journal of Robotics Research* 38
38. Alur R, Courcoubetis C, Halbwachs N, Henzinger TA, Ho PH, et al. 1995. The algorithmic analysis of hybrid systems. *Theoretical computer science* 138:3–34
39. Alur R, Henzinger TA, Lafferriere G, Pappas GJ. 2000. Discrete abstractions of hybrid systems. *Proceedings of the IEEE* 88:971–984
40. Henzinger TA. 2000. The theory of hybrid automata. In *Verification of Digital and Hybrid Systems*. Springer, 265–292
41. Wang LC, Chen CC. 1991. A combined optimization method for solving the inverse kinematics problems of mechanical manipulators. *IEEE Transactions on Robotics and Automation* 7:489–499
42. Diankov R. 2010. Automated construction of robotic manipulation programs. Ph.D. thesis,

Robotics Institute, Carnegie Mellon University
43. Karpas E, Magazzeni D. 2019. Automated Planning for Robotics. *Annual Review of Control, Robotics, and Autonomous Systems* 3
44. Bäckström C, Nebel B. 1995. Complexity results for SAS+ planning. *Computational Intelligence* 11
45. McDermott D. 1991. Regression planning. *International Journal of Intelligent Systems* 6
46. Fox M, Long D. 2003. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research* 20
47. Edelkamp S. 2004. PDDL2.2: The language for the classical part of the 4th international planning competition. *4th International Planning Competition (IPC'04), at ICAPS'04*.
48. Fox M, Long D. 2006. Modelling Mixed Discrete-Continuous Domains for Planning. *J. Artif. Intell. Res.* 27
49. Bryce D, Gao S, Musliner DJ, Goldman RP. 2015. SMT-Based Nonlinear PDDL+ Planning. In *AAAI Conference on Artificial Intelligence*
50. Coles AJ, Coles AI, Fox M, Long D. 2012. COLIN: Planning with continuous linear numeric change. *Journal of Artificial Intelligence Research* 44
51. Lin F, Reiter R. 1994. State constraints revisited. *Journal of logic and computation* 4:655–677
52. Thiébaux S, Hoffmann J, Nebel B. 2005. In defense of PDDL axioms. *Artificial Intelligence* 168
53. Dechter R. 1992. Constraint networks. Tech. rep., Information and Computer Science, University of California, Irvine
54. Dechter R. 2003. Constraint processing. Morgan Kaufmann
55. Garrett CR, Lozano-Perez T, Kaelbling LP. 2018. Sampling-based methods for factored task and motion planning. *The International Journal of Robotics Research* 37
56. Mandalika A, Choudhury S, Salzman O, Srinivasa S. 2019. Generalized lazy search for robot motion planning: Interleaving search and edge evaluation via event-based toggles. In *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 29
57. Bohlin R, Kavraki LE. 2000. Path planning using lazy {PRM}. In *IEEE International Conference on Robotics and Automation (ICRA)*, vol. 1. IEEE
58. Dellin CM, Srinivasa SS. 2016. A Unifying Formalism for Shortest Path Problems with Expensive Edge Evaluations via Lazy Best-First Search over Paths with Edge Selectors. *International Conference on Automated Planning and Scheduling (ICAPS)*
59. Bacchus F, Yang Q. 1994. Downward refinement and the efficiency of hierarchical problem solving. *Artificial Intelligence* 71
60. 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 *IEEE International Conference on Robotics and Automation*
61. Toussaint M. 2015. Logic-geometric programming: an optimization-based approach to combined task and motion planning. In *International Joint Conference on Artificial Intelligence*
62. Dornhege C, Eyerich P, Keller T, Trüg S, Brenner M, Nebel B. 2009. Semantic Attachments for Domain-Independent Planning Systems. In *International Conference on Automated Planning and Scheduling*
63. Dornhege C, Gissler M, Teschner M, Nebel B. 2009. Integrating Symbolic and Geometric Planning for Mobile Manipulation. In *IEEE International Workshop on Safety, Security and Rescue Robotics*
64. Dornhege C. 2015. Task planning for high-level robot control. Ph.D. thesis, Verlag nicht ermittelbar
65. Lagriffoul F, Dimitrov D, Saffiotti A, Karlsson L. 2012. Constraint propagation on interval bounds for dealing with geometric backtracking. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*
66. Lagriffoul F, Dimitrov D, Bidot J, Saffiotti A, Karlsson L. 2014. Efficiently combining task

and motion planning using geometric constraints. *International Journal of Robotics Research*
67. Lagriffoul F, Andres B. 2016. Combining task and motion planning: A culprit detection problem. *The International Journal of Robotics Research* 35
68. Toussaint M, Lopes M. 2017. Multi-bound tree search for logic-geometric programming in cooperative manipulation domains. In *IEEE International Conference on Robotics and Automation*
69. Toussaint M, Allen K, Smith K, Tenenbaum JB. 2018. Differentiable physics and stable modes for tool-use and manipulation planning. *Proc. of Robotics Science & Systems*
70. Garrett CR, Lozano-Perez T, Kaelbling LP. 2015. Backward-Forward Search for Manipulation Planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*
71. Grey MX, Garrett CR, Liu CK, Ames AD, Thomaz AL. 2016. Humanoid Manipulation Planning using Backward-Forward Search. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 2016-Novem
72. Garrett CR, Lozano-Perez T, Kaelbling LP. 2017. Sample-Based Methods for Factored Task and Motion Planning. In *Robotics: Science and Systems*, vol. 13
73. Garrett CR, Lozano-Perez T, Kaelbling LP. 2020. PDDLStream: Integrating Symbolic Planners and Blackbox Samplers. In *International Conference on Automated Planning and Scheduling*
74. 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 *IEEE International Conference on Robotics and Automation*
75. Erdem E, Patoglu V, Saribatur ZG. 2015. Integrating hybrid diagnostic reasoning in plan execution monitoring for cognitive factories with multiple robots. In *IEEE International Conference on Robotics and Automation*
76. Srivastava S, Riano L, Russell S, Abbeel P. 2013. Using Classical Planners for Tasks with Continuous Operators in Robotics. In *ICAPS Workshop on Planning and Robotics*
77. Dantam NT, Kingston Z, Chaudhuri S, Kavraki LE. 2016. Incremental Task and Motion Planning: A Constraint-Based Approach. In *Robotics: Science and Systems*
78. Dantam NT, Kingston ZK, Chaudhuri S, Kavraki LE. 2018. An incremental constraint-based framework for task and motion planning. *The International Journal of Robotics Research* 37
79. Dantam NT, Chaudhuri S, Kavraki LE. 2018. The Task-Motion Kit: An Open Source, General-Purpose Task and Motion-Planning Framework. *IEEE Robotics & Automation Magazine* 25
80. Liffiton MH, Sakallah KA. 2008. Algorithms for computing minimal unsatisfiable subsets of constraints. *Journal of Automated Reasoning* 40
81. Shoukry Y, Nuzzo P, Saha I, Sangiovanni-Vincentelli AL, Seshia SA, et al. 2016. Scalable lazy SMT-based motion planning. In *IEEE Conference on Decision and Control*
82. Shoukry Y, Nuzzo P, Sangiovanni-Vincentelli AL, Seshia SA, Pappas GJ, Tabuada P. 2017. SMC: Satisfiability modulo convex optimization. In *International Conference on Hybrid Systems: Computation and Control*
83. Shoukry Y, Nuzzo P, Sangiovanni-Vincentelli AL, Seshia SA, Pappas GJ, Tabuada P. 2018. SMC: Satisfiability Modulo Convex Programming. *Proceedings of the IEEE* 106
84. Ferrer-Mestres J, Francès G, Geffner H. 2017. Combined Task and Motion Planning as Classical AI Planning. *arXiv preprint arXiv:1706.06927*
85. Ferrer-Mestres J, Es G, Geffner H. ???? Planning with State Constraints and its Application to Combined Task and Motion Planning
86. Garrett CR, Lozano-Perez T, Kaelbling LP. 2014. FFRob: An efficient heuristic for task and motion planning. In *Workshop on the Algorithmic Foundations of Robotics*
87. Krontiris A, Bekris KE. 2015. Dealing with Difficult Instances of Object Rearrangement. In *Robotics: Science and Systems*
88. Krontiris A, Bekris KE. 2016. Efficiently Solving General Rearrangement Tasks: A Fast Extension Primitive for an Incremental Sampling-based Planner. In *International Conference on*

Robotics and Automation
89. Akbari A, Rosell J. 2016. Task planning using physics-based heuristics on manipulation actions. In *International Conference on Emerging Technologies and Factory Automation*
90. Vega-Brown W, Roy N. 2016. Asymptotically optimal planning under piecewise-analytic constraints. In *Workshop on the Algorithmic Foundations of Robotics*
91. Dornhege C, Hertle A, Nebel B. 2013. Lazy Evaluation and Subsumption Caching for Search-Based Integrated Task and Motion Planning. In *IROS Workshop on AI-based robotics*
92. Gaschler A, Petrick RPA, Giuliani M, Rickert M, Knoll A. 2013. KVP: A knowledge of volumes approach to robot task planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*
93. Gaschler A, Kessler I, Petrick RPA, Knoll A. 2015. Extending the Knowledge of Volumes approach to robot task planning with efficient geometric predicates. In *IEEE International Conference on Robotics and Automation*
94. Gaschler A, Petrick RPA, Khatib O, Knoll A. 2018. KABouM: Knowledge-level action and bounding geometry motion planner. *Journal of Artificial Intelligence Research*
95. Colledanchise M, Almeida D, Ögren P. 2019. Towards blended reactive planning and acting using behavior trees. In *International Conference on Robotics and Automation*
96. Gravot F, Cambon S, Alami R. 2005. aSyMov: a planner that deals with intricate symbolic and geometric problems. In *International Symposium on Robotics Research*
97. Cambon S, Alami R, Gravot F. 2009. A hybrid approach to intricate motion, manipulation and task planning. *International Journal of Robotics Research* 28
98. Stilman M, Kuffner JJ. 2006. Planning Among Movable Obstacles with Artificial Constraints. In *Workshop on Algorithmic Foundations of Robotics (WAFR)*
99. Stilman M, Schamburek JU, Kuffner JJ, Asfour T. 2007. Manipulation Planning Among Movable Obstacles. In *IEEE International Conference on Robotics and Automation*
100. Plaku E, Hager G. 2010. Sampling-based Motion Planning with Symbolic, Geometric, and Differential Constraints. In *IEEE International Conference on Robotics and Automation*
101. Kaelbling LP, Lozano-Perez T. 2011. Hierarchical task and motion planning in the now. In *ICRA*
102. Kaelbling LP, Lozano-Perez T. 2013. Integrated task and motion planning in belief space. *International Journal of Robotics Research (IJRR)*
103. Barry J, Hsiao K, Kaelbling LP, Lozano-Perez T. 2012. Manipulation with Multiple Action Types. In *Int. Symp. on Experi. Robotics*
104. Barry JL. 2013. Manipulation with diverse actions. Ph.D. thesis, Massachusetts Institute of Technology
105. Thomason W, Knepper RA. 2019. A unified sampling-based approach to integrated task and motion planning. In *International Symposium on Robotics Research*
106. Kim B, Shimanuki L. 2019. Learning value functions with relational state representations for guiding task-and-motion planning. In *Conference on Robot Learning*
107. Kim B, Lee K, Lim S, Kaelbling LP, Lozano-Perez T. 2020. Monte Carlo Tree Search in continuous spaces using Voronoi optimistic optimization with regret bounds. *AAAI Conference on Artificial Intelligence*
108. Kingston Z, Wells AM, Moll M, Kavraki LE. 2020. Informing Multi-Modal Planning with Synergistic Discrete Leads. In *2016 IEEE International Conference on Robotics and Automation*
109. Fernández-González E, Williams B, Karpas E. 2018. Scottyactivity: Mixed discrete-continuous planning with convex optimization. *Journal of Artificial Intelligence Research* 62
110. Pandey AK, Saut JP, Sidobre D, Alami R. 2012. Towards Planning Human-Robot Interactive Manipulation Tasks. In *RAS/EMBS International Conference on Biomedical Robotics and Biomechatronics*
111. de Silva L, Pandey AK, Gharbi M, Alami R. 2013. Towards Combining {HTN} Planning and Geometric Task Planning. In *RSS Workshop on Combined Robot Motion Planning and AI*

Planning for Practical Applications
112. Lozano-Perez T, Kaelbling LP. 2014. A Constraint-Based Method for Solving Sequential Manipulation Planning Problems. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*
113. Lo SY, Zhang S, Stone P. 2018. PETLON: planning efficiently for task-level-optimal navigation. In *International Conference on Autonomous Agents and MultiAgent Systems*
114. Wolfe J, Marthi B, Russell S. 2010. Combined Task and Motion Planning for Mobile Manipulation. In *International Conference on Automated Planning and Scheduling*
115. Hadfield-Menell D, Lin C, Chitnis R, Russell S, Abbeel P. 2016. Sequential Quadratic Programming for Task Plan Optimization. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*. IEEE
116. Hadfield-Menell D, Groshev E, Chitnis R, Abbeel P. 2015. Modular task and motion planning in belief space. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*
117. Phiquepal C, Toussaint M. 2019. Combined task and motion planning under partial observability: An optimization-based approach. In *2019 International Conference on Robotics and Automation (ICRA)*. IEEE
118. Garrett CR, Paxton C, Lozano-Perez T, Kaelbling LP, Fox D. 2020. Online Replanning in Belief Space for Partially Observable Task and Motion Problems. In *International Conference on Robotics and Automation*
119. Yoon SW, Fern A, Givan R. 2006. Learning Heuristic Functions from Relaxed Plans. In *ICAPS*
120. Shen W, Trevizan F, Thiébaux S. 2020. Learning domain-independent planning heuristics with hypergraph networks. In *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 30
121. Yoon S, Fern A, Givan R. 2008. Learning control knowledge for forward search planning. *The Journal of Machine Learning Research* 9:683–718
122. Garrett CR, Kaelbling LP, Lozano-Perez T. 2016. Learning to Rank for Synthesizing Planning Heuristics. In *Int. Joint Conf. on Artificial Intelligence (IJCAI)*, vol. 2016-Janua
123. Silver D, Huang A, Maddison CJ, Guez A, Sifre L, et al. 2016. Mastering the game of go with deep neural networks and tree search. *Nature* 529
124. Driess D, Oguz O, Ha JS, Toussaint M. 2020. Deep Visual Heuristics: Learning Feasibility of Mixed-Integer Programs for Manipulation Planning
125. Chitnis R, Hadfield-Menell D, Gupta A, Srivastava S, Groshev E, et al. 2016. Guided search for task and motion plans using learned heuristics. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE
126. Kocsis L, Szepesvári C. 2006. Bandit based {M}onte-{C}arlo planning. In *European Conf. on Machine Learning*
127. Browne C, Powley EJ, Whitehouse D, Lucas SM, Cowling PI, et al. 2012. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games* 4
128. Munos R. 2014. From bandits to monte-carlo tree search: The optimistic principle applied to optimization and planning. *Foundations and Trends in Machine Learning* 7
129. Kim B, Kaelbling LP, Lozano-Perez T. 2018. Guiding search in continuous state-action spaces by learning an action sampler from off-target search experience. In *AAAI Conference on Artificial Intelligence*